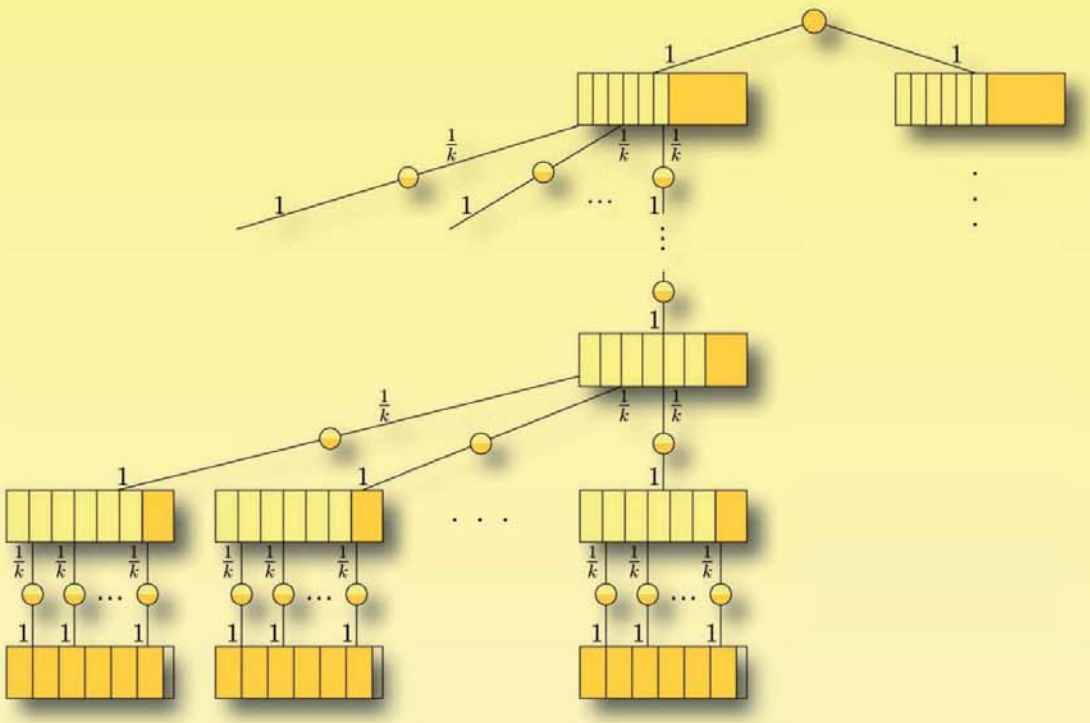
Andreas Wiese

Packet Routing and Scheduling



Cuvillier Verlag Göttingen
Internationaler wissenschaftlicher Fachverlag

Packet Routing and Scheduling

vorgelegt von
Diplom-Mathematiker
Andreas Wiese
aus Berlin

Von der Fakultät II - Mathematik und Naturwissenschaften
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
– Dr. rer. nat. –

genehmigte Dissertation

Vorsitzender:	Prof. Dr. Volker Mehrmann
Berichter:	Prof. Dr. Martin Skutella
	Prof. Dr. David P. Williamson

Tag der wissenschaftlichen Aussprache: 13. April 2011

Berlin, 2011
D 83

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie; detaillierte bibliographische Daten sind im Internet über http://dnb.d-nb.de abrufbar.
1. Aufl. - Göttingen: Cuvillier, 2011
Zugl.: (TU) Berlin, Univ., Diss., 2011

978-3-86955-827-1

Nonnenstieg 8, 37075 Göttingen
Telefon: 0551-54724-0
Telefax: 0551-54724-21
www.cuvillier.de

1. Auflage 2011
Gedruckt auf säurefreiem Papier.

978-3-86955-827-1

Acknowledgments

First of all, I would like to thank my advisor Martin Skutella for his invaluable support and guidance for my research and other academical questions. It has been wonderful working with him and I highly appreciate having benefited from his experience in scheduling and approximation algorithms. In particular, I am grateful for the freedom of pursuing research topics that I am interested in. Also, I am thankful for having had the opportunity to join the academic research community in several conferences and workshops. I am also very grateful to Britta Peis for introducing me to the packet routing problem when I joined the COGA group in spring 2008 and for collaborating with me during the entire time of my PhD. I would like to thank her for lots of advice and for her commitment in our joint projects, in particular when she was on parental leave. I also want to thank David P. Williamson who agreed to take the second assessment for this thesis.

During my research, I spent a lot of time with José Verschae discussing ideas for proofs and research questions. I am very grateful for that. My thank goes to Sebastian Stiller who introduced me to the flow scheduling problem which I find very interesting and which turned out to be a very fruitful research questions. Also, I want to thank Christina Büsing, Wiebke Höhn, Torsten Gellert, Martin Groß, Jan-Philipp Kappmeier, Jannik Matuschke, Britta Peis, Madeleine Theile, and Wolfgang Welz who agreed to proof-read parts of this thesis. Finally, I would like to thank all COGA members for providing a very lively research environment here at the TU Berlin.

Since no mathematician can live on proofs alone, I would like to thank the TU Berlin, the Cusanuswerk, and the DFG for providing me funding for my thesis and the Berlin Mathematical School (BMS) for travel support. Also, I thank my BMS mentor Christian Haase for helpful discussions and advice.

Apart from all academical help I would like to thank my parents and my family who supported me all the time. Also, my gratefulness goes to Anna for her support and her trust in me. I want to thank all my friends for listening, spending time with me, and simply for being there.

Last but not least, I would like to thank the developers of LyX for creating a really great software!

Berlin, February 2011 *Andreas Wiese*

Contents

Introduction

In combinatorial optimization one wants to find the best solution among many possible choices. The set of candidates is usually given implicitly by the input data and often has exponential size in comparison to the input. The aim is to construct an efficient algorithm which computes the best solution or a solution of provably good quality. A very important topic in combinatorial optimization is scheduling. It treats the assignment of limited resources to activities. In the actual applications, the resources can be very diverse, e. g., machines, money, gasoline, teachers, cars, watchmen, etc. Examples for the many imaginable activities are producing industrial goods, guarding a building, delivering parcels, executing a computer program, working on a project, or the classes of a university. The goal is to assign the resources to the activities in order to optimize some performance measure. Such a measure could be the time when the last activity finishes or the total delay of all activities.

The problems studied in this thesis can be understood as *machine scheduling problems*. In machine scheduling, one is usually given a set of jobs with certain processing times which need to be assigned to some given machines. One has to compute a schedule such that each job is assigned and each machine processes at most one job at a time. Often, additional constraints are present. For instance, jobs could have a deadline by which they have to be finished, some jobs might not be able to start before some others have finished (precedence constraints), or some jobs might not be available before a certain given time (release dates).

A broad range of applications can be understood as machine scheduling problems. A simple example is the manufacturing process of industrial goods in a factory. The jobs model the manufacturing steps which need to be done on the different processing units. The latter are modeled by the machines. However, also the organization of a garden party fits into the machine scheduling framework. The preparation steps (putting on the barbecue, prepare salads, invite friends, etc.) are modeled by jobs whereas the organizers are modeled by the machines. Finally, finding a timetable for the classes of a university and their assignment to lecture theaters can also be formalized as a machine scheduling problem.

For evaluating a computed schedule we need a suitable quality measure. This measure can vary significantly depending on the actual application. For example, two common objective functions in machine scheduling are the *sum of weighted completion times* and the *makespan*. For applications like the pro-

duction of cars in a factory it is desirable to finish as many units as early as possible. To this end, a good objective function is the sum of the completion times of all jobs. If some jobs are more important than others one can additionally give each job a weight which reflects its significance. The resulting objective function is then the sum of the weighted completion times. When scheduling the work of a project or for the mentioned garden party one might rather want that the overall schedule finishes as early as possible. The time when the last job finishes is defined as the *makespan* of a schedule which is a good objective function in this case. In other settings such as time-tabling there might be no objective function necessary and one only wants to compute *some* feasible schedule. However, in such settings it is likely that there are also some not strictly necessary but still desirable "soft" constraints which can be modeled by a suitable objective function.

One very fundamental scheduling problem is the *packet routing problem* (even though in the literature it is usually referred to as a routing problem, see [5, 65, 92], it can also be understood as a scheduling problem). In computer networks like the internet huge amounts of data need to be transported. In particular, due to applications like video-streaming and Voice over IP (VoIP) the internet traffic has grown significantly in the last few years. In computer networks, the transported data is split into packets. Those form the atoms of the network communication. Each packet needs to be transported from its origin to its destination. Since the bandwidth of the communication links is limited, the network might need to delay packets. This results in a scheduling problem. The packets form jobs which need to be "processed" by the links on its path. The limited bandwidth is captured by the fact that each link/machine can process only one packet (or maybe some constant number of packets) at a time.

In some scheduling applications jobs are created repeatedly. For instance, consider an on-board computer of a modern airplane. In a repeated fashion, the computer executes jobs like checking the altitude, operating the auto-pilot, etc. For flight safety it is crucial that each operation finishes before its deadline. Already a small delay in a critical operation might endanger the aircraft and its passengers. Therefore, one needs a (mathematical) proof that a system operates according to its specifications at all times. The area of *real-time scheduling* provides the mathematical foundation for such proofs. The operations are modeled by *tasks* which continuously generate new jobs. Since the schedule is assumed to run infinitely long one cannot compute it explicitly. Instead, one is interested in *scheduling policies* which decide at every point in time which jobs are executed on the machines. Given such a scheduling policy one needs to prove that every job which is ever created meets its deadline.

Almost all problems which we consider in this thesis are NP-hard. Since it is widely believed that $P \neq NP$, there is not much hope for efficient (polynomial time) algorithms which solve our problems exactly. Therefore, we relax the aim of always finding an optimal solution and search for *approximation algorithms* instead. For this thesis, an α-approximation algorithm is a polynomial time algorithm which computes solutions whose objective values differ by at most a factor of α from the respective optimum. We refer to α as the *approximation factor* or *performance ratio* of an algorithm.

At first glance, approximation algorithms might not seem to be relevant for practical applications. For instance, a solution whose value differs by a factor of two from the optimum is by far not satisfying if it has an impact on a budget of thousands of euros. However, the performance ratio of an algorithm is always based on a worst-case analysis. Hence, there could be (and usually there are) many instance of the respective problem where the algorithm performs much better than in the worst-case scenario. Also, for exact methods like IP-solvers it is often useful to have primal solutions of good quality which help pruning the branch-and-bound tree. Apart from being heuristics that one could use directly, approximation algorithms usually yield important structural properties of the problem. These properties can be exploited in approaches to solve the problem exactly (with more computational effort). A good example is our work on the periodic maintenance problem which we present in Chapter 6. We designed approximation algorithms for the several settings of the problem, especially for the practical relevant harmonic case with pairwise dividing period lengths. The gained structural insights then allowed us to develop an IP-formulation for the problem which was able to solve all instances of our industrial partner optimally [28]. Straightforward approaches without the additional insights failed to solve instances of real-world size.

Outline of the Thesis

In Part I of this thesis we study the packet routing problem. As described above, it is a very fundamental question in computer networks. However, it is by far not fully understood theoretically. The best known approximation algorithm produces a schedule whose length is bounded by $O(C + D)$ [66]. The congestion C and the dilation D form the two natural lower bounds on the length of each schedule. One can argue that with the constructive proof for the Lovász Local Lemma (LLL) [79] the proof by Scheideler [92] yields an algorithm computing schedules which finish after at most $39(C + D)$ steps. The above results almost completely rely on the LLL which – due to its generality – cannot make use of the entire structure of the problem. In this work, we make a step towards a better theoretical understanding of the important properties of the problem. We present approximation algorithms, bounds on the makespan of optimal schedules, and complexity results. In particular, we show how structural insights help improving the above LLL-based results.

Chapter 1: The general packet routing problem is very complex. However, when the underlying graph topology is well-structured – like a tree – one can design better and simpler algorithms. In this chapter, we study the packet routing problem on instances where the given graph is a tree. The gained insights will be very useful later in Chapter 2 where we study general graphs. We show that the straightforward farthest-destination-first (FDF) algorithm has an arbitrarily large approximation factor, even on directed trees. Hence, more sophisticated algorithmic methods are needed. For undirected trees we present

a 2-approximation algorithm. For directed trees we derive structural properties which allow us to design an algorithm computing direct schedules (which delay packets only in their respective origins) which finish within $C + D - 1$ steps. Hence, this algorithm is also a 2-approximation algorithm, but it yields a much better performance guarantee if $C \gg D$ or $D \gg C$. Finally, we derive a general condition for the existence of a direct schedule with a certain makespan, even for general directed graphs. We published the results of this chapter in [84].

Chapter 2: We broaden our scope to general graphs. It is known that there is always a schedule which finishes within $39(C+D)$ steps [92]. No improvement on this bound has been made in more than 10 years, even though it is a very natural and important question. We improve and generalize the previous results for the problem. First, we improve the above bound to $23.4(C + D)$. Moreover, we generalize the problem by allowing arbitrary bandwidths and arbitrary transit times for the edges. If every link in the network has at least a certain bandwidth and/or transit time we prove even better bounds. For example, if every link has a transit time of at least 63 we obtain a bound of $4.32(C + D)$ for the makespan of the optimal schedule. We derive these results with a novel framework. Once one has established a good bound for the makespan of optimal schedules for instances with small dilation (i. e., each packet travels only along a small number of edges) our framework implies a bound for *all* instances. For deriving a bound for instances with small dilation we use the insights gained in Chapter 1. Moreover, our framework has the potential to give even better bounds using further insights for such instances. See [88] for our conference paper which contains these results. Due to the mentioned recent result of Moser and Tardos [79] we even obtain an algorithm which computes a schedules with the above bounds.

Chapter 3: After having studied algorithms for the packet routing problem, in this chapter we analyze its complexity. It is known that the problem is NP-hard (implicitly in [25]). We show that it is even NP-hard to approximate the problem with a factor better than $6/5$. In particular, this rules out the existence of a PTAS, assuming that $P \neq NP$. Even more, we prove the latter also for the special case that the underlying graph is a directed tree. For this very restricted graph class we show that the existence of an approximation algorithm with a factor better than $8/7$ implies $P = NP$. This surprising result underlines the difficulty of the packet routing problem since most NP-hard optimization problems (e. g., Coloring, Independent Set, etc.) become polynomial time solvable if one restricts the input graphs to trees. The results of this chapter were published in [84].

Chapter 4: So far we studied the packet routing problem only in the static setting where a finite set of packets needs to be transported to their respective destinations. However, in applications like live video-streams or Voice Over IP (VoIP) usually arbitrarily many packets are created repeatedly. This requires a different model. Therefore, in this chapter we study the *periodic packet routing problem*. The input consists of tasks (rather than single packets) which continuously generate new packets. We assume an infinite time horizon. Hence, we cannot compute the actual schedule explicitly. Instead, we need to design

scheduling policies which define the prioritization of the packets at runtime. We study two paradigms for these policies, *template schedules* and *priority schedules*. Priority schedules are an adaption of rate-monotonic schedules as they are known in classical real-time scheduling. Template schedules are a class of schedules which are especially designed for periodic packet routing. We give a comprehensive characterization of these paradigms. We present algorithms which compute schedules of the respective types and prove (almost) matching lower bounds for the potential of the two paradigms. We refer to [86] for our paper with the results of this chapter.

In Part II of this thesis, we study three other scheduling problems, the flow scheduling problem, the periodic maintenance problem, and finally the problem of scheduling jobs on unrelated machines.

Chapter 5: Dynamic flows and scheduling are two important areas of combinatorial optimization which in practice often arise in a combined manner. However, in theory both problems are mostly treated separately. Better practical solutions need thorough theoretical knowledge. Therefore, in this chapter we make a step towards a better understanding of the interaction of the two areas by studying the flow scheduling problem. In that problem, we want to transport items (jobs) through a network by a dynamic flow from a source to a sink. The objective is to minimize the sum of weighted arrival (completion) times at the sink. We first establish the connection between the routing and the scheduling aspect of this problem. Then we show that the scheduling part of the problem reduces to the problem of scheduling jobs on a single machine whose speed might increase over time (the objective is still to minimize the sum of weighted completion times). We treat this interesting problem in its own right. In contrast to the case where the speed of the machine stays constant, the natural Smith's Rule algorithm is not always optimal in this setting. However, we show that it is exactly a $\big((\sqrt{3}+1)/2\big)$-approximation algorithm. Usually in approximation algorithms, one establishes a lower bound against which one compares the value of the computed solution. Here we pursue a different approach. We constructively characterize properties of worst-case instances. We do this so precisely, that finally computing the worst-case approximation ratio of Smith's Rule on these instances reduces to basic calculus. This procedure gives us the exact approximation ratio of the algorithm and a family of tight examples. In addition, we generalize the PTAS for the unit speed case with release dates [3] to our problem with release dates. Studying the online setting, we show that here Smith's Rule is still a 2-approximation algorithm. We round up the picture by showing certain other helpful properties of the problem which yield some polynomial time solvable special cases. The results of this chapter have been published in [102].

Chapter 6: As part of our collaboration with our industrial partner, a major avionics company, we study the periodic maintenance problem (PMP). As described above, in a modern aircraft the flight control is to a great extent automatically done by the on-board computer. Therefore, it is crucial that the computer works according to its specifications at all times. The computer programs are modeled by tasks which periodically need processing time on the

processors that they are assigned to. The scheduling rule is very conservative: Once a task has released a new job, this job has to be processed immediately without any delay or preemption. The only degrees of freedom are the assignment of the tasks to processors and the initial time-offsets for the tasks. Even though real-time scheduling is a very active field of research, this particular problem has not been studied much. It arises at our industrial partner and is likely to appear in similar settings where very conservative scheduling rules are necessary. So far, it lacked the theoretical understanding of the properties which are needed for real-world solutions. In this chapter, we study the problem from a theoretical perspective and give a comprehensive characterization of its approximation landscape in the various settings. In particular, we study the practical relevant case of harmonic periods where the period lengths of the tasks divide each other pairwise. This setting can be understood as a generalization of the well-known Bin-Packing problem. Therefore, studying the problem is not only interesting from a practical point of view but also yields interesting theoretical insights. It turns out that important algorithmic properties of Bin-Packing do not generalize to the PMP. Among other results, using involved analytic tools we present a 2-approximation algorithm for the harmonic case and a tightly matching non-approximability result. As mentioned above, the gained insights for the harmonic case helped us to design a sophisticated IP-formulation for the problem which was very successful in practice [28]. See [27] for our publication on the theoretical aspects of the problem presented in this chapter.

Chapter 7: One of the most prominent open problems in scheduling is to determine the best possible approximation factor for scheduling jobs on unrelated machines to minimize the makespan. In contrast, for parallel and related machines approximation schemes and matching NP-hardness results have been known for a long time [52, 53]. The best possible approximation factor for unrelated machines is still not clear. Since they capture a very general setting, settling this question is a very important problem. In a seminal work, Lenstra et al. [69] developed an LP-based 2-approximation algorithm for the problem. On the other hand, they proved that the problem is NP-hard to approximate with a factor of $3/2 - \varepsilon$ for any $\varepsilon > 0$. This gap between 2 and 3/2 has not been diminished in more than 20 years[1] even though the problem is considered to be very important, see e. g. [95]. One natural way to approach the problem is to strengthen the linear program by adding suitable inequalities. A linear program which already implicitly contains a huge class of such inequalities is the configuration-LP. Recently, Svennson [104] showed that the configuration-LP has an integrality gap of $33/17 < 2$ for the restricted assignment case of the problem. However, we show that for the general case this is not true. We prove that in general the configuration-LP has an integrality gap of 2. Even more, we show that this is still true if we require that each job can be assigned to at most two machines (unrelated graph balancing). This is an indicator that the core difficulty of the problem lies in the unrelated graph bal-

[1] The only improvement is a slightly better rounding procedure for the LP by Lenstra et al. that yields a $2 - 1/m$ approximation algorithm [96].

ancing case rather than in the restricted assignment case. Then we study the closely related MaxMin-allocation problem where one wants to maximize the minimum machine load (rather than minimizing the maximum machine load as before). This objective can be understood as establishing a fairness condition for some given agents/machines. For that problem, it is known that in the unrelated graph balancing case the configuration-LP has an integrality gap of 2. However, solving the configuration-LP is very difficult and results only in a computationally very costly $(2+\varepsilon)$-approximation algorithm. In contrast, we give a purely combinatorial 2-approximation algorithm with a running time of $O(n^2)$. Our approximation factor is best possible, unless $P = NP$.

Most results of this thesis have already been published in conference proceedings [27, 84, 86, 88, 102, 106]. Moreover, as part of our research on the packet routing problem we published three further conference papers [60, 85, 87]. However, those results are beyond the scope of this thesis. Likewise, the computational aspects of the periodic maintenance problem were published in [28] but are not covered here.

Throughout this thesis, we assume knowledge of basic concepts of combinatorial optimization such as graphs, algorithms, NP-completeness, etc. For an introduction see e. g. [61]. Most algorithms presented in this thesis are approximation algorithm. For introductions to approximation algorithms see [51, 105, 109].

Part I

Packet Routing

Chapter 1

Trees and Direct Schedules

1.1 Introduction

Today's computer networks transport huge quantities of data. In particular, in the internet huge files can be copied from one end of the world to the other within seconds. Even though the communication links in a network, e. g., optical fiber or Ethernet cable, have the capability to transfer lots of information within very small time spans, their bandwidth is limited. In particular, if the data which needs a specific link at some point in time exceeds the bandwidth of the link, some data has to be buffered and is hence delayed.

In computer networks, the data that is transmitted is subdivided into *packets*. Each packet has a source (the sending entity) and a destination (the receiver) and travels from its source to its destination. Every communication link has a limited bandwidth and hence sometimes packets have to be delayed. Deciding what packets have to wait if the bandwidth of a link is not sufficient is a non-trivial task. Some packets might have to wait for their subsequent connection anyway and can thus safely be delayed. Other packets might need to be transported straight away to avoid a collision with other packets at some later link. In particular, little changes of the schedule in some part of the network might have consequences for some other distant parts of the network.

Our goal is to determine a *schedule* which defines how the packets move through the network along their paths (obeying the bandwidths of the links at all times). The schedule should ensure that the packets reach all their respective destinations as early as possible. Hence, in our model, the objective is to minimize the *makespan*, i. e., the time when the last packet reaches its destination. This results in the *packet routing problem*. We assume that the paths of the packets are given in advance and are unchangeable. We do so because we want to design algorithms for computing schedules which do not rely on any specific properties of the paths (e. g., being shortest paths).

The results presented in this chapter are joint work with Britta Peis and Martin Skutella [84].

1.1.1 Problem Definition

Now we define the packet routing problem formally. Let $G = (V, E)$ be a directed or undirected graph. A packet $M_i = (s_i, t_i, P_i)$ is a tupel consisting of a start vertex $s_i \in V$, a destination vertex $t_i \in V$, and a path $P_i \subseteq E$ from s_i to t_i. Let $\mathcal{M} = \{M_1, M_2, M_3, ..., M_N\}$ be a set of packets. We call $I = (G, \mathcal{M})$ an instance of the *packet routing problem*.

The goal is to find a *packet routing schedule* for I, i. e., a schedule which defines when the packets move along the edges of their respective paths. We assume that time is discrete and that all packets take their steps simultaneously. A schedule is *feasible* if it obeys the following properties:

- each packet M_i follows its path P_i from s_i to t_i,
- each edge e is used by at most one packet at a time, and
- each packet needs one timestep to traverse each edge e.

Packets are allowed to wait on intermediate nodes. If the underlying graph is directed, we require that every path using an edge e traverses it obeying the orientation of e. The objective is to minimize the makespan, i. e., the time when the last packet has reached its destination vertex. See Figure 1.1 for an example. For each packet M_i we define D_i to be the length of P_i, i. e., $D_i := |P_i|$. The *dilation* D is defined by $D := \max_i D_i$. Clearly, D is a lower bound for the length of an optimal schedule. For each edge e we define C_e to be the number of paths that use e. We define the *congestion* C by $C := \max_e C_e$. Then also C is a lower bound on the length of an optimal schedule. We say a schedule is *direct* if every packet waits only in its start vertex (and its destination vertex) but on no intermediate vertex.

We will use the notation $|S|$ for the length of a schedule S. For a packet routing instance I let $OPT\,(I)$ denote a schedule with minimum makespan. For an algorithm $\mathcal{A}$ for the packet routing problem denote by $\mathcal{A}(I)$ the schedule computed by $\mathcal{A}$ for the instance I. The algorithm $\mathcal{A}$ is an α-approximation algorithm if it runs in polynomial time and for all instances I it holds that $|\mathcal{A}(I)| \leq \alpha \cdot |OPT(I)|$. We call α the *approximation ratio* or *performance ratio* of $\mathcal{A}$.

1.1.2 Related Work

The packet routing problem and related problems are widely studied in the literature. In a celebrated paper Leighton et al. show that there is always a routing schedule that finishes in $O(C + D)$ steps [65]. Leighton, Maggs, and Richa present an algorithm that finds such a schedule in polynomial time [66]. However, this algorithm is not suitable for practical applications since the hidden constants in the schedule length are very large (the same applies to the $O(C+D)$ existence-result in [65]). Scheideler [92] improves the non-constructive result by proving that there is always a schedule with a makespan of at most $39(C + D)$.

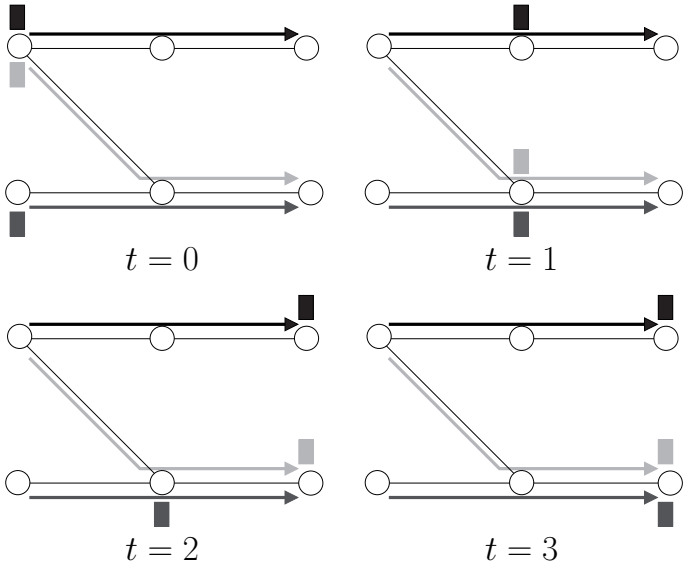


Figure 1.1: An example for a packet routing instance and a schedule for it. Note that at time $t = 1$ on the middle bottom vertex the light gray and the dark gray packet need to use same edge. Hence, one of the packets has to be delayed (the dark gray packet in this example). The depicted schedule has a makespan of 3.

There are various "constant factor approximation"-results in the area of packet routing which are based on the fact that a constant factor approximation for packet routing with given paths exists. Our improved bounds in Chapter 2 thus have implications for all of these results. We mention some examples, where store-and-forward packet routing is a subproblem that needs to be solved:

Srinivasan and Teo [101] present a constant factor approximation algorithm for packet routing with *variable paths*, i. e., in the setting where the routing paths are not part of the input, but need to be found by the algorithm. Koch et al. [60] improve and generalize this algorithm to the more general message routing problem (where each message consists of several packets). In both results, suitable paths are found that yield a constant factor approximation on the minimum of $C + D$ over all possible choices of paths. The remaining problem is then the ordinary packet routing problem. For the case that each vertex of a grid graph is the start vertex of at most one packet, Mansour and Patt-Shamir [76] prove the existence of a constant factor approximation on an optimal schedule, again by reducing the problem to an ordinary packet routing problem.

There are other result that go in the direction of the work of Leighton et al. [65]. For example, Meyer auf der Heide et al. [11] present a randomized online-routing protocol which finishes after at most $O(C + D + \log N)$ steps if all paths are shortest paths. Rabani et al. [89] give a distributed algorithm which guarantees a bound of $O(C) + (\log^* n)^{O(\log^* n)} D + poly(\log n)$. This is improved by Ostrovsky et al. [81] who give a local protocol that needs at most $O(C + D + \log^{1+\varepsilon} N)$ steps.

Packet routing is also studied in the case of special graph topologies. Leung et al. [70, chapter 37] study packet routing on certain tree topologies.

Leighton, Makedon and Tollis [67] show that the permutation routing problem on an $n \times n$ grid can be solved in $2n - 2$ steps using constant size queues. Rajasekaran [91] presents several randomized algorithms for packet routing on grids.

Studying the complexity for the ordinary packet routing problem, Di Ianni shows that the delay routing problem [25] is NP-hard. The proof implies that the packet routing problem on general graphs is NP-hard as well. Busch et al. [17] study the *direct* routing problem, that is the problem of finding a routing schedule such that a packet is never delayed once it has left its start vertex. They give complexity results and algorithms for finding direct schedules. Finally, Adler et al. [1, 2] study the problem of scheduling as many packets as possible through a given network in a certain time frame. They give approximation algorithms and NP-hardness results.

The packet routing problem is closely related to the multi-commodity flow over time problem [37, 50, 56]. In particular, Hall et al. [50] show that this problem is NP-hard, even in the very restricted case of series-parallel networks. We obtain the packet routing problem with variable paths if we additionally require unit edge capacities, unit transit times, and integral flow values. If there is only one start and one destination vertex then the packet routing problem with variable paths is equivalent to the quickest flow problem. It can be solved optimally in polynomial time, e. g., using the Ford-Fulkerson algorithm for the maximum flow over time problem [39, 40] together with a binary search framework. Using Megiddo's method of parametric search [77], Burkard, Dlaska, and Klinz [16] present a faster algorithm which solves the quickest flow problem in strongly polynomial time.

For our algorithm on directed trees we need to find a path coloring for the paths of the packets. The path coloring problem is widely studied in the literature. For instance, Raghavan et al. [90] present a (3/2)-approximation algorithm for the path coloring problem on undirected trees. Erlebach et al. [33] give NP-hardness results and algorithms for the problem. In particular, they improve the mentioned algorithm by Raghavan et al. and present a (4/3)-approximation. For coloring directed paths on bidirected trees (i. e., trees in which each edge represents two links, one in each direction) there are algorithms known which need at most $\frac{5}{3}L$ colors where L denotes the maximum load on a directed link [32]. This is tight since there are instances which actually need $\frac{5}{3}L$ colors [58]. Finally, Gargano et al. [44] investigate the problem of coloring all directed paths in a bidirected tree.

1.1.3 Outline of the Chapter

The general packet routing problem is very complex. However, in some applications the network topology is quite simple. For instance, in computer buses like Universal Serial Bus (USB) the topology is a tree. Such topologies make it easier to find good schedules (in comparison to the optimum). Therefore, in this chapter we study the packet routing problem on trees and additionally in the setting that a direct schedule is desired. Apart from obtaining algorithms,

we reveal important structural properties of the problem. In particular, the gained insights will be helpful in Chapter 2 where we will study the problem on general graphs. We present three individual new algorithms for the packet routing problem in different settings.

First, in Section 1.2 we show that even on directed trees the natural farthest-destination-first-algorithm (FDF) can yield arbitrarily large approximation factors. This implies that more sophisticated algorithms are necessary to obtain constant factor approximations. We show how to use the FDF-algorithm as a subroutine to obtain a 2-approximation algorithm for undirected trees. This guarantees a better performance ratio than the algorithm by Busch et al. [17] since their bound of $2C+D-2$ can asymptotically be as bad as a 3-approximation.

In Section 1.3 we study the problem on directed trees. First, we show how to solve the path coloring problem optimally. This is based on ideas presented in [33, 44]. Having computed such a coloring, we present a new technique which constructs a direct schedule whose length is bounded by $C + D - 1$. In comparison, the best known algorithm for computing direct schedules for packet routing on general trees guarantees a schedule of length $2C + D - 2$ [17]. (Note that in [17] it is assumed that each edge can be used in opposite directions independently at the same time.) The new method we employ is likely to be useful as a subroutine for packet routing on other topologies as well. In particular, we will use it in Chapter 2. Note that a makespan of $C + D - 1$ is a 2-approximation since C and D are both lower bounds on the optimum, but it guarantees a much better ratio if $C \ll D$ or $C \gg D$. Moreover, we show that $C + D - 1$ is the best ratio we can possibly guarantee in terms of C and D since there are instances which actually need this many steps.

Then, in Section 1.4 we present a very general condition which guarantees a direct schedule of a certain time horizon T in general graphs. Also, we present an algorithm which computes this schedule. This result is particularly substantial in the case where $T = D$ since then we can guarantee an optimal schedule. As an application, we show that if the paths of all packets are shortest paths and their lengths are pairwise different, we can compute an optimal direct schedule of length D. This improves a result in [75] where it was shown that under this condition there exists a (not necessarily direct) schedule of this length.

For all our algorithms we show that the analysis of the respective approximation ratios is tight. Finally, in Section 1.5 we conclude and discuss open questions.

1.2 Schedules for Undirected Trees

A very natural algorithm for the packet routing problem is the Farthest-Destination-First-Algorithm (FDF). It prioritizes the packets according to the length of their remaining path. That is, packets whose remaining path is longer have a higher priority than packets whose remaining path is shorter. Ties are broken arbitrarily. It was shown by Leung [70] that on in-trees (trees in which all vertices have an out-degree of at most one) and on out-trees (trees in which all

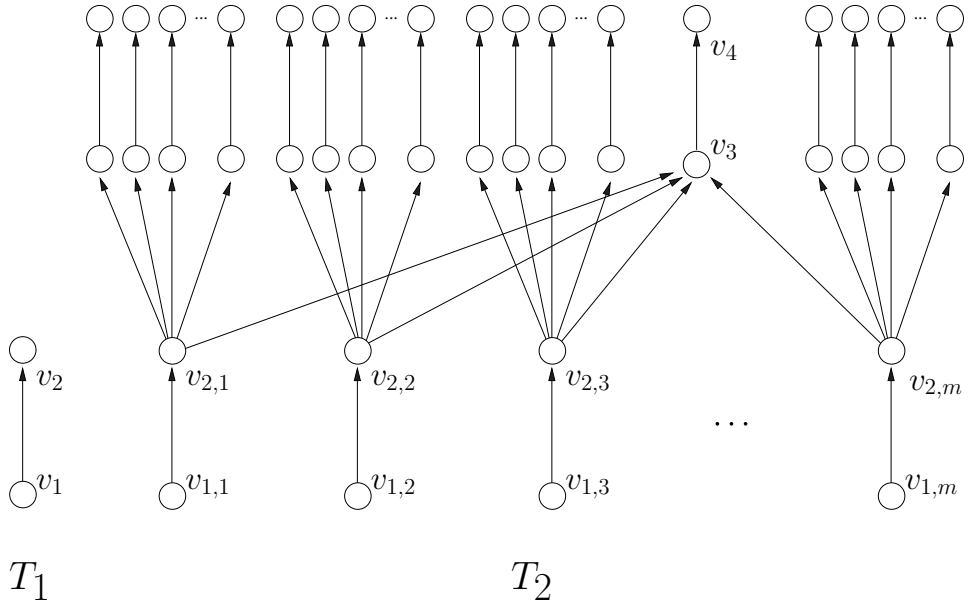


Figure 1.2: The trees T_1 and T_2 as defined in the proof of Theorem 1.1.

vertices have an in-degree of at most one) the FDF-algorithm works optimally. However, here we prove that on general directed trees the FDF-algorithm can perform arbitrarily bad in terms of the achieved performance ratio. Nevertheless, we present a 2-approximation algorithm for the packet routing problem on undirected trees that uses the FDF-algorithm as a subroutine.

First, we show that the FDF-algorithm alone can perform arbitrarily bad. For an instance I of the packet routing problem, denote by $FDF(I)$ a longest schedule that the FDF-algorithm could possibly compute (with the worst possible tie-breaking decisions).

Theorem 1.1. *For every $k \geq 1$ there is a directed tree T_k and a packet routing instance $I_k = (T_k, \mathcal{M}_k)$ such that*

$$|FDF\,(I_k)| \geq k \cdot |OPT\,(I_k)|\,.$$

Proof. In order to prove the claim, we inductively construct packet routing instances $(T_1, \mathcal{M}_1), ..., (T_k, \mathcal{M}_k)$. Let $m \geq 1$ be an integer and let $i = 1$. The directed tree T_1 (see Figure 1.2) consists of two vertices $\{v_1, v_2\}$. The set of packets $\mathcal{M}_1$ has m packets which all start in v_1 and whose destination vertex is v_2. We define all packets to be *upper packets* (in the inductive step we will see the reason for this definition). It is easy to see that for each upper packet $M \in \mathcal{M}_1$ there is a schedule with total makespan $m =: \beta_1$ such that M reaches its destination vertex after $1 =: \alpha_1$ timesteps. Moreover, the lengths of the paths of all packets are identical. Since in the FDF-algorithm ties are broken arbitrarily, the algorithm might compute a schedule in which M reaches its destination v_2 after $m =: \gamma_1$ timesteps.

Now let $i = 2$. For $I_2 = (T_2, \mathcal{M}_2)$ we take m copies of $I_1 = (T_1, \mathcal{M}_1)$. From each copy $I_{1,\ell}$ we choose one packet M_ℓ and extend its path by two more

vertices $\{v_3, v_4\}$. We call these packets M_ℓ the *upper packets* and all other packets the *lower packets*. For all lower packets we extend their path by two more vertices (so for each of these packet we introduce two new vertices). The whole construction is a directed tree and the congestion on the edge (v_3, v_4) is exactly m. See Figure 1.2 for a sketch of T_2.

We can easily see that an optimal schedule has length $2 + m =: \beta_2$ (give priority to the upper packets and schedule the lower packets in arbitrary order). We also see that for each upper packet M_ℓ there is an optimal schedule such that M_ℓ arrives at its destination after $3 =: \alpha_2$ timesteps. Since the paths of all packets have the same length, the FDF-algorithm schedules them in arbitrary order. In the worst possible schedule, the upper packets are scheduled with lowest priority. For each upper packet M_ℓ the FDF-algorithm might compute a schedule in which M_ℓ reaches its destination after $m + 1 + m =: \gamma_2$ steps.

We continue inductively: For the construction of I_i we take m copies of I_{i-1}. From each copy $I_{i-1,\ell}$ we choose one upper packet M_ℓ and extend its path by the vertices v_{2i-1} and v_{2i} (so all these packets now share one edge). These packets form the upper packets of I_i. All other packets are lower packets. The paths of all lower packets are extended by two new vertices (so we add two new vertices for each packet). Thus, the lengths of the paths of all packets are identical (since by the induction hypothesis they were identical in the previous iteration).

From the induction hypothesis we know that for each upper packet M_ℓ there is a schedule for $I_{i-1,\ell}$ with length β_{i-1} such that M_ℓ arrives at its destination after α_{i-1} timesteps. Also, there is another schedule for $I_{i-1,\ell}$ in which M_ℓ reaches its destination after γ_{i-1} steps. Thus, for each upper packet M_ℓ in I_i there is a schedule for I_i of length $\max\{\alpha_{i-1} + 1 + m, \beta_{i-1} + 2\} =: \beta_i$ in which $M_{i,\ell}$ reaches its destination after $\alpha_{i-1} + 2 =: \alpha_i$ steps. Also, there is an FDF-schedule for I_i in which M_ℓ reaches its destination after $\gamma_{i-1} + 1 + m =: \gamma_i$ steps.

Some basic calculus shows that $\alpha_i = 2i-1$, $\beta_i = 2i-2+m$ and $\gamma_i = i \cdot m + i - 1$. Recall that $FDF(I_i)$ is the longest possible schedule produced by the FDF-algorithm for I_i. Then it holds that

$$FDF(I_i) \geq \frac{\gamma_i}{\beta_i} \cdot OPT(I_i) = \frac{i \cdot m + i - 1}{m + 2i - 1} \cdot OPT(I_i).$$

The claim follows by choosing m and i sufficiently large. □

The above theorem is bad news if one is interested in a good approximation algorithm. However, we can design a 2-approximation algorithm for the packet routing on undirected trees. It uses the FDF-algorithm as a subroutine.

The algorithm works as follows: Let $T = (V, E)$ be a tree and let $I = (T, \mathcal{M})$ be a packet routing instance. We define an arbitrary vertex v_r to be the root of the tree and orientate all edges towards v_r. We observe that the orientation of the edges splits the path of each packet M_i into two parts: in the first part M_i moves according to the direction of the edges. In the second part of the path M_i moves in the opposite direction of the edges. For each packet M_i let v_i be

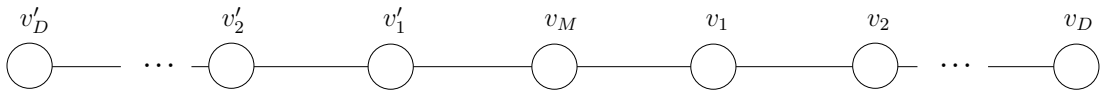


Figure 1.3: Graph for showing that the analysis in Theorem 1.2 is tight.

the vertex which divides the two parts of P_i. We split the whole problem into two subproblems: First we move each packet M_i from s_i to v_i. In the second part we move each packet M_i from v_i to t_i. We observe that the first part is a packet routing problem on an in-tree and can therefore be solved optimally using the FDF-algorithm (see [70] for a proof). Similarly, the second part is a packet routing instance on an out-tree which can also be solved optimally using the FDF-algorithm (see [70]). In the overall schedule for I, we run the optimal (FDF-)schedule for the first part. Then we run the optimal (FDF-)schedule for the second part. Denote by $TREE(I)$ the resulting schedule for the instance I.

Theorem 1.2. *For the schedule $TREE(I)$ it holds that $|TREE(I)| \leq 2 \cdot |OPT(I)|$. Moreover, it can be computed in $O\left(|\mathcal{M}|^2 + |V|^2\right)$.*

Proof. The length of an optimal schedule for each of the two subproblem forms a lower bound on the length of an optimal schedule for the whole problem. We solve both subproblems optimally. Therefore, we achieve an approximation ratio of two.

In $O(|V|)$ steps we can determine for every vertex its distance to v_r. Then, every step of the FDF-algorithm can be computed in $O(|\mathcal{M}| + |V|)$. The length of an optimal schedule in an in- or out-tree is bounded by $O(|\mathcal{M}|+|V|)$, see [75]. Since the FDF-algorithm computes an optimal schedule the overall running time is bounded by $O(|\mathcal{M}|^2 + |V|^2)$. □

When implementing the algorithm one would not let packets wait until all other packets have finished the first part of the schedule. We would rather always move a packet when the next edge on its path is free and prioritize the packets according to the algorithm. But even then there are instances which show that our analysis is tight.

Let D be an arbitrary positive integer (in the packet routing instance this will be the dilation). Consider the graph shown in Figure 1.3. We introduce $2 \cdot D$ packets as follows: We define $M_1 := (v_M, v_D)$ and $M'_1 := (v_M, v'_D)$. For each $k \in \{2, ..., D\}$ we define $M_k := (v_1, v_M)$ and $M'_k := (v'_1, v_M)$. The optimal makespan for this packet routing instance is D, obtained by giving priority to the packets M_1 and M'_1 and scheduling the other packets in arbitrary order. In order to analyze the schedule obtained by our algorithm we need to consider all possible choices for v_r. If $v_r = v_M$ or $v_r = v_k$ with $1 \leq k \leq D$ then the packets M'_2 to M'_k have a higher priority than M'_1 and therefore M'_1 will need $2 \cdot D - 1$ steps to reach v'_D. Analogously, if $v_r = v'_k$ with $1 \leq k \leq D$ then the packets M_2 to M_k have a higher priority than M_1 and therefore M_1 will need $2 \cdot D - 1$ steps to reach v'_D. Thus, the competitive ratio of $\mathcal{A}$ is at best $\frac{2 \cdot D - 1}{D}$.

Since we can choose D arbitrarily large this shows that the proven competitive ratio of 2 is tight.

Theorem 1.2 proves that there is a 2-approximation algorithm for the packet routing problem on trees. We will see in Chapter 3 that on trees the problem cannot be approximated with a factor of $8/7 - \varepsilon$ for any $\varepsilon > 0$, unless $P = NP$. As we will see, this holds even for directed trees.

1.3 Schedules for Directed Trees

After having studied the packet routing problem on undirected trees we turn to directed trees. Note that the trees that we mean here are not necessarily in- or out-trees but arbitrary directed trees. In comparison to undirected trees we have more structure that we can use. This allows us to construct a *direct* schedule of length at most $C + D - 1$ in polynomial time. In comparison, the best known algorithm for direct schedules in general trees requires $2C + D - 2$ steps [17]. Note that the bound of $C + D - 1$ is best possible since there are instances where this many steps are needed. Moreover, in Chapter 2 we will use the gained insights when we derive upper bounds for the lengths of optimal schedules in general graphs.

In the remainder of this section we describe our algorithm for computing direct schedules on instances on directed trees. The algorithm works as follows: First we find a coloring for the paths of the packets such that two paths that share an edge have different colors. We will show that the number of needed colors is exactly C. We assign each packet the color of its path. Then we assign each edge a time-dependent color. The idea behind this is that we transfer a packet M with color c_M along an edge $e = (u, v)$ only when e has the color c_M. We define the coloring of the edges such that for two consecutive edges $e = (u, v)$ and $e' = (v, w)$ it holds that at each time t the edge e' has always the color that e had at time $t - 1$. This ensures that once a packet starts moving, it will never stop until it reaches its destination. Since at the very beginning each packet is delayed for at most $C - 1$ steps this gives a makespan of at most $C + D - 1$.

In the sequel we describe the algorithm in detail. Assume we are given an instance of the packet routing problem on a directed tree.

1.3.1 Path Coloring

First we want to find a coloring for the paths such that two paths with the same color do not share an edge. Our algorithm works in two phases: in the first phase we consider each vertex v together with its adjacent vertices (this subgraph forms a star). For each of these subgraphs – together with the paths of the packets in this subgraph – we solve the path coloring problem optimally (here we use the fact that our tree is directed). Then we combine all these partial solutions and obtain a solution for the global path-coloring problem.

Phase one: Let v be a vertex and denote by $T_v = (V, E)$ the subgraph induced by v and its adjacent vertices. We want to find a coloring for the paths

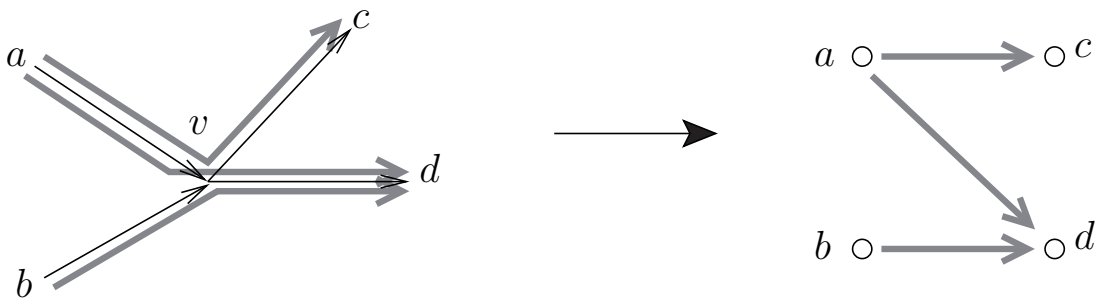


Figure 1.4: Left: a vertex v with its neighbors and the paths touching v. Right: The corresponding bipartite graph B_v.

which use edges in T_v. We reduce this problem to the edge-coloring problem on bipartite multigraphs (for this it is crucial that the edges in T_v are directed). This construction was already mentioned in [44].

Let U be the set of vertices which have outgoing edges to v, i. e., $U = \{u \in V | (u, v) \in E\}$. Similarly, let W be the set of vertices having ingoing edges from v, i. e., $W = \{w \in V | (v, w) \in E\}$. We construct an undirected bipartite graph B_v as follows: the set $U \cup W$ forms the set of vertices in B_v. For each path P that goes from a vertex $u \in U$ through v to a vertex $w \in W$ we introduce an edge $e_P := \{u, w\}$ in B_v. For all paths P that start in a vertex $u \in U$ and end in v we introduce a new vertex $w_P \in W$ and an edge $e_P := \{u, w_P\}$ in B_v. Similarly, for paths P that start in v and end in a vertex $w \in W$ we add a vertex v_P and introduce an edge $e_P := \{v_P, w\}$ in B_v. Thus, for the maximum degree $\Delta(B_v)$ of a node in B_v it holds that $\Delta(B_v) \leq C$. Also, it holds that two edges e_P and $e_{P'}$ share an end-vertex if and only if their corresponding paths P and P' share an edge in T_v. Thus, a valid edge-coloring for B_v implies a valid path coloring for T_v and vice versa. Moreover, from the construction it follows that B_v is bipartite. We compute a minimum edge coloring for B_v (e. g., see [22]). The number of colors needed equals the maximum node degree $\Delta(B_v)$, see [22].

Phase two: Now we combine the found solutions for the graphs T_v one by one to obtain a global solution (a similar construction is described in [33, Lemma 2]). We start with an arbitrary vertex v and the path coloring of T_v. Now let v' be a vertex adjacent to v and consider the graph $T_{v'}$. We permute the colors of the paths in $T_{v'}$ such that the paths which use the edge (v, v') (or (v', v), respectively) have the same colors in T_v and $T_{v'}$. We iterate over the vertices by always adding a vertex that is adjacent to one of the vertices that have been considered already. Eventually, for each edge $e = (u, v)$ all paths that use e have the same color in T_u and T_v. Since T is a tree, in each iteration we can find a valid permutation of the colors of the paths by using a simple greedy strategy. Since for each graph T_v we found path colorings with at most C colors, the resulting path coloring for T has C colors as well.

We summarize the path coloring procedure in the following lemma.

Lemma 1.3. *Let T be a directed tree and let $I = (T, \mathcal{M})$ be a packet routing instance. There is a coloring with C colors for the paths of the packets $\mathcal{M}$ such*

that any two paths that share an edge have different colors. Moreover, such a coloring can be computed in $O(n \cdot N \cdot \log C)$*, where* n *denotes the number of vertices in* T *and* $N = |\mathcal{M}|$*.*

Proof. The edge coloring problem on bipartite multigraphs can be solved optimally in $O(m \log \Delta)$ where m denotes the number of edges in the graph and Δ denotes the maximum node degree, see [22]. Thus, computing the optimal path coloring for one graph T_v can be done in $O(N \cdot \log C)$ and for all graphs T_v in $O(n \cdot N \cdot \log C)$ steps.

Then we need to combine the colorings for the graphs T_v to a global path coloring. We say a path P *touches* a vertex v if P goes through v, starts in v or ends in v. We pick an arbitrary vertex v and color all paths which touch v in the colors that they have in T_v. After this initialization we iterate by taking vertices v' which are adjacent to already considered vertices. When we iterate we need to find a color permutation for $T_{v'}$ which is consistent with the coloring for T_v. This permutation is partly already defined by the color assignment for T_v (from the paths that appear in T_v and $T_{v'}$). The remainder can be found greedily in $O(C) \subseteq O(N)$ steps. Since the order of the vertices can be obtained by a depth-first-search the second phase can be done in $O(n \cdot N)$. This gives a total running time of $O(n \cdot N \cdot \log C)$. □

1.3.2 Time-Dependent Edge Coloring

Now we construct a time-dependent coloring $g : E \times \mathbb{N} \to \{0, 1, ..., C-1\}$ for the edges of T. We will use this later to define the routing schedule. We require it to have the *consecutive property:*

- For two consecutive edges $e = (u, v)$ and $e' = (v, w)$ we require that $g(e, i) = g(e', (i+1) \bmod c)$ for $0 \leq i < C$.
- For two adjacent edges $e = (u, v)$ and $e' = (u, v')$ (or $e = (u, v)$ and $e' = (u', v)$) we require that $g(e, i) = g(e', i)$ for $0 \leq i < C$.

We compute such a coloring with the following greedy algorithm: Start with an arbitrary edge e and define its coloring by $g(e, i) := i \bmod C$ for all $i \in \mathbb{N}$. Then we inductively assign the colors to the remaining edges such that the consecutive property holds. Note that in the computed coloring each color appears exactly every C timesteps on every edge.

1.3.3 Routing Schedule

Finally, we describe the routing schedule. First, we assign each packet the color of its path. Now let M be a packet which is located on a vertex u at time t and which needs to use the edge $e = (u, v)$ next. Let c_M be the color of M. We route M along e in the first timestep t' with $t' \geq t$ and $g(e, t') = c_M$. We define our schedule like this for every packet and every timestep. Due to

the consecutive property of the time-dependent edge coloring a packet is never delayed once it has left its start vertex. Denote by $DTREE(I)$ the resulting schedule for an instance I. The following theorem summarizes the schedule and its properties.

Theorem 1.4. *Let T be a directed tree and let $I = (T, \mathcal{M})$ be a packet routing instance. Then it holds that $|DTREE(I)| \leq C + D - 1$. Also, a packet is never delayed once it has left its start vertex (direct routing). Moreover, $DTREE(I)$ can be computed in $O(n \cdot |\mathcal{M}| \cdot \log C)$, where n denotes the number of vertices in T.*

Proof. Since no two paths with the same color share an edge there can be at most one packet that uses each edge e at each time t. According to the schedule, each packet waits in its start vertex for at most $C - 1$ timesteps. Due to the consecutive property once it left its start vertex it moves to its destination without being delayed any further. Thus, the length of the overall makespan is bounded by $C + D - 1$. The running time of the algorithm is dominated by the computation of the path coloring. The latter can be done in $O(n \cdot |\mathcal{M}| \cdot \log C)$, see Lemma 1.3. □

Note that the bound $C + D - 1$ is the best bound we can give in terms of C and D since there are packet routing instances which need this many steps. For example, consider a path of length D with vertices $v_0, ..., v_D$ and C packets all with start vertex v_0 and destination vertex v_D.

1.4 Direct Schedules

In the previous section we showed that for instances of the packet routing problem on directed trees there is always a direct schedule with length at most $C + D - 1$. In this section we give a condition which guarantees the existence of a direct schedule within some time horizon T in general graphs. First we proved this condition for directed trees. However, it turns out that it also holds in general directed graph if every packet uses a shortest path. As an application of the condition we prove that if the lengths of the paths are pairwise different there is an optimal direct schedule of length D (which is optimal). The latter improves a result in [75] where the existence of some schedule, but not necessarily a direct schedule, with this makespan was shown.

Our main result in this section is the following theorem.

Theorem 1.5. *Let G be a directed graph, let $T \geq 0$ be a time horizon, and let $I = (G, \mathcal{M})$ be a packet routing instance. Assume that for the paths of the packets the following conditions hold:*

- *All paths are shortest paths.*
- *For each packet M_i denote by $\mathcal{M}_i$ the set of packets M_j such that P_j shares an edge with P_i and $D_j \geq D_i$. We assume that $|\mathcal{M}_i| \leq T - D_i + 1$.*

Then, there is a direct schedule which needs at most T timesteps. Moreover, such a schedule can be computed in $O(|\mathcal{M}| \cdot (m + \log |\mathcal{M}|))$ where m denotes the number of edges in G.

As we will see later, the bound $|\mathcal{M}_i| \leq T - D_i + 1$ guarantees the existence of a direct schedule for the given instance. In fact, we will show that if only a weaker bound holds, e.g., $|\mathcal{M}_i| \leq T - D_i + 2$ for each packet M_i, then a direct schedule with length T might not exist.

Assume that we are given an instance I and a value T with the above properties. We describe an algorithm which computes a direct schedule $DIRECT(I)$ which needs at most T steps. For two packets M_i and M_j such that P_i and P_j share an edge we define a value $d(M_i, M_j)$. Let v_{ij} be the first vertex on P_i and P_j which is used by both paths. Denote by $d(s_i, v_{ij})$ and $d(s_j, v_{ij})$ the number of edges on P_i and P_j between s_i and v_{ij} and between s_j and v_{ij}, respectively. Then we define $d(M_i, M_j) := d(s_i, v_{ij}) - d(s_j, v_{ij})$. Note that $d(M_i, M_j) = -d(M_j, M_i)$.

Assuming that M_i is delayed for k timesteps in the beginning, M_i collides with M_j if and only if P_i and P_j share an edge and M_j is delayed for $d(M_i, M_j) + k$ steps (here we use the fact that G is a directed graph and all paths are shortest paths). We sort the packets in non-increasing order by the lengths of their paths. W. l. o. g. we assume that $M_0, ..., M_k$ is such an order. Now we iterate over the packets. In the i-th iteration, we consider the packet M_i. Denote by d_j the computed waiting time for each packet M_j with $0 \leq j < i$. (Note that a direct schedule is completely defined by the initial waiting times for the packets.) We say a packet M_j *blocks* a certain waiting time m for M_i if P_i and P_j share an edge and $m = d(M_i, M_j) + d_j$. Note that each packet whose path shares an edge with P_i blocks at most one waiting time for P_i. Let m be the smallest unblocked waiting time for M_i. We define $d_i := m$. Denote by $DIRECT(I)$ the resulting schedule.

Proof of Theorem 1.5. We prove that $DIRECT(I)$ has the properties claimed in the theorem. By construction it is a direct schedule. Hence, it remains to prove that $D_i + d_i \leq T$ for each packet M_i. We considered the packets in non-increasing order of their path lengths. Hence, only packets in $\mathcal{M}_i$ can possibly block a certain waiting time for M_i. We know that $|\mathcal{M}_i| \leq T - D_i + 1$ and $M_i \in \mathcal{M}_i$. Thus, we conclude that at most $T - D_i$ waiting times for M_i are blocked by packets M_j with $j < i$. This proves that $d_i \leq T - D_i$ which implies that $D_i + d_i \leq D_i + (T - D_i) = T$.

Now we want to bound the running time. First, we need to sort the packets by the lengths of their paths. Determining the length of the path of each packet can be done in $O(|\mathcal{M}| \cdot m)$. The sorting takes $O(|\mathcal{M}| \log |\mathcal{M}|)$ steps. For each packet we need to determine the smallest unblocked waiting time. For each edge we maintain a list of timesteps when it is already used by some packet. Hence, for each packet M_i we can determine in $O(D_i)$ steps the smallest unblocked waiting time. This gives an overall running time of $O(|\mathcal{M}| \log |\mathcal{M}| + |\mathcal{M}| \cdot m) \subseteq O(|\mathcal{M}| \cdot (m + \log |\mathcal{M}|))$. □

Note that the bound for the length of $DIRECT(I)$ is tight. E. g., let C and D be arbitrary positive integers and consider a packet routing instance as follows: Let the graph be a directed path with vertices $v_0, v_1, ..., v_D$ and consider C packets all with start vertex v_0 and destination vertex v_D. We define $T := C + D - 1$. Then the above conditions are satisfied (since for all packets M_i we have that $|\mathcal{M}_i| = |\mathcal{M}| = C = T - D_i + 1$) and the length of the optimal schedule is exactly T. Moreover, if we weaken our condition and require only that $|\mathcal{M}_i| \leq T - D_i + 2$ we cannot guarantee the existence of a schedule of length T anymore. E. g., take the above example with $C + 1$ packets from v_0 to v_D and $T := C + D - 1$. Then each schedule needs at least $C + D > T$ steps.

Theorem 1.5 directly implies the following two corollaries.

Corollary 1.6. *Let G be a directed graph, let $I = (G, \mathcal{M})$ be a packet routing instance, and let $T \geq 0$ be a time horizon with the following conditions:*

- *All paths are shortest paths.*
- *Let $\mathcal{M}_i$ denote the set of packets whose path has at least $D - i$ edges. For each $i \geq 0$ we have that $|\mathcal{M}_i| \leq i + 1$.*

Then the schedule $DIRECT(I)$ is optimal with $|DIRECT(I)| = D$.

Corollary 1.7. *Let G be a directed graph and let $I = (G, \mathcal{M})$ be a packet routing instance such that all paths are shortest paths and the lengths of all paths are pairwise different. Then the schedule $DIRECT(I)$ is optimal with $|DIRECT(I)| = D$.*

Compare that in [75] it was shown under the conditions of Corollary 1.7 that there is a (not necessarily direct) schedule whose length is bounded by D. We proved that in this case there is even a direct schedule with this makespan.

1.5 Conclusion

In this chapter we studied the packet routing problem on undirected and directed trees. We gave a 2-approximation algorithm for undirected trees and an algorithm with a bound of $C + D - 1 \leq 2 \cdot OPT$ for directed trees. Note that the latter algorithm performs much better than $2 \cdot OPT$ if $C \ll D$ or $C \gg D$. However, it remains open to design an approximation algorithm which guarantees a better performance ratio than 2 in worst-case. For achieving this goal, better lower bounds are necessary. In our algorithm for undirected trees the lower bounds are given by the two subproblems where the packets travel towards the root or away from the root. It is easy to construct instances where each of the two lower bounds is by a factor of 2 away from the optimum. For instance, consider a path used by only one packet and define the root of the tree to be the vertex in the middle of the path. In our algorithm for directed trees the lower bounds are C and D. As mentioned earlier, the bound of $C + D - 1$ is best possible since there are instances which need this many steps. An important

step to improve the approximation factors is hence to construct better lower bounds. Also for direct schedules better lower bounds are needed to improve the existing results.

Chapter 2

Schedules for General Graphs

2.1 Introduction

When designing approximation algorithms, a key ingredient are lower bounds on the value of an optimal solution. For instance, many known approximation algorithms are based on solving a linear program and subsequently rounding the fractional solution to an integral one. An example is one of the many 2-approximation algorithms for Vertex Cover. Recall that Vertex Cover is the problem of finding a minimum subset of vertices in a graph such that each edge is adjacent to at least one chosen vertex. In the algorithm, one solves the (canonical) LP-relaxation of the problem. One can show that rounding up all variables whose value is at least $\frac{1}{2}$ yields an integral 2-approximative solution for the problem [93, Vol. B]. Other approximation algorithms are based on a combinatorial relaxation that can be solved efficiently. They compute a solution to the original problem based on the relaxation and compare their cost with the cost of the relaxation. For Vertex Cover a combinatorial lower bound is the size of a maximal matching. Such a matching can easily be obtained by a greedy algorithm. Then, we pick all vertices which are adjacent to a matching edge and obtain a 2-approximation for the problem.

Key for proving in both cases that we have a 2-approximation is that the obtained integral solution is by at most a factor of 2 larger than the value of the lower bound (LP-optimum and the cardinality of the matching, respectively). An important question is whether this analysis is best possible. In other words: Are there really instances where the LP-optimum is by a factor of 2 smaller than the best integral solution? Are there instances where a maximal matching can be by a factor of 2 smaller than an optimal Vertex Cover? For both questions the answer is "Yes", simply consider complete graphs. In fact, the two algorithms described above are (asymptotically) the best known approximation algorithms for the problem.

In this chapter, we study lower bounds for the packet routing problem. The two (trivial) lower bounds for the length of an optimal schedule are the con-

gestion C and the dilation D. But how good are these bounds? Can an optimal schedule be significantly longer than $\max\{C, D\}$? In a celebrated paper Leighton, Maggs, and Rao proved that the length of an optimal packet routing schedule is bounded by $O(C + D)$ [65]. This means that there is an (in their proof very large) constant k such that each optimal packet routing schedule needs at most $k \cdot (C + D)$ steps. Later, the constant was improved by Scheideler who proved a bound of $39(C + D)$ [92]. It is easy to see that there are packet routing instances that need $C + D - 1$ steps to finish, e. g., consider a path of length D and C packets which travel along this path with equal start and destination vertices. However, to the best of our knowledge, no instance is known that needs significantly more time in comparison to C and D.

In this chapter, we make a step to close the gap between the upper bound of $39(C + D)$ and the lower bound of $C + D - 1$. We improve the bound of $39(C + D)$ to $23.4(C + D)$. Even more, we study packet routing in a wider setting than before. We allow the edges to have non-unit bandwidths and non-unit transit times. This means that more than one packet might enter some edge at the same time and it might take more than one timestep for a packet to traverse an edge (both depending on the respective edge). The above bound on the length of an optimal schedule carries over quite easily to this more general setting. However, we can show even *better* bounds when in the network every edge has at least a certain minimum bandwidth or at least a certain minimum transit time. For instance, if every edge has a transit time of at least 63 then our bound improves to $4.32(C+D)$. Note here that we generalize the notions of congestion and dilation to arbitrary bandwidths and transit times in a canonical way.

Our methods use the Lovász Local Lemma (LLL) which in itself is non-constructive. However, Moser and Tardos [79] recently provided an algorithmic version of the general LLL. Hence, we also obtain efficient algorithms which compute schedules with the stated lengths.

The results presented in this chapter are joint work with Britta Peis [88].

2.1.1 The Model

We modify the definition of the packet routing given in Chapter 1 as follows. We assume that all edges are directed and the packets use the edges only in their given direction. Each edge $e \in E$ is equipped with a certain bandwidth $b_e \in \mathbb{N}$ denoting the maximal number of packets that are allowed to enter e simultaneously, and a certain transit time $\tau_e \in \mathbb{N}$ denoting the time needed for a single packet to traverse e[1]. We define

$$b := \min_{e \in E} b_e \quad \text{and} \quad \tau := \min_{e \in E} \tau_e.$$

A *feasible packet routing schedule* for the instance $I = (G, \mathcal{M})$ is now a schedule which defines for each packet when it enters each edge of its path, respecting the transit times and the bandwidths of the edges.

[1] Note that even if $b_e = 1$ in principle an arbitrary number of packets is allowed to traverse e simultaneously. The value b_e denotes a bound on the packets that can *enter* e at the same time.

Congestion and Dilation

The two trivial lower bounds dilation and congestion easily carry over to non-unit transit times and bandwidths. For each packet M_i we define D_i to be the length of P_i and D to be the maximal length of a path, i. e.,

$$D_i := \sum_{e \in P_i} \tau_e \quad \text{and} \quad D := \max_{M_i \in \mathcal{M}} D_i.$$

Also, for each edge $e \in E$ we define C_e to be the number of paths using e. We define the *congestion* C by

$$C := \max_{e \in E} \lceil C_e / b_e \rceil .$$

Clearly, the *dilation* D as well as the *congestion* C provide lower bounds on the length of an optimal schedule.

Remark regarding large $C + D$

In our analysis we will always assume that $C + D$ is large enough such that $\lceil k \cdot (C + D) \rceil \approx k \cdot (C + D)$ for certain constants k. This simplifies the calculations and was also implicitly used by Scheideler [92]. In order to give a fair comparison with his bounds, we use this assumption as well. Moreover, we believe that also for instances where $C + D$ is small, our techniques can be used to prove good bounds for the optimal makespan. However, this would require further case distinctions and is beyond the scope of this work.

2.1.2 Outline of the Chapter

As mentioned above, we prove bounds on the length of optimal packet routing schedules in the case of arbitrary transit times and bandwidths for the edges. For the classical setting with $b = 1$ and $\tau = 1$, we improve the best known bound of $39(C + D)$ due to Scheideler [92] to $23.4(C + D)$. Even more, for larger b or τ our bounds improve further to $7.63(C + D)$ for $b \to \infty$ and to $4.32(C + D)$ for $\tau \geq 63$. See Table 2.1 for an overview for some values depending on b and τ.

The key insight for our analysis is to prove and exploit a good bound for schedules with very small dilation: For $D \leq \tau + 1$ we show that there is always a schedule of length $C + D - 1$. Note that this bound is tight, since there exist instances which require a schedule of this length (e. g., consider C packets that need to take the same path of length D). Our proof framework uses this insight in order to develop good bounds for general instances. Moreover, our approach points into a direction of promising further research: If one could prove similarly tight bounds for instances with, e. g., $D \leq k\tau + 1$ for small values of k, our proof framework would immediately give even better bounds

Bound on length of optimal schedule						
	$\tau = 1$	$\tau = 2$	$\tau = 5$	$\tau = 10$	...	$\tau = 63$
$b = 1$	23.40	23.21	18.54	16.37	...	4.32
$b = 2$	21.59	18.85	15.58	14.27	...	4.32
$b = 5$	16.19	14.50	12.98	12.38	...	4.32
$b = 10$	14.03	13.05	11.86	11.54	...	4.32
...	...	...	...	...	...	...
$b \to \infty$	7.63	7.63	7.63	7.63	...	4.32

Table 2.1: Bounds for schedules obtained in this chapter depending on the minimum bandwidth b and the minimum transit time τ of the edges. The given values denote the constants in front of $(C + D)$. Note that for technical reasons (which will become clear in Section 2.4) the bounds do not improve any further when τ is larger than 63.

for *all* instances. This work gives the first improvement for the bounds since the result by Scheideler [92] from 1998.

The rest of this chapter consists of four parts: In Section 2.2 we prove a bound of $C + D - 1$ for instances with $D \leq \tau + 1$. Then, in Section 2.3, we give a high-level overview of our techniques for proving bounds for general instances. There, we need the insights obtained in Section 2.2. In Section 2.4 we prove our bounds for general graphs in detail. Finally, in Section 2.5 we conclude and discuss open problems.

2.2 Tight Bound for Small Dilation

Ideally, we would like to determine for each combination of C, D, b, and τ a tight upper bound on the maximal length of an optimal schedule for instances with these parameters. In this section, we make a first step towards this goal: We give a tight bound for instances with $D \leq \tau + 1$. As we will see in Section 2.3, this insight will allow us to prove good upper bounds for *all* instances.

For the sake of analysis, for the remainder of this chapter we replace every edge $e \in E$ with transit time τ_e by a path consisting of τ_e edges with unit transit time, all with the same bandwidth as e. In the resulting graph, we call every vertex that was created due to this transformation a *small* vertex. All other vertices are called *big* vertices. We say a *small path* is a path connecting two big vertices. Hence, τ is the minimum length of a small path. Even more, we allow each small vertex to be the start or the destination vertex of a packet. Note that this assumption makes the problem slightly more general since now packets are allowed to have their start vertex "in the middle" of an edge. We introduce this generalization because it will be needed in our proof framework later. To simplify notation later we introduce the notion $A^{b,\tau}(C, D)$.

Definition 2.1. Let $\mathcal{I}^{b,\tau}(C, D)$ be the set of all packet routing instances with minimum bandwidth b, minimum transit time τ, congestion C, and dilation D.

We define

$$A^{b,\tau}(C,D) := \max_{I \in \mathcal{I}^{b,\tau}(C,D)} OPT(I),$$

where $OPT(I)$ denotes the length of an optimal schedule for an instance I.

The main result of this section is given in the following theorem.

Theorem 2.2. *Let I be an instance of the packet routing problem with $D \leq \tau + 1$. Then there is a schedule for I whose makespan is bounded by $C+D-1$ which can be computed in polynomial time. Moreover, $A^{b,\tau}(C,D) = C+D-1$ if $D \leq \tau+1$.*

Proof of Theorem 2.2

The strategy of the proof is the following: We assume that we are given an instance I of the packet routing problem with $D \leq \tau + 1$. Due to the latter assumption the path of each packet uses at most two small paths. Thus, we can divide the packets which use any small path P into two sets $\mathcal{M}_P^1$ and $\mathcal{M}_P^2$, such that $\mathcal{M}_P^1$ contains the packets for which P is the first of two used small paths, and $\mathcal{M}_P^2$ contains the packets for which P is the second small path (of two used small paths) or the only small path which they use.

In a first step, we transform I into an instance I' such that $OPT(I') \geq OPT(I)$ and for every small path P' either $\mathcal{M}_{P'}^1 = \emptyset$ or $\mathcal{M}_{P'}^2 = \emptyset$. This procedure reduces the complexity of the instance significantly. While performing the necessary changes we do not change C, D, b, or τ at all. It turns out that the underlying graph topology of the resulting instance I' is equivalent to a directed forest, i. e., each connected component is a directed tree. We show then that for each connected component there is a schedule which finishes after at most $C + D - 1$ steps (Lemma 2.4). Since $OPT(I') \leq C + D - 1$ and $OPT(I) \leq OPT(I')$, it follows that $A^{b,\tau}(C,D) \leq C + D - 1$.

It is straight forward to construct instances I with $D \leq \tau+1$ and $OPT(I) = C + D - 1$. For example, consider an instance with only one edge which is used by $C \cdot b$ packets. This shows that the bound is tight, i. e., $A^{b,\tau}(C,D) = C+D-1$.

Transformation from I to I'. We describe the necessary transformation to obtain the instance I' from I. First note that for an optimal schedule for I we can assume w. l. o. g. that on any small path P no packet in $\mathcal{M}_P^2$ ever delays a packet in $\mathcal{M}_P^1$. Consider a small path $P = \{e_1, e_2, ..., e_k\}$ connecting two big vertices u and v. We substitute the path by two parallel paths $P^1 = \{e_1^1, e_2^1, ..., e_k^1\}$ and $P^2 = \{e_1^2, e_2^2, ..., e_k^2\}$ connecting u and v where each edge e_j^i has the same bandwidth as e_j. In the new instance I', we define that the packets of $\mathcal{M}_P^1$ use the respective edges in P^1, whereas the packets in $\mathcal{M}_P^2$ use P^2.

Now, let $M \in \mathcal{M}_P^1$ be a packet that uses the edges $e_\ell^1, ..., e_{\ell'}^1$ for $1 \leq \ell \leq \ell' \leq k$. (As mentioned above, we allow the start- and destination vertices to be small vertices.) For each such packet M, we introduce an *artificial packet* M' whose path consists of the edges $e_\ell^2, ..., e_{\ell'}^2$. If $\ell' = k$ we call M' a *far artificial packet*. We do this procedure with every packet in $\mathcal{M}_P^1$. Denote by I' this transformed instance. For every small path $P = \{e_1, e_2, ..., e_k\}$ in G we denote

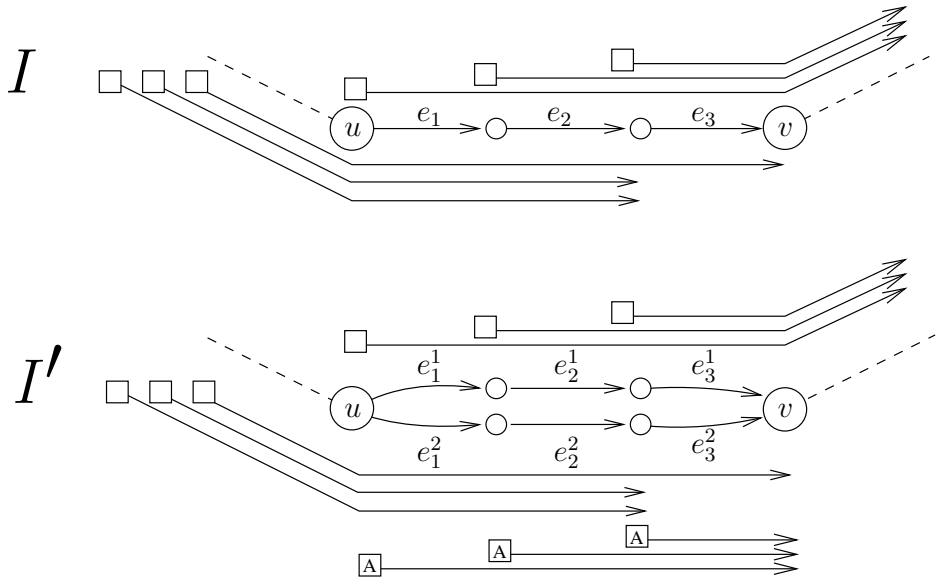


Figure 2.1: A sketch of the transformation from I to I' in Section 2.2 with $\tau_{(u,v)} = 3$ and u and v being big vertices. In the upper instance I each of the the six packets uses some of the edges e_1, e_2, e_3. In the transformed instance I' below the upper three packets use e_1^1, e_2^1, and e_3^1 since $\{e_1, e_2, e_3\}$ is their first small path in I. Similarly, the middle three packets use e_1^2, e_2^2, and e_3^2. The three packets on the bottom are new artificial packets.

by ${\mathcal{M}'}_P^1$ and ${\mathcal{M}'}_P^2$ the packets in I' which use edges of $P^1 = \left\{e_1^1, e_2^1, ..., e_k^1\right\}$ and $P^2 = \left\{e_1^2, e_2^2, ..., e_k^2\right\}$, respectively. See Figure 2.1 for a sketch.

Note that in I' we still have that $D \leq \tau + 1$. Moreover, in I' we have that either ${\mathcal{M}'}_{P'}^1 = \emptyset$ or ${\mathcal{M}'}_{P'}^2 = \emptyset$ for each small path P' (where the sets ${\mathcal{M}'}_{P'}^1$ and ${\mathcal{M}'}_{P'}^2$ correspond to the respective sets $\mathcal{M}_P^1$ and $\mathcal{M}_P^2$ for small paths P in I). Also, the transformation did not change C, D, b, or τ. We observe that the topology of I' is equivalent to a directed forest, i. e., a graph where each connected component is a directed tree. Even more, in each of the connected components there is at most one vertex with a larger degree than 2. We call directed trees with this property *directed spiders*.

Upper Bound for $OPT(I')$. We establish that the transformation to I' did not reduce the length of an optimal schedule.

Lemma 2.3. *We have that $OPT\,(I') \geq OPT\,(I)$. Also, a schedule S for I' can be transformed in polynomial time to a schedule for I which is not longer than S.*

Proof. For the scheduling decisions within each small path there is always an optimal schedule which obeys the farthest-destination-first rule (since FDF is optimal for packet routing on a path). Hence, for the instance I' there is always an optimal schedule in which no far artificial packet is ever delayed by a non-artificial packet. Thus, our transformation ensures that $OPT\,(I') \geq OPT\,(I)$.

Moreover, we can modify a schedule S for I' to a schedule for I: On each small path P we schedule the packets in $\mathcal{M}_P^1$ according to the schedule I'. The packets in $\mathcal{M}_P^2$ are scheduled according to the farthest-destination-first rule. Since we introduced the artificial packets in I', the resulting schedule for I is not longer than S. □

To prove Theorem 2.2 it now remains to show that $OPT(I') \leq C + D - 1$.

Claim. There is a schedule for I' whose length is at most $C + D - 1$.

We prove the claim by constructing such a schedule. Take a small path P' with bandwidth $b \geq 2$ (recall that all edges on a small path have the same bandwidth). We replace P' by b paths $P_1'', ..., P_b''$ of unit bandwidth. The packets using P' in I' are now distributed among the new paths such that no path is used by more than C packets. We do this transformation with each small path. By construction, any feasible schedule of the resulting instance can be transformed to a feasible schedule for I' with the same length.

Consider one connected component. Recall that it is a directed spider where each edge has unit bandwidth. We showed in Chapter 1 that for instances of the packet routing problem on directed trees there is always a schedule of length $C + D - 1$ which can be computed in polynomial time. This already implies the existence of a schedule with this bound for each component and a polynomial time algorithm for computing it. Nevertheless, for this special case we give a simpler direct proof.

We showed in Lemma 1.3 that any set of paths on a directed tree can be colored with C colors such that no two paths with the same color share an edge (where C denotes the maximal number of paths which share an edge). Such a coloring can be obtained by first reducing the problem on directed star graphs (i. e., directed trees with diameter two) to edge-coloring in bipartite multigraphs. Do this computation for each star. Then, the solutions for the star graphs are glued together to obtain a global solution. We assume in the sequel that each packet M_i is colored with a color $c_i \in \{0, ..., C - 1\}$ such that no two packets with the same color share an edge.

Routing Schedule. Using the coloring defined above, we define a schedule which finishes after at most $C + D - 1$ steps. We define v to be the single vertex in the component which has a larger degree than two (or an arbitrary vertex if there is no such vertex). For each vertex v' we define a value $h(v')$. If there is a directed path from v to v' of length ℓ then we define $h(v') := \ell$. If there is a directed path from v' to v of length ℓ then we define $h(v') := -\ell$. Note that since our component is a directed spider one of the two cases must apply.

Now we define our routing schedule. It is a direct schedule, i. e., every packet waits in its start vertex for some time and then proceeds to its destination without any further delay. We assign each packet M_i an initial waiting time of $d_i := (c_i + h(s_i)) \bmod C$. Note that this already characterizes the schedule completely.

Lemma 2.4. *The defined schedule is feasible and finishes after at most* $C+D-1$ *steps. Hence,* $OPT(I') \leq C + D - 1$.

Proof. Assume on the contrary that at time t two packets $M_i, M_{i'}$ collide when leaving some vertex v'. Hence, $t = d_i - h(s_i) + h(v') = d_{i'} - h(s_{i'}) + h(v')$. This implies that $(c_i + h(s_i)) \bmod C - h(s_i) = (c_{i'} + h(s_{i'})) \bmod C - h(s_{i'})$ and hence $c_i \equiv c_{i'} \bmod C$. However, this contradicts that the c_i values form a valid coloring.

By definition, each packet waits in its start vertex for at most $C - 1$ steps and then moves to its destination. Hence, the length of the schedule is bounded by $C+D-1$. Doing the described adjustments with every connected component shows that $OPT(I') \leq C + D - 1$. □

Proof of Theorem 2.2. According to Lemma 1.3 the path coloring can be computed in polynomial time. The remaining operation can also be computed efficiently. Then the claim follows from Lemmas 2.3 and 2.4. □

Remark. At first glance our technique might seem to work also if D can be as large as 2τ. Unfortunately, this is not the case. Recall that we allow the packets to start in the middle of a short path (we will need this property later in our framework for general instances). If there is a packet whose path uses more than $\tau + 1$ edges then its path could cross three small paths. Then, our technique to reduce the problem to spiders does not work any longer. However, if we do not allow the packets to start in the middle of short paths then our bound also holds if we require only that $D \leq 2\tau$.

2.3 High Level Ideas for General Bounds

In the previous section, we provided a tight bound for the length of optimal schedules of instances of the packet routing problem where $D \leq \tau+1$. Unfortunately, the vast majority of instances does not fall under this category. However, we provide an upper bound for *all* instances. In order to do this, we provide a framework which uses bounds for instances with small dilation (like $D \leq \tau + 1$) for proving good bounds for all instances. Using our bounds for instances with $D \leq \tau + 1$ from the previous section, we prove the following theorems.

Theorem 2.5. *There is a function* $f(\tau, b)$ *which tends to 1 if* τ *or* b *increases such that for each packet routing instance* I *with minimum bandwidth* b *and minimum transit time* τ *there is a feasible schedule whose length is bounded by*

$$\left(6.44 \cdot f(\tau + 1, b) + 2.11\frac{\tau}{\tau + 1} + \delta\right)(C + D)$$

for $\delta = \frac{1}{60}(\tau + 3.05 \cdot f(\tau + 1, b)(\tau + 1))$.

Note that the value δ in the theorem above increases for increasing τ. However, if $\tau \geq 63$ the following theorem gives an alternative bound.

Theorem 2.6. *Let I be an instance of the packet routing problem with minimum transit time $\tau \geq 63$. Then there is a feasible schedule for I whose length is bounded by $4.32(C + D)$.*

Even more, assuming that one has good upper bounds for instances with small dilation we present a framework which gives good upper bounds for *all* instances.

Theorem 2.7. *There is a function $f(\ell, b)$ which tends to 1 if ℓ or b increases such that for every instance I of the packet routing problem with minimum bandwidth b and minimum transit time τ there is a feasible schedule for I whose length is bounded by*

$$A^{b,\tau}\left(3.05\ell \cdot f(\ell, b), \ell\right) \cdot \left(\frac{2.11}{\ell} + \delta\right)(C + D)$$

for $\delta = \frac{1}{60}$.

For using the above theorem, it suffices to have a good bound for the quantity $A^{b,\tau}\left(3.05\ell \cdot f(\ell, b), \ell\right)$ for some (small) value ℓ. As an application of this framework, the proof of Theorem 2.5 uses that $A^{b,\tau}(C, \tau+1) = C+\tau$ as proven in Theorem 2.2 (here we choose $\ell := \tau + 1$). Also, for the important special case of unit bandwidths and unit transit time (i. e., $b = 1$ and $\tau = 1$) our framework gives the following bound.

Theorem 2.8. *Let I be an instance of the packet routing problem with unit transit times and unit bandwidths. Then there is a feasible schedule for I whose length is bounded by $23.4(C + D)$.*

Table 2.1 shows the bounds obtained by the above theorems for certain values of b and τ. Figure 2.2 shows a plot of the function f used in Theorems 2.5 and 2.7 for some values b.

In this section we describe the high-level concepts of our proofs. The complete proofs requires many technical details which we give in Section 2.4. Our reasoning uses the concepts introduced by Scheideler [92] who proved that there is always a schedule of length $39(C + D)$ for instances with unit transit times and unit capacities. At several times in our analysis, we make use of the Lovász Local Lemma (LLL) which (in its symmetric version) states the following:

Lemma 2.9 (Lovász Local Lemma (LLL) [82])**.** *Let $E_1, \ldots, E_k$ be a set of "bad events" in an arbitrary probability space. Assume that each event occurs with probability at most p. Also, assume that each bad event is mutually independent of all other but at most d events. If $ep(d + 1) < 1$ then, with probability greater than zero, no bad event occurs.*

The LLL as stated above is non-constructive. However, Moser and Tardos [79] recently gave an algorithmic version, even for the general LLL. It requires only that a violated bad event and the variables that it depends on can be detected efficiently. This is immediate in our application of the LLL as we

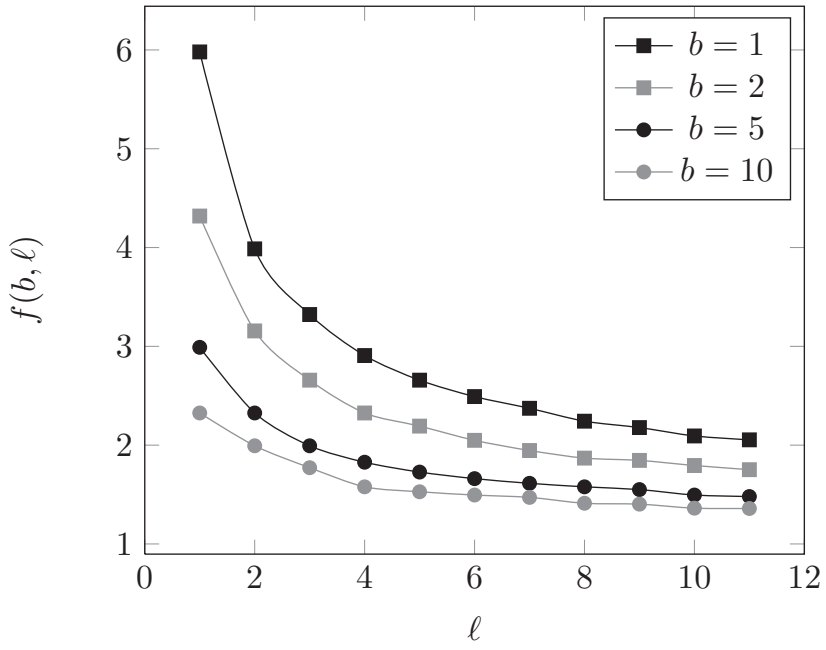


Figure 2.2: The function f for the values of b and ℓ which are relevant for the bounds shown in Table 2.1 and for Theorems 2.5 and 2.7.

will see in the sequel. Also, our bound for instances with $D \leq \tau + 1$ given in Theorem 2.2 is constructive (given by a polynomial time algorithm). We need the latter insight to obtain the bounds in the above theorems. Hence, we also obtain efficient algorithms which compute schedules with the stated lengths. We say that a randomized algorithm runs in *polynomial time with high probability* if for each $k \in \mathbb{N}$ there is a polynomial $p(n)$ such that the algorithm needs at most $p(n)$ steps with probability $1 - \frac{1}{k}$. We obtain the following corollary.

Corollary 2.10. *There are randomized algorithms which compute schedules with the bounds stated in Theorems 2.5, 2.6, and 2.8 and which run in polynomial time with high probability. If there is a polynomial time algorithm which computes a schedule of length at most $A^{b,\tau}(3.05\ell \cdot f(\ell, b), \ell)$ for all instances with $C \leq 3.05\ell \cdot f(\ell, b)$ and $D \leq \ell$ then there is a randomized algorithm that computes a schedule whose length is given in Theorem 2.7 and which runs in polynomial time with high probability.*

In the first part of our proof, we give a careful adaption of the concepts introduced by Scheideler [92] to the setting of arbitrary bandwidths and transit times. In the second part of the proof we introduce our framework. Any good bound for instances with small dilation, e. g., $D \leq \tau+1$, $D \leq 2\tau+1$, etc., allows the framework to prove better bounds for general instances (with arbitrary dilation). We incorporate the bounds for instances with $D \leq \tau + 1$ (obtained in Section 2.2) into our framework. In the setting of unit bandwidths and transit times we improve the bound of $39(C + D)$ by Scheideler [92] to $23.4(C + D)$. For larger b and/or larger τ we can reduce the constant in front of $(C + D)$ even further.

In the sequel, we will use the concept of infeasible schedules. We call a schedule *infeasible* if some edge e is used by more than b_e packets at a time, but the packets still follow their predefined paths and respect the transit times of the edges. For ease of notation we say a schedule is infeasible if it is not necessarily feasible. We use the following strategy: We start with an infeasible schedule in which no packet is ever delayed. Denote this schedule by S_0. We perform the following steps:

- One can enlarge S_0 by adding a random delay of at most C/b for each packet at the beginning of S_0, yielding a schedule S_1. We will show using the LLL that there are delays for the packets such that S_1 fulfills certain properties.

- Inductively, assume that we have an infeasible schedule S_i (with $i \geq 1$). Using the LLL we show that there are further refinement steps yielding a schedule S_{i+1} which is – intuitively speaking – "more feasible" than S_i.

- Eventually, we prove that there is a schedule S_k with the property that in every interval of length 64 at most $195.1b$ packets use each edge. Furthermore, we prove that the length of S_k is bounded by $1.0626(C + D)$.

Starting with the infeasible schedule S_k, we establish our framework. Let $\ell \in \mathbb{N}$. Using the LLL we show that there is an infeasible schedule S_{k+1} that can be partitioned such that in any time-interval of length ℓ at most C_b^ℓ packets traverse each edge with bandwidth b (for a constant C_b^ℓ to be defined later). Hence, we can turn S_{k+1} into a feasible schedule by refining each interval of length ℓ separately. In order to do this, we treat each of these intervals as a subinstance of the packet routing problem with dilation ℓ and congestion $\max_e \left\{C_{b_e}^\ell / b_e\right\}$. Hence, it suffices to have good bounds for instances with dilation $D = \ell$ in order to obtain a bound for the original instance. We use our framework with $\ell := \tau + 1$ since Theorem 2.2 gives a bound for instances with $D \leq \tau + 1$. Using the framework one could obtain even better bounds for general instances if one had good upper bounds for instances with slightly higher dilation, e. g., $D \leq k\tau + 1$ for some small value k. In particular, the larger we can choose ℓ, the better our bounds become. This can be seen in Table 2.1 since for increasing τ the bounds improve. Also, if b increases the bounds improve as well. The reason is that C_b^ℓ / b decreases when b increases. Hence, the congestion in the subinstances (given as $\max_e \left\{C_{b_e}^\ell / b_e\right\}$ above) will be smaller for larger values of b.

In the following section we give a detailed technical analysis of the reasoning described above.

2.4 Technical Analysis

In this section we give the full proofs of the theorems and techniques described in Section 2.3. First, we adapt the concepts of Scheideler [92] to the setting

of arbitrary bandwidths and transit times. Then, we introduce our framework which then allows us to prove our bounds for the lengths of optimal schedules.

Let I be an instance of the packet routing problem with congestion C and dilation D. Assume that each edge has a bandwidth of at least b and a transit time of at least τ. Our bounds depend on these four parameters. In particular, they improve for larger b or τ. As already mentioned in Section 2.2, we replace every edge e with transit time τ_e by a path consisting of τ_e edges with unit transit time. Also, paths of packets are allowed to start or end on the new (small) vertices.

First, we prove the existence of the schedule S_k with the property that in every interval of length 64 at most $195.1b$ packets use each edge. We bound the length of S_k later. We define $I_0 := \max\{C, D\}$. Let $k := (\log^* I_0) - 1$ where for our purposes we define $\log^* I_0$ to be the smallest integer ℓ such that we need to apply the log-function ℓ times to I_0 in order to obtain a value of at most 4. We set $I_k := 4$ and $I_j := 2^{I_{j+1}}$ for all j with $1 \leq j \leq k-1$. Note that $2^{I_1} \geq I_0$. If $I_0 \leq 64$ then we define $S_k := S_0$. Hence, for the remaining reasoning for the construction of S_k we assume that $I_0 > 64$. Let S_0 be the infeasible schedule in which no packet is ever delayed. We define $D_0 := D$. We will prove the existence of schedules S_i with certain properties (with $i \geq 1$). We denote by D_i the length of S_i. Let C_i be the maximum number of packets that use an edge in each interval of length I_i^3 in the schedule S_i.

We start with the schedule S_0. We assign each packet an initial random delay. Using the Lovász Local Lemma we prove that there are random delays such that the resulting schedule is "relatively feasible". The schedule S_1 is the schedule resulting from those "good" initial delays.

Lemma 2.11. *There is an infeasible schedule S_1 with the property that in every interval of length I_1^3 at most $C_1 b_e$ packets use each edge e with $C_1 := (1 + \frac{3}{I_1})I_1^3$. Also, $D_1 \leq C + D$.*

Proof. We prove the existence of S_1 using the Lovász Local Lemma. We change S_0 by giving each packet an initial delay chosen uniformly at random from the set $\{0, ..., C-1\}$. This results in an infeasible schedule S_1 whose length is bounded by $(C + D)$. In order to make S_1 "relatively feasible" we want to ensure that in S_1 at most $C_1 b_e$ packets traverse every edge e during any interval of length I_1^3. We ensure this property by using the Lovász Local Lemma. We define a *bad event* E_e for each edge e. The bad event E_e is that there is an interval of length I_1^3 in which more than $C_1 b_e$ traverse e. Using the Lovász Local Lemma we show that there are initial delays for the packets such that no bad event occurs.

Note here that in polynomial time we can check whether a bad event occurs. If this is the case we can determine in polynomial time what random variables affect the occured bad event. This is necessary for the algorithmic version of the LLL [79]. The same will apply for all subsequent applications of the lemma.

Consider an edge e and an interval J of length I_1^3. Assume that the packets $M_1, ..., M_\ell$ use e (note that $\ell \leq Cb_e$). Let X_i be a binary random variable such that $X_i = 1$ if and only if M_i traverses e during J. We define $X := \sum_{i=1}^{\ell} X_i$.

The initial delays were chosen uniformly at random and hence $\Pr[X_i = 1] \leq I_1^3/C$ for each i. This implies $\mathbb{E}[X] = \sum_{i=1}^{\ell} \mathbb{E}[X_i] \leq I_1^3 b_e$. Using the Chernoff bounds [4, 49] we derive with $\varepsilon = \frac{3}{I_1}$ that

$$\begin{aligned} \Pr\left[X \geq (1+\varepsilon) I_1^3 b_e\right] &= \Pr\left[X \geq \left(1 + \frac{3}{I_1}\right) I_1^3 b_e\right] \\ &\leq \exp\left(-3 b_e I_1\right) \end{aligned}$$

where, as usual, $\exp(x) = e^x$. Since there are at most $(C + D)$ intervals of length I_1^3, the probability for E_e is bounded by

$$\Pr[E_e] \leq (C + D) \cdot \exp\left(-3 b_e I_1\right).$$

We can bound the dependence of each bad event E_e by $CDb_e - 1$: The event E_e for the edge e is clearly independent of a bad event $E_{e'}$ for an edge e' if there is no packet in the instance which uses both e and e'. The edge e is used by at most Cb_e packets. Each of these packets uses at most D edges in total. Hence, E_e is independent of all but at most $CDb_e - 1$ other bad events. In order to apply the Lovász Local Lemma we have to ensure that

$$\Pr[E_e] \cdot e \cdot CDb_e < 1.$$

Due to our bound for $\Pr[E_e]$ above it suffices to ensure that

$$(C + D) \cdot \exp\left(-3 b_e I_1\right) \cdot e \cdot CDb_e < 1.$$

We show the inequality for $b_e = 1$ (since then by monotonicity it holds for all b_e). Also, since I_1 depends only on $\max\{C, D\}$ it suffices to consider the case that $C = D$. This yields the inequality

$$2D \cdot \exp\left(-3 I_1\right) \cdot e \cdot D^2 < 1.$$

Since we assumed that $I_0 \leq 2^{I_1}$ and hence $D \leq 2^{I_1}$ it suffices to ensure that

$$2 \cdot 2^{I_1} \cdot \exp\left(-3 I_1\right) \cdot 2^{2 I_1} < 1. \tag{2.1}$$

Since $I_0 > 64$ we have in particular that $I_1 \geq 4$. Inequality 2.1 holds for $I_1 = 4$ and due to monotonicity also for all $I_1 \geq 4$. Concluding this lemma we obtained the infeasible schedule S_1 with the property that in each time interval of length I_1^3 each edge is used by at most $C_1 b_e$ packets. □

Denote by S_1 the schedule whose existence was proved in Lemma 2.11. Given an infeasible schedule S_i we want to prove the existence of a schedule S_{i+1} which is – intuitively speaking – "more feasible" than S_i. This is again done by giving each packet random delays. We use the Lovász Local Lemma to ensure that there are delays for the packets such that in any interval of length I_{i+1}^3 only a bounded number of packets use each edge.

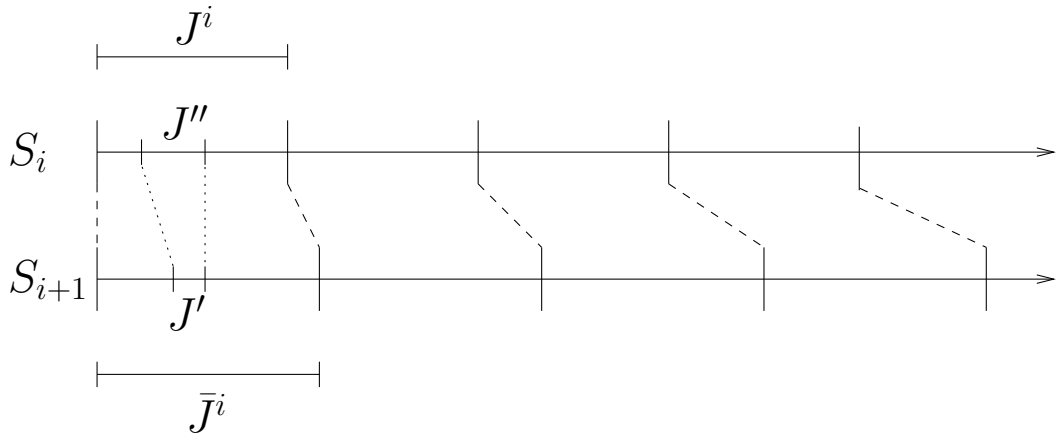


Figure 2.3: Each interval J^i of length I_i^4 in S_i is refined to an interval $\bar{J}^i$ of length at most $I_i^4 + I_i^3 - I_{i+1}^3$ in S_{i+1} (proof of Lemma 2.12).

Lemma 2.12. *Let S_i be an infeasible schedule of length D_i with the property that in every interval of length I_i^3 at most $C_i b_e$ packets use each edge e for some value $C_i \geq I_i^3$. Then there is an infeasible schedule S_{i+1} with the property that in every interval of length I_{i+1}^3 at most $C_{i+1} b_e$ packets use each edge e, with*

$$C_{i+1} := C_i \cdot \left(1 + \frac{5.1}{I_{i+1}}\right) \cdot \frac{I_{i+1}^3}{I_i^3 - I_{i+1}^3}.$$

Moreover, $D_{i+1} \leq \left(1 + \frac{1}{I_i}\right) D_i$ and $C_{i+1} \geq I_{i+1}^3$.

Proof. We split the timeline into intervals of length I_i^4. We refine the infeasible schedule S_i by enlarging each of these intervals. The schedule S_{i+1} is the concatenation of all enlarged intervals.

Let J^i be a time interval of length I_i^4. W. l. o. g. assume that $J^i = \left[0, I_i^4 - 1\right]$. At the beginning of J^i we define for each packet a delay chosen uniformly at random from the set of delays $\left\{0, 1, ..., I_i^3 - I_{i+1}^3 - 1\right\}$. Denote by $\bar{J}^i$ the resulting (enlarged) interval. Note that the length of $\bar{J}^i$ is bounded by $I_i^4 + I_i^3 - I_{i+1}^3$. See Figure 2.3 for a sketch. We want to ensure that in the resulting schedule within each interval of length I_{i+1}^3 at most $C_{i+1} b_e$ packets use each edge e. Using the Lovász Local Lemma we show that there are random delays for the packets such that this property is fulfilled.

Let e be an edge. We define that the bad event E_e^{i+1} occurs if there is a subinterval $J' \subseteq \bar{J}^i$ with length I_{i+1}^3 in which more than $C_{i+1} b_e$ packets use e. Let $J' = [x, x + I_{i+1}^3 - 1] \cap \bar{J}^i$ for some $x \in \bar{J}^i$. If after the random experiment a packet M uses e during J' then in S_i it must have used e during the interval $J'' = [x - I_i^3 + I_{i+1}^3, x + I_{i+1}^3 - 1]$. Since J'' has length I_i^3 there can be at most $C_i b_e$ such packets. Denote them by $M_1, ..., M_\ell$. Let X_i be a random variable such that $X_i = 1$ if and only if M_i uses e during J'. We define $X := \sum_{i=1}^{\ell} X_i$.

For each X_i we get $\Pr[X_i] \leq I_{i+1}^3 / \left(I_i^3 - I_{i+1}^3\right)$. We obtain

$$\mathbb{E}[X] = \sum_{i=1}^{\ell} \mathbb{E}[X_i] \leq C_i b_e \cdot \frac{I_{i+1}^3}{I_i^3 - I_{i+1}^3}.$$

The Chernoff bounds give that

$$\begin{aligned}\Pr[X \geq C_{i+1} b_e] &= \Pr\left[X \geq C_i \left(1 + \frac{5.1}{I_{i+1}}\right) \frac{I_{i+1}^3}{I_i^3 - I_{i+1}^3} b_e\right] \\ &\leq \exp\left(-\frac{1}{3}\left(\frac{5.1}{I_{i+1}}\right)^2 \cdot C_i \cdot \frac{I_{i+1}^3}{I_i^3 - I_{i+1}^3} b_e\right).\end{aligned}$$

Since there are at most $I_i^4 + I_i^3$ intervals of length I_{i+1}^3 (like the interval J') which we need to consider, we obtain

$$\Pr[E_e] \leq \left(I_i^4 + I_i^3\right) \exp\left(-\frac{1}{3}\left(\frac{5.1}{I_{i+1}}\right)^2 \cdot C_i \cdot \frac{I_{i+1}^3}{I_i^3 - I_{i+1}^3} b_e\right).$$

The interval J^i is the union of I_i intervals of length I_i^3. Therefore, by assumption during J^i (and by construction also during $\bar{J}^i$) at most $I_i \cdot C_i b_e$ packets pass e. Hence, the dependence of E_e is bounded by $I_i^5 \cdot C_i b_e - 1$. Therefore, in order to apply the Lovász Local Lemma, we need to guarantee that

$$\Pr[E_e] \cdot e \cdot I_i^5 \cdot C_i b_e < 1.$$

In order to analyze whether the inequality holds we study the function

$$g(I_i, C_i) := \left(I_i^4 + I_i^3\right) \exp\left(-\frac{1}{3}\left(\frac{5.1}{\log I_i}\right)^2 \cdot C_i \cdot \frac{\log^3 I_i}{I_i^3 - \log^3 I_i} b_e\right) e I_i^5 \cdot C_i b_e.$$

We need to show that $g(I_i, C_i) < 1$. First, we calculate that $\frac{dg}{dC_i} < 0$ and hence it suffices to ensure that $g\left(I_i, I_i^3\right) < 1$ (recall that $C_i \geq I_i^3$). Similarly, it suffices to show the bound for $b_e = 1$. We define $h(I_i) := g\left(I_i, I_i^3\right)$ and set $b_e := 1$ which yields

$$\begin{aligned}h(I_i) &= \left(I_i^4 + I_i^3\right) \exp\left(-\frac{1}{3}\left(\frac{5.1}{\log I_i}\right)^2 \cdot I_i^3 \cdot \frac{\log^3 I_i}{I_i^3 - \log^3 I_i}\right) e I_i^5 \cdot I_i^3 \\ &\leq \left(I_i^4 + I_i^3\right) \cdot I_i^8 \cdot e \cdot \exp\left(-8.6 \cdot \log I_i\right) \\ &=: \hat{h}(I_i).\end{aligned}$$

For $I_i = 16$ calculations show that $\hat{h}(I_i) < 1$ and $\frac{d\hat{h}}{dI_i} < 0$. Hence, if $I_i \geq 16$ then there are random delays for the packets such that in each interval $J' \subseteq \bar{J}^i$ of length at most I_{i+1}^3 there can be at most $C_{i+1} b_e$ packets using any edge e.

It remains to show that $C_{i+1} \geq I_{i+1}^3$. By assumption $C_i \geq I_i^3$. We calculate that

$$\begin{aligned} C_{i+1} &= C_i \cdot \left(1 + \frac{5.1}{I_{i+1}}\right) \cdot \frac{I_{i+1}^3}{I_i^3 - I_{i+1}^3} \\ &\geq I_1^3 \cdot \left(1 + \frac{5.1}{I_{i+1}}\right) \cdot \frac{I_{i+1}^3}{I_i^3 - I_{i+1}^3} \\ &= \frac{1}{1 - \frac{I_{i+1}^3}{I_i^3}} \cdot \left(1 + \frac{5.1}{I_{i+1}}\right) \cdot I_{i+1}^3 \\ &\geq I_{i+1}^3. \end{aligned}$$

□

We apply Lemma 2.12 iteratively until we have proven the existence of the schedule S_k with the respective properties. In particular, since $I_k = 4$ Lemma 2.12 shows that in S_k in every interval of length $4^3 = 64$ every edge is used by at most C_k packets. In the following two lemmas we bound C_k and D_k.

Lemma 2.13. *It holds that* $D_k < 1.0626(C + D)$.

Proof. Due to our refinement steps from Lemmas 2.11 and 2.12 we have that $D_k \leq (D + C) \prod_{i=1}^{k-1} \left(1 + \frac{1}{I_i}\right)$. For bounding the latter term we define a sequence a. Let $a_0 := 4$, $a_1 := 16$, $a_2 := 65536 = 2^{16}$, and $a_3 := 2^{65536}$. For $i \geq 4$ we define $a_{i+1} := 2a_i$. Our definition of the a_i implies that $\prod_{i=1}^{\infty} \left(1 + \frac{1}{I_i}\right) \leq \prod_{i=1}^{\infty} \left(1 + \frac{1}{a_i}\right)$. Therefore, it suffices to show that $\prod_{i=1}^{\infty} \left(1 + \frac{1}{a_i}\right) < 1.0625$. With the definition of the sequence a we are able to use the geometric series for bounding $\sum_{i=3}^{\infty} \frac{1}{a_i}$. Since the latter term is very small, defining $a_{i+1} := 2a_i$ does not introduce too much inaccuracy. Let $\alpha := \prod_{i=3}^{\infty} \left(1 + \frac{1}{a_i}\right)$. We calculate that

$$\begin{aligned} \log(\alpha) &= \sum_{i=3}^{\infty} \log\left(1 + \frac{1}{a_i}\right) \\ &\leq \sum_{i=3}^{\infty} \frac{1}{a_i} \\ &\leq 2^{-32768}. \end{aligned}$$

This implies that $\alpha \leq 2^{2^{-32768}}$. Calculations show that

$$\prod_{i=1}^{\infty} \left(1 + \frac{1}{a_i}\right) \leq \alpha \cdot \left(1 + \frac{1}{a_1}\right) \cdot \left(1 + \frac{1}{a_2}\right) < 1.0626.$$

□

Lemma 2.14. *It holds that* $C_k < 195.1$.

Proof. We calculate that

$$\begin{aligned} C_k &= \left(1+\frac{3}{I_1}\right) I_1^3 \cdot \prod_{i=1}^{k-1} \left(1+\frac{5.1}{I_{i+1}}\right) \cdot \frac{I_{i+1}^3}{I_i^3 - I_{i+1}^3} \\ &= \left(1+\frac{3}{I_1}\right) I_k^3 \cdot \prod_{i=1}^{k-1} \frac{1}{1-(I_{i+1}/I_i)^3} \left(1+\frac{5.1}{I_{i+1}}\right). \end{aligned}$$

Now we distinguish two cases: If $I_1 = 16$ then $k = 2$ and the last product has only one factor. Calculations show that then

$$\left(1+\frac{3}{I_1}\right) \cdot \frac{1}{1-(I_2/I_1)^3} \left(1+\frac{5.1}{I_2}\right) < 2.75.$$

If $I_1 > 16$ then in particular $I_1 \geq 2^{16}$. It can be shown that

$$\prod_{i=1}^{k-1} \frac{1}{1-(I_{i+1}/I_i)^3} \left(1+\frac{5.1}{I_{i+1}}\right) < 3.04$$

with a similar proof as for the bound of D_k in Lemma 2.13. Since $I_1 \geq 2^{16}$ we still have that $\left(1+\frac{3}{I_1}\right) \prod_{i=1}^{k-1} \frac{1}{1-(I_{i+1}/I_i)^3} \left(1+\frac{5.1}{I_{i+1}}\right) < 3.04$. Since $I_k^3 = 64$ in both cases we obtain the stated bound. □

Note that if $I_0 \leq 64$ then by definition $S_0 = S_k$ and also $D_k = D_0 = D < 1.0626(C+D)$ and $C_k = C_0 = C < 195.1$.

2.4.1 Framework

Having established the existence of the schedule S_k with the above properties we introduce our framework. The idea is the following: We split the schedule S_k into intervals of length $I_k^3 = 64$. We treat each of these intervals individually as a subinstance. Let F be such an interval. At the beginning of F we assign each packet a random delay from the range $\{0, 1, ..., 63\}$. The length of the resulting schedule is at most 127. Let $\ell \in \mathbb{N}$. Using the Lovász Local Lemma we show that there are random delays for the packets such that in each subinterval of length ℓ at most C_b^ℓ packets use each edge with bandwidth b (for a constant C_b^ℓ to be defined later). Denote by S_{k+1} such a schedule. Each subinterval of length ℓ can now be treated as a subinstance of the packet routing problem with dilation ℓ and congestion $\bar{C} := \max_e \left\{C_{b_e}^\ell / b_e\right\}$. Assume that we have a good bound $A^{b,\tau}\left(\bar{C}, \ell\right)$ for the maximum length of an optimal schedule for such an instance. This implies that by losing only a factor of (roughly) $A^{b,\tau}\left(\bar{C}, \ell\right)/\ell$ we can turn S_{k+1} into a feasible schedule. The length of S_{k+1} then gives us our bound on the length of an optimal schedule for the original instance. As an application of this framework we derive bounds by setting $\ell := \tau + 1$.

First, we define the values C_b^ℓ.

Definition 2.15. Let $b, \ell \in \mathbb{N}$. Consider $\lfloor 195.1b \rfloor$ binary random variables X_i such that $\Pr[X_i] = \frac{\ell}{64}$ and let $X := \sum_{i=1}^{\lfloor 195.1b \rfloor} X_i$. We define C_b^ℓ to be the minimum integer such that $\Pr\left[X > C_b^\ell\right] \cdot e \cdot \left\lceil \frac{1}{\ell} 127 \right\rceil \cdot \lfloor 195.1b \rfloor \cdot 64 \leq 1$. We write $\Pr\left(C_b^\ell\right) := \Pr\left[X > C_b^\ell\right]$.

Later we will split the whole time axis into intervals of length 127. We will split those again into even smaller intervals of length ℓ or less if ℓ. To simplify notation we introduce the notion of an ℓ-partition.

Definition 2.16. An *ℓ-partition* of an interval J with $|J| = 127 \cdot M$ (for an integer M) is a partition into $\left\lfloor \frac{127}{\ell} \right\rfloor \cdot M$ subintervals of length ℓ and M subintervals of length (k mod ℓ). In the sequel we call those subintervals *ℓ-subintervals*.

Using the Lovász Local Lemma, in the next lemma we show that there are random delays which turn S_k into the schedule S_{k+1} which is "almost feasible".

Lemma 2.17. *Let $\ell, b \in \mathbb{N}$. Assume we are given an infeasible schedule S_k of length D_k such that in every interval of length 64 each edge e is used by at most $\lfloor 195.1b \rfloor$ packets. Then there is an infeasible schedule S_{k+1} whose length is bounded by $D_{k+1} := D_k \cdot \frac{127}{64}$ that can be ℓ-partitioned such that in every ℓ-subinterval at most $C_{b_e}^\ell$ packets use each edge e.*

Proof. We split the timeline into intervals of length $I_k^3 = 64$. We consider each of these intervals separately and refine them. Let F be one of the intervals. At the beginning of F we perform a random experiment and assign each packet a delay chosen uniformly and independently at random from the set $\{0, 1, ..., 63\}$.

We define a *bad event* for each edge e. The bad event is that during an interval $[i \cdot \ell, (i+1) \cdot \ell - 1] \cap F$ for some i more than $C_{b_e}^\ell$ packets traverse the edge e. The interval F was enlarged to a length of at most 127. Hence, there are at most $\left\lceil \frac{1}{\ell} 127 \right\rceil$ intervals of the form $[i \cdot \ell, (i+1) \cdot \ell - 1] \cap F$. Also, the probability for each bad event is bounded by $\Pr\left(C_{b_e}^\ell\right) \cdot \left\lceil \frac{1}{\ell} 127 \right\rceil$. Each bad event is independent from all other but at most $C_k b_e \cdot 64 - 1$ other bad events (follows from the bound on the congestion and the dilation). The definition of $C_{b_e}^\ell$ implies that $\Pr\left(C_{b_e}^\ell\right) \cdot e \cdot \left\lceil \frac{1}{\ell} 127 \right\rceil \cdot (C_k b_e \cdot 64 - 1 + 1) < 1$. The Lovász Local Lemma implies that with non-zero probability no bad event occurs. Doing this reasoning for each interval F proves the existence of the schedule S_{k+1} with the desired properties. □

We can turn S_{k+1} into a feasible schedule by solving each subinstance induced by a ℓ-subinterval optimally. This increases the length of S_{k+1} at most by a factor of (roughly) $A\left(\bar{C}, \ell\right)/\ell$ with $\bar{C} = \max_e \left\{ \left\lceil C_{b_e}^\ell / b_e \right\rceil \right\}$. If we have a good bound for $A\left(\bar{C}, \ell\right)$ this yields a good bound for the length of an optimal schedule for the original instance.

In order to bound $A^{b,\tau}\left(\bar{C}, \ell\right)$ we need to estimate $\bar{C}$. As a first step, in the next lemma we upper-bound C_b^ℓ by a value $\tilde{C}_b^\ell$ that we will work with later.

Values for $\tilde{C}_b^\ell$				
	$\ell = 2$	$\ell = 3$	$\ell = 6$	$\ell = 11$
$b = 1$	23.8	29.5	44.7	67.1
$b = 2$	36.2	46.2	73.3	114.1
$b = 5$	67.0	88.8	148.9	242.3
$b = 10$	111.8	151.7	264.0	441.5

Table 2.2: Quantities $\tilde{C}_b^\ell$ for several values of b and τ. Note that $\tilde{C}_b^\ell$ is an upper bound for C_b^ℓ.

Lemma 2.18. *Let $b, \ell \in \mathbb{N}$. Then $C_b^\ell \leq \tilde{C}_b^\ell := (1 + \varepsilon_b^\ell)\mu_b^\ell$, where $\mu_b^\ell := \ell \cdot \frac{1}{64} \cdot 195.1b$ and ε_b^ℓ is the smallest real such that*

$$\left(\frac{e^{\varepsilon_b^\ell}}{\left(1+\varepsilon_b^\ell\right)^{1+\varepsilon_b^\ell}}\right)^{\mu_b^\ell} \left\lceil \frac{1}{\ell} \cdot 127 \right\rceil \cdot e \cdot 195.1b \cdot 64 \leq 1.$$

Proof. Consider the $\lfloor 195.1b \rfloor$ binary random variables X_i from the definition of C_b^ℓ and set $X := \sum_{i=1}^{\lfloor 195.1b \rfloor} X_i$. Then $\mathbb{E}[X] = \lfloor 195.1b \rfloor \frac{\ell}{64}$. The Chernoff bounds give that

$$p := \Pr\left[X > \left(1+\varepsilon_b^\ell\right)\mu_b^\ell\right] \leq \Pr\left[X \geq \left(1+\varepsilon_b^\ell\right)\mu_b^\ell\right] \leq \left(\frac{e^{\varepsilon_b^\ell}}{\left(1+\varepsilon_b^\ell\right)^{1+\varepsilon_b^\ell}}\right)^{\mu_b^\ell}.$$

By the choice of ε_b^ℓ this implies that

$$\Pr\left[X \geq C_b^\ell\right] \cdot e \cdot \left\lceil \frac{1}{\ell} 127 \right\rceil \cdot 195.1b \cdot 64 \leq 1.$$

□

Table 2.2 shows some values $\tilde{C}_b^\ell$ for some b and ℓ. We note that for these values for fixed ℓ and increasing b the values $\tilde{C}_b^\ell / b$ do not increase. We show in the following lemma that this holds in general. It will become useful later: it allows us to argue that $\bar{C} = \max_e \left\{\left\lceil C_{b_e}^\ell / b_e \right\rceil\right\} \leq \max_e \left\{\left\lceil \tilde{C}_{b_e}^\ell / b_e \right\rceil\right\} = \tilde{C}_b^\ell / b$, where b denotes the minimum bandwidth of any edge in the instance.

Lemma 2.19. *Let $b, b', \ell \in \mathbb{N}$ with $b \leq b'$. Then $\tilde{C}_b^\ell / b \geq \tilde{C}_{b'}^\ell / b'$.*

Proof. It suffices to show the claim for $b' := b + 1$. By definition, $\tilde{C}_b^\ell = (1 + \varepsilon_b^\ell) \cdot \frac{\ell}{64} \cdot 195.1b$ and $\tilde{C}_{b+1}^\ell = (1 + \varepsilon_{b+1}^\ell)\frac{\ell}{64} \cdot 195.1(b+1)$ with ε_b^ℓ and ε_{b+1}^ℓ as defined in Lemma 2.18. Hence, it suffices to show that $\varepsilon_{b+1}^\ell \leq \varepsilon_b^\ell$. By definition, we have that

$$195.1b \cdot 64 \cdot \left\lceil \frac{1}{\ell} \cdot 127 \right\rceil \cdot e \cdot \left(\frac{e^{\varepsilon_b^\ell}}{\left(1+\varepsilon_b^\ell\right)^{1+\varepsilon_b^\ell}} \right)^{\frac{\ell}{64} \cdot 195.1b} \leq 1. \tag{2.2}$$

To show that $\varepsilon_{b+1}^\ell \leq \varepsilon_b^\ell$ we calculate that

$$\begin{aligned} & 195.1(b+1) \cdot 64 \cdot \left\lceil \frac{1}{\ell} \cdot 127 \right\rceil \cdot e \left(\frac{e^{\varepsilon_b^\ell}}{\left(1+\varepsilon_b^\ell\right)^{1+\varepsilon_b^\ell}} \right)^{\frac{\ell}{64} \cdot 195.1(b+1)} \\ = \quad & 195.1b \cdot 64 \left\lceil \frac{1}{\ell} \cdot 127 \right\rceil \cdot e \cdot \left(\frac{e^{\varepsilon_b^\ell}}{\left(1+\varepsilon_b^\ell\right)^{1+\varepsilon_b^\ell}} \right)^{\frac{\ell}{64} \cdot 195.1b} \cdot K. \end{aligned}$$

with $K := \frac{(b+1)}{b} \left(\frac{e^{\varepsilon_b^\ell}}{\left(1+\varepsilon_b^\ell\right)^{1+\varepsilon_b^\ell}} \right)^{\frac{\ell}{64} 195.1}$. Hence, it remains to show that

$$K = \frac{b+1}{b} \left(\frac{e^{\varepsilon_b^\ell}}{\left(1+\varepsilon_b^\ell\right)^{1+\varepsilon_b^\ell}} \right)^{\frac{\ell}{64} 195.1} \leq 1 \tag{2.3}$$

or equivalently

$$\left(\frac{b+1}{b} \right)^b \left(\frac{e^{\varepsilon_b^\ell}}{\left(1+\varepsilon_b^\ell\right)^{1+\varepsilon_b^\ell}} \right)^{\frac{\ell}{64} \cdot 195.1b} \leq 1^b = 1. \tag{2.4}$$

From Inequality 2.2 we conclude that

$$\left(\frac{e^{\varepsilon_b^\ell}}{\left(1+\varepsilon_b^\ell\right)^{1+\varepsilon_b^\ell}} \right)^{\frac{\ell}{64} \cdot 195.1b} \leq \frac{1}{195.1b \cdot 64 \cdot \left\lceil \frac{1}{\ell} \cdot 127 \right\rceil \cdot e}.$$

For proving Inequality 2.3 it hence suffices to show that

$$\left(1 + \frac{1}{b}\right)^b \leq 195.1b \cdot 64 \cdot \left\lceil \frac{1}{\ell} \cdot 127 \right\rceil \cdot e.$$

We have that $\lim_{b\to\infty} \left(1+\frac{1}{b}\right)^b = e$. Moreover, since $195.1b \cdot 64 \cdot \left\lceil \frac{1}{\ell} \cdot 127 \right\rceil \cdot e \geq e$ for all $b \geq 1$ Inequality 2.3 holds. This implies that $\varepsilon_{b+1}^\ell \leq \varepsilon_b^\ell$. □

In the following lemma, we formalize how we can derive bounds for general instances using good bounds for $A^{b,\tau}(C,\ell)$ (for some value ℓ).

Lemma 2.20. *Let I be an instance of the packet routing problem with minimum bandwidths and transit times b and τ, respectively, and let $\ell, M \in \mathbb{N}$. Assume we are given an infeasible schedule S_{k+1} for I of length $127 \cdot M$ which is ℓ-partitioned such that every ℓ-subinterval is used by at most C_b^ℓ packets. Then there is a feasible schedule for I whose length is bounded by*

$$\left\lceil \frac{127}{\ell} \right\rceil \cdot M \cdot A^{b,\tau}\left(\max_e \left\{ \left\lceil C_{b_e}^\ell / b_e \right\rceil \right\}, \ell\right).$$

Proof. We change the given schedule S_{k+1} to a feasible schedule by refining each ℓ-subinterval of the ℓ-partition. Each ℓ-subinterval can be modeled as an instance of the packet routing problem. This subinstance has a dilation of at most ℓ and each edge e is used by at most $C_{b_e}^\ell$ packets. Hence, the congestion of this subinstance is $\bar{C} := \max_e \left\{ \left\lceil C_{b_e}^\ell / b_e \right\rceil \right\}$. According to our assumption concerning b and τ the schedule in each ℓ-subinterval can be refined to a feasible schedule of length $A^{b,\tau}\left(\max_e \left\{ \left\lceil C_{b_e}^\ell / b_e \right\rceil \right\}, \ell\right)$. This yields the bound stated in this lemma. □

Later, we will work with the values $\tilde{C}_b^\ell$ instead of the values C_b^ℓ. For giving a general theorem for the bounds derived by our framework, we study the value $\tilde{C}_b^\ell / b$.

Lemma 2.21. *It holds that*

$$\left\lceil \tilde{C}_b^\ell / b \right\rceil \leq 3.05\ell \cdot g(\ell, b)$$

for a function $g(\ell, b)$ which approaches 1 for increasing ℓ or b.

Proof. From Lemma 2.18 we derive that $\tilde{C}_b^\ell := (1+\varepsilon_b^\ell)\mu_b^\ell$ where $\mu_b^\ell := \ell \cdot \frac{1}{64} \cdot 195.1b$ and ε_b^ℓ is the smallest real such that

$$\left(\frac{e^{\varepsilon_b^\ell}}{\left(1+\varepsilon_b^\ell\right)^{1+\varepsilon_b^\ell}} \right)^{\mu_b^\ell \left\lceil \frac{1}{\ell} \cdot 127 \right\rceil \cdot e \cdot 195.1b \cdot 64} \leq 1.$$

We see that for fixed ℓ and increasing b, the value ε_b^ℓ decreases. Similarly, for fixed b and increasing ℓ, we also have that ε_b^ℓ decreases. We conclude that $\tilde{C}_b^\ell$ approaches $\mu_b^\ell = \ell \cdot \frac{1}{64} \cdot 195.1b \leq 3.05\ell \cdot b$ for increasing ℓ or increasing b. This implies that $\left\lceil \tilde{C}_b^\ell / b \right\rceil$ approaches 3.05ℓ for increasing ℓ or increasing b. □

Now we can prove our main theorem for the bounds derived by our framework (theorem restated).

Theorem 2.7. *There is a function $f(\ell, b)$ which tends to 1 if ℓ or b increases such that for every instance I of the packet routing problem with minimum bandwidth b and minimum transit time τ there is a feasible schedule for I whose length is bounded by*

$$A^{b,\tau}\left(3.05\ell \cdot f(\ell, b), \ell\right) \cdot \left(\frac{2.11}{\ell} + \delta \right)(C + D)$$

for $\delta = \frac{1}{60}$.

Proof. We have that $C_b^\ell \leq \tilde{C}_b^\ell$ for all ℓ and b. From Lemma 2.19 it follows that $\max_e \left\{ \left\lceil \tilde{C}_{b_e}^\ell / b_e \right\rceil \right\} \leq \left\lceil \tilde{C}_b^\ell / b \right\rceil$. With Lemmas 2.20 and 2.21 we obtain a bound of

$$\begin{aligned} & A^{b,\tau}\left(\left\lceil \tilde{C}_b^\ell / b \right\rceil, \ell\right) \cdot \left\lceil \frac{127}{\ell} \right\rceil \cdot \frac{D_k}{64} \\ < \quad & A^{b,\tau}\left(3.05\ell \cdot f(\ell, b), \ell\right) \cdot \left\lceil \frac{127}{\ell} \right\rceil . \frac{1.0626(C+D)}{64}. \end{aligned}$$

Note that here we used that $(C + D)$ is large as mentioned in the introduction since then $\left\lceil \frac{D_k}{64} \right\rceil \approx \frac{D_k}{64}$. Since D_k is *strictly* smaller than $1.0626(D + C)$ there is a value N_0 such that for all C, D with $(C + D) \geq N_0$ our bounds hold. To simplify the expression further we calculate that

$$\frac{1}{64} \cdot \left\lceil \frac{127}{\ell} \right\rceil \cdot 1.0626 \leq \frac{1.0626}{64} \cdot \frac{127 + \ell}{\ell} \leq \frac{2.11}{\ell} + \frac{1}{60}.$$

□

Theorem 2.2 allows us to bound the expression $A^{b,\tau}(C, \ell)$ if $\ell = \tau + 1$. Note here that the theorem also yields a polynomial time algorithm which computes a schedule of length $A^{b,\tau}(C, \ell)$ for the respective instances (this is important for the algorithmic version of our bounds given in Corollary 2.10). Using this insight we can use the theorem above to derive general bounds for all packet routing instances (theorem restated).

Theorem 2.5. *There is a function $f(\tau, b)$ which tends to 1 if τ or b increases such that for each packet routing instance I with minimum bandwidth b and minimum transit time τ there is a feasible schedule whose length is bounded by*

$$\left(6.44 \cdot f(\tau + 1, b) + 2.11 \frac{\tau}{\tau + 1} + \delta\right)(C + D)$$

for $\delta = \frac{1}{60}(\tau + 3.05 \cdot f(\tau + 1, b)(\tau + 1))$.

Proof. We choose $\ell := \tau + 1$ in Theorem 2.7. In Theorem 2.2 we proved that $A^{b,\tau}(C, \tau + 1) \leq C + \tau$. This gives a bound of

$$(3.05(\tau + 1) \cdot f(\tau + 1, b) + \tau) \cdot \left(\frac{2.11}{\tau + 1} + \delta'\right)(C + D)$$

for $\delta' = \frac{1}{60}$. Calculations show that this expression is upper-bounded by the expression stated in the theorem. □

Note here that – for values of $f(\tau+1, b)$ close to 1 – the formula stated in Theorem 2.5 gives better bounds if $\tau = 1$ than for higher values of τ. However, for small τ and b the bound of the formula improves as τ increases.

Observe that δ increases for increasing τ. This worsens the bound for very large values τ. However, in the following section we prove a much better bound for the case that $\tau \geq 63$.

2.4.2 High Values for τ

Given the schedule S_k, in our framework we used the Lovász Local Lemma to prove the existence of the schedule S_{k+1}. However, we can alternatively turn the schedule S_k to a feasible schedule directly. This is in particular useful for cases where τ is relatively large, as we will see in Theorem 2.6.

Theorem 2.22. *Let I be an instance of the packet routing problem with minimum bandwidth b and minimum transit time τ. Then there is feasible schedule for I whose length is bounded by*

$$\frac{1}{60}(C+D) \cdot A^{b,\tau}(196, 64).$$

Proof. Recall that we proved the existence of the schedule S_k which has the property that in every interval of length 64 each edge is used by at most $C_k b = 195.1b$ packets. Hence, there is a feasible schedule for I whose length is bounded by

$$\frac{D_k}{64} \cdot A^{b,\tau}(\lceil C_k \rceil, 64) < \frac{1.0626(D+C)}{64} \cdot A^{b,\tau}(196, 64)$$

which is bounded by the expression in the theorem statement. Note that here again we used that $(C+D)$ is large as mentioned in the introduction since then we have that $\lceil \frac{D_k}{64} \rceil \approx \frac{D_k}{64}$. Since D_k is *strictly* smaller than $1.0626(D+C)$ there is a value N_0 such that for all C, D with $(C+D) \geq N_0$ our bounds hold. □

Using our insight gained in Theorem 2.2 for $A^{b,\tau}(C_k, \tau+1)$ allows us to prove the following corollary (given as Theorem 2.6 above).

Corollary 2.23. *Let I be an instance of the packet routing problem with minimum transit time $\tau \geq 63$. Then there is a feasible schedule for I whose length is bounded by $4.32(C+D)$.*

Proof. For $\tau \geq 63$ we have that $64 \leq \tau + 1$. Hence, Theorem 2.2 implies that $A^{b,\tau}(196, 64) \leq 196 + 64 - 1 = 259$ if $\tau \geq 63$. Plugging this into the bound of Theorem 2.22 yields the bound of $4.32(C+D)$. □

Table 2.1 shows some bounds for the lengths of schedules depending on τ and b.

2.4.3 Unit Transit Times and Unit Bandwidths

Finally, we use the above framework to derive a bound of $23.4(C+D)$ for the case of unit transit times and unit bandwidths. This improves the bound of $39(C+D)$ proven by Scheideler [92]. First, we precisely calculate that $C_1^2 = 21$ instead of using the estimation $\tilde{C}_1^2$. We did the calculation using the software MATLAB. We can do this exact calculation since C_1^2 does not depend on any parameter. Now we can use our framework together with the above lemma to derive our desired bound (theorem restated).

Theorem 2.8. *Let I be an instance of the packet routing problem with unit transit times and unit bandwidths. Then there is a feasible schedule for I whose length is bounded by $23.4(C + D)$.*

Proof. Let $\ell := 2$. If all edges have unit transit times and bandwidths then $\max_e \left\{ \left\lceil C_{b_e}^{\ell} / b_e \right\rceil \right\} = C_1^2$. Hence, in the reasonings of Theorem 2.7 and Theorem 2.5 we can use C_1^2 instead of $\tilde{C}_1^2$. From this we derive a bound of $\left(C_1^2 + 1\right) \cdot 1.0626(C + D)$. Using that $C_1^2 = 21$ implies the bound claimed in this theorem. □

2.4.4 Algorithmic Bounds

As mentioned above, Moser and Tardos [79] recently gave an algorithmic version of the general LLL. It requires that in a given variable assignment we can check in polynomial time for bad events: we need to be able to determine whether there is a bad event and if there is one we need to determine all variables that it depends on. In our applications of the LLL, there is only a polynomial number of bad events and hence we can check efficiently whether one of them occurs. Moreover, it is straight forward to determine efficiently the variables that a bad event depends on. Also, our bound for instances with $D \leq \tau + 1$ given in Theorem 2.2 is constructive. This gives the following corollary (restated).

Corollary 2.10. *There are randomized algorithms which compute schedules with the bounds stated in Theorems 2.5, 2.6, and 2.8 and which run in polynomial time with high probability. If there is a polynomial time algorithm which computes a schedule of length at most $A^{b,\tau}\left(3.05\ell \cdot f(\ell, b), \ell\right)$ for all instances with $C \leq 3.05\ell \cdot f(\ell, b)$ and $D \leq \ell$ then there is a randomized algorithm that computes a schedule whose length is given in Theorem 2.7 and which runs in polynomial time with high probability.*

Proof. In all our applications of the LLL the number of bad events is bounded by a polynomial in the input. Hence, the result by Moser and Tardos [79] is applicable. Each application of the LLL enlarges the length of the (intermediate) schedule. Since the final length of the schedule is bounded by $O(C + D)$, the number of times that we apply the LLL is bounded by a polynomial in the input.

For one application of the LLL, Moser and Tardos bound the expected number of resampling steps in their algorithm by $\sum_{A \in \mathcal{A}} \frac{x(A)}{1-x(A)}$ where $\mathcal{A}$ denotes the

set of all bad events. (In a resampling step a subset of the variables is redefined randomly.) Since we are using the symmetric version of the LLL, we have that $x(A) = 1/(d+1)$ with $d \geq 1$ [4]. Hence, in our case the expected number of resampling steps is bounded by $O(|\mathcal{A}|)$. One resampling step can be executed in polynomial time and the overall running time of the algorithm is dominated by the resampling procedure. Using the Chernoff bounds we can hence reason that each application of the LLL has polynomial running time with high probability. This implies that the overall algorithm also has polynomial running time with high probability. □

2.5 Conclusion

In this chapter we presented our framework for proving upper bounds for the length of an optimal packet routing schedule. We invoked the framework with the insight that $A^{b,\tau}(C, D) = C + D - 1$ if $D \leq \tau + 1$. If one had similar insights for instances with $D \leq k\tau + 1$ for some small values of k this would immediately give even better bounds. In particular, for the case of unit transit times the first step would be a tight bound for instances with $D = 3$. While this might look easy at first glance, in our attempts it turned out to be highly non-trivial to prove a bound that goes beyond $2C + D - 2$ (which follows easily from Theorem 2.2). On the other hand, we know no instance with $D = 3$ in which an optimal schedule needs more than $C + D$ steps. This connects to another important question in the area: find an instance of the packet routing problem with $OPT \geq k(C+D)$ for $k > 1$. To the best of our knowledge, no such instance is known. This leaves a large gap to the upper bound of $23.4(C + D)$. It is a challenging question to improve the lower bound.

Our proofs for the upper bounds highly use the Lovász Local Lemma. Due to its generality, it cannot exploit the entire structure of the respective underlying problem. We believe that what is really needed is a better understanding of the key properties of the packet routing problem. This might yield better upper bounds as well as better approximation algorithms.

Chapter 3

Complexity of Packet Routing

3.1 Introduction

In the previous chapters we studied approximation algorithms for the packet routing problem and the maximal length of an optimal schedule in terms of the lower bounds C and D. In this chapter we study complexity results for the problem.

In order to distinguish problems according to their computational difficulty, several complexity classes were introduced in the literature, see [83] for an introduction. The most important classes for discrete optimization are the classes P and NP. Roughly speaking, the class P contains all problems for which there is an exact algorithm with polynomial running time. Examples are the Shortest-Path and Maximum-Flow problems. Intuitively, the class NP contains all problems for which there is a *certificate* that asserts the existence of a solution or a solution with a certain quality which can be verified in polynomial time. The packet routing problem is in NP. There, the certificate is a schedule. Given a (possibly infeasible) schedule, it can be verified easily in polynomial time whether the given schedule is feasible and whether its length exceeds a given makespan.

In fact, the complexity classes are formally defined via Turing machines and formal languages. In particular, formally the classes P and NP do not contain problems but formal languages which can be decided by deterministic and nondeterministic Turing machines, respectively, in polynomial time. For instance, one language in NP is the language of all pairs (I, k) where I is an instance of the packet routing problem whose optimal makespan is at most $k \in \mathbb{N}$. Since a formal introduction of complexity theory is beyond the scope of this thesis, we refer the interested reader to [83].

It is easy to see that problems like the packet routing problem are in NP. However, it is not known whether every problem in NP is also contained in P. Indeed, one of the most important open problems in theoretical computer science is the question whether $P = NP$ or $P \neq NP$. Nevertheless, it is widely

assumed that $P \neq NP$. A helpful concept to estimate the complexity of a problem is *NP-hardness.* A problem is NP-hard if every problem in NP can be reduced to it, see [83] for a formal definition. This implies that a polynomial time algorithm for *any* NP-hard problem would yield a polynomial time algorithm for *every* problem in NP. However, no such algorithm is known for any NP-hard problem and it seems unlikely that one exists. Hence, once a problem is proven to be NP-hard this justifies to search for approximation algorithms rather than exact polynomial time algorithms. In fact, sometimes it can be shown that even certain approximation factors are NP-hard to achieve. For instance, the problem of scheduling jobs on unrelated machines to minimize the makespan (see Chapter 7) is known to be NP-hard to approximate within a factor of $3/2 - \varepsilon$ for any $\varepsilon > 0$ [69]. This means that the existence of a polynomial time approximation algorithm for the problem with such an approximation factor would imply that $P = NP$.

The results presented in this chapter are joint work with Britta Peis and Martin Skutella [84].

3.1.1 Outline of the Chapter

In this chapter we give NP-hardness results for the packet routing problem. As discussed above, these are important insights since they show us the limits of polynomial time algorithms and their possible approximation factors (always assuming that $P \neq NP$). In Section 3.2 we prove that it is NP-hard to approximate the problem within a factor of $6/5 - \varepsilon$, for any $\varepsilon > 0$. This rules out the existence of a polynomial time approximation scheme (PTAS) for the problem. Interestingly, the latter even holds for the very restricted graph class of directed trees. There, it is NP-hard to approximate the problem with a factor of $8/7 - \varepsilon$, for any $\varepsilon > 0$, as we will prove in Section 3.3.

As the above factors suggest, the reductions create a gap of one time unit between yes- and no-instances of the problem that we reduce from (which is 3-Bounded-3-Sat in this case). For instance, for general graphs we give a reduction such that for satisfiable 3-Bounded-3-Sat formulea the constructed packet routing instance has an optimal makespan of 5 whereas for unsatisfiable formulae each schedule has a makespan of at least 6. This raises the question whether one can construct an approximation algorithm with an *absolute* approximation guarantee. In Section 3.4 we answer this question by showing that for any $k \in \mathbb{N}$ it is NP-hard to approximate the packet routing problem with an absolute error of k.

We conclude this chapter with Section 3.5 where we discuss open problems.

3.2 General Graphs

First, we study the complexity of the packet routing problem on general graphs. In the following theorem we rule out the existence of a PTAS for the problem and even any better approximation factor than 6/5 (assuming that $P \neq NP$).

Theorem 3.1. *It is NP-hard to approximate the packet routing problem with an approximation factor of* $6/5 - \varepsilon$ *for all* $\varepsilon > 0$.

We describe a reduction which proves Theorem 3.1. In this proof we employ a technique which was used in [108] for showing that the general acyclic job shop problem is NP-hard to approximate with an approximation factor better than 5/4. We reduce from 3-Bounded-3-Sat. In this problem we are given a 3-Sat formula in which each variable occurs at most three times. The question is whether there is an assignment for the variables such that the formula is satisfied. It was shown in [43] that 3-Bounded-3-Sat is NP-hard. We assume that we are given a 3-Bounded-3-Sat formula ϕ. We construct an instance of the packet routing problem corresponding to ϕ. We will show that ϕ is satisfiable if and only if the length of an optimal schedule for the constructed packet routing instance is at most 5. In particular, this means that if ϕ is not satisfiable, an optimal schedule for its corresponding instance has a makespan of at least 6. This implies that an approximation algorithm for packet routing with an approximation factor of $6/5 - \varepsilon$ for some $\varepsilon > 0$ could deceide whether ϕ is satisfiable.

Denote by $\{Q_1, Q_2, ..., Q_m\}$ the clauses of ϕ. W. l. o. g. we assume that each variable occurs at most two times as a positive literal and at most two times as a negative literal in the formula. If a variable occurs three times positive then we simply set it true and thus satisfy all the clauses containing this variable. If the variable occurs three times negative then we set the variable false. We also assume that each clause contains two or three variables.

In the sequel we will explain how we construct our graph and the packets with their start and destination vertices from ϕ. Figure 3.1 shows a part of our construction for the formula $(x \vee y \vee z)$. For each variable x we introduce the vertices $v_{x,1}, v_{x,2}, ..., v_{x,11}$. For each clause Q there is a vertex v_Q and a vertex v'_Q. If Q contains only two variables there is an additional vertex v''_Q. We add all the edges to the graph which are necessary for the packets according to their predefined paths (we define the paths below).

Now we introduce the packets. For each variable x we have four packets labeled $x_1, \bar{x}_1, x_2, \bar{x}_2$. We will call them the *variable packets* later on. These packets encode whether x is set to true or false in the variable assignment which corresponds to the schedule. The intuition is that a variable x is set to true if x_1 and x_2 are scheduled at time $t = 0$ and $\bar{x}_1$ and $\bar{x}_2$ are scheduled at time $t = 1$. If $\bar{x}_1$ and $\bar{x}_2$ are scheduled at time $t = 0$ and x_1 and x_2 are scheduled at time $t = 1$ then x is set to false. The predefined paths for the variable packets will ensure that other schedules for the variable packets would cause the overall schedule to be longer than 5. For each occurrence of the variable x in a clause we have a packet. We denote by $q_{x,i}$ the packet for the i-th positive occurrence of the variable x and by $q_{\bar{x},i}$ the packet for the i-th negative occurrence of the variable x. We will call those packets the *clause packets*. In a satisfying variable assignment there must be at least one variable x (or $\bar{x}$ respectively) in each clause Q which satisfies Q. The intuition is that the packet $q_{x,j}$ (or $q_{\bar{x},j}$ respectively) corresponding to this variable x (or $\bar{x}$ respectively) is scheduled

Packet	Path	
Packets for variables x		
x_1	$v_{x,1}, v_{x,3}, v_{x,5}, v_{x,7}, v_{x,10}$	for each variable x
x_2	$v_{x,2}, v_{x,4}, v_{x,5}, v_{x,6}, v_{x,8}$	for each variable x
$\bar{x}_1$	$v_{x,1}, v_{x,3}, v_{x,5}, v_{x,6}, v_{x,9}$	for each variable x
$\bar{x}_2$	$v_{x,2}, v_{x,4}, v_{x,5}, v_{x,7}, v_{x,11}$	for each variable x
Clause packets for clauses Q with three variables		
$q_{x,1}$	$v_Q, v'_Q, v_{x,7}, v_{x,10}$	where Q contains the first pos. literal x
$q_{x,2}$	$v_Q, v'_Q, v_{x,6}, v_{x,8}$	where Q contains the second pos. literal x
$q_{\bar{x},1}$	$v_Q, v'_Q, v_{x,6}, v_{x,9}$	where Q contains the first neg. literal $\bar{x}$
$q_{\bar{x},2}$	$v_Q, v'_Q, v_{x,7}, v_{x,11}$	where Q contains the second neg. literal $\bar{x}$

Table 3.1: The predefined paths of the packets for the proof of Theorem 3.1. Clause tasks of clauses Q with only two literals visit one additional vertex v''_Q between v_Q and v'_Q.

last among the packets corresponding to Q (namely at time $t = 2$). In Table 3.1 we define the predefined paths of the packets.

Now we want to prove that the length of an optimal schedule is at most 5 if and only if ϕ is satisfiable.

Lemma 3.2. *If ϕ is satisfiable then there is a schedule whose length is at most 5.*

Proof. Assume that ϕ is satisfiable and consider a variable assignment such that ϕ is satisfied. We construct a schedule as follows. For each variable x which is set true in the satisfying variable assignment we schedule the packets x_1 and x_2 at time 0 and the packets $\bar{x}_1$ and $\bar{x}_2$ are scheduled at time 1. For each false variable y we schedule the packets y_1 and y_2 at time 1 and the packets $\bar{y}_1$ and $\bar{y}_2$ at time 0. Since our assignment of the variables satisfies the formula, for each clause Q there has to be one literal x which satisfies the clause. We schedule the clause packets corresponding to the literals in Q in such a way that the clause packet $q_{x,i}$ (or $q_{\bar{x},i}$) for $i \in \{1, 2\}$ corresponding to the literal x in Q leaves v_Q at time $t = 2$. We schedule the two other packets of Q in an arbitrary order at times $t = 0$ and $t = 1$.

Occurring collisions (i. e., situations where two or more packets need to use the same edge at a time) are resolved arbitrarily. We want to show that the length of this schedule is 5. Let x_i be a variable packet which leaves its start vertex at time $t = 0$. It can be delayed at most once in our schedule, namely by a clause packet which leaves its start vertex at time $t = 1$. Thus x_i reaches its destination vertex after at most 5 timesteps. Now let x_i be a variable packet which left its start vertex at time $t = 1$. This implies that x was set to false in the variable assignment. Thus, there can be no clause packet $q_{x,i}$ which left its start vertex at time $t = 2$. Therefore, x_i will not be delayed on its way and will reach its destination vertex after 5 timesteps. For "negative" variable packets $\bar{x}_i$ we can apply the same reasoning. Now let $q_{x,i}$ be a clause packet

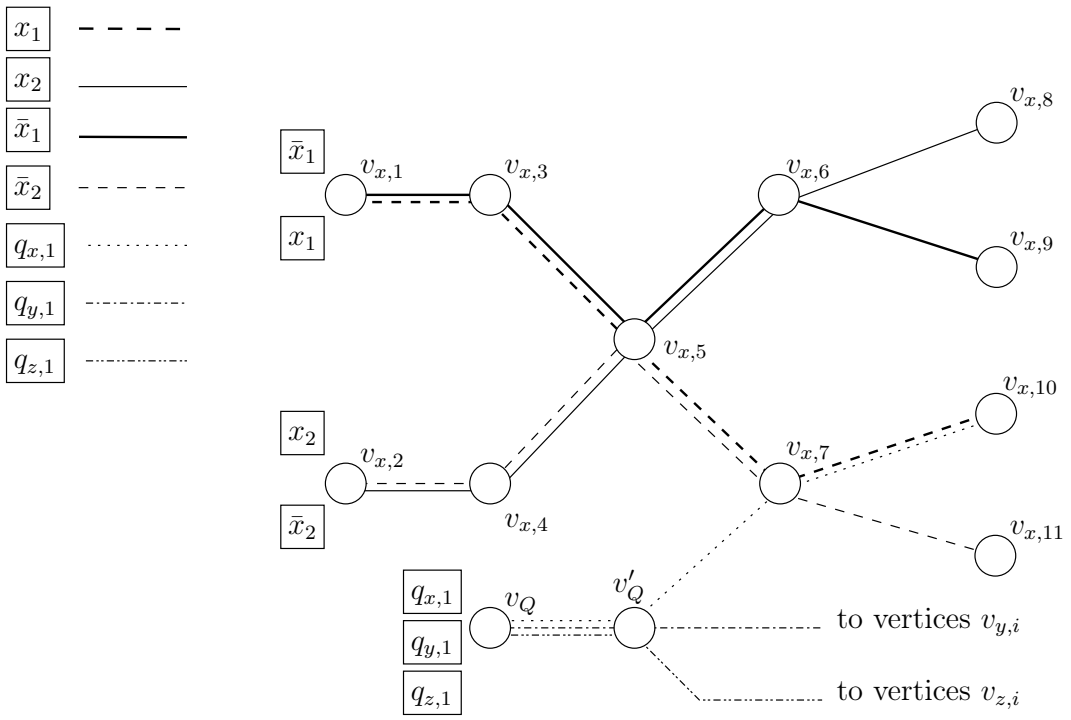


Figure 3.1: A part of the graph for the formula ϕ with only the clause $Q = (x \vee y \vee z)$. The different lines represent the paths of the packets through the network. The parts corresponding to the variables y and z and their packets are omitted for brevity.

of a clause with three literals. If $q_{x,i}$ was scheduled at time $t = 0$ it will not be delayed at all and will reach its destination after 3 timesteps. If $q_{x,i}$ was scheduled at time $t = 1$ it can be delayed by at most one variable packet (by at most one timestep) and will therefore reach its destination after at most 5 timesteps. Finally, if $q_{x,i}$ was scheduled at time $t = 2$ we know that x was set true in the variable assignment. This implies that the packets x_1 and x_2 were scheduled at time $t = 0$ and will not be able to delay $q_{x,i}$. Thus, $q_{x,i}$ will reach its destination vertex at time $t = 5$. We can apply the same reasoning to clause packets $q_{\bar{x},i}$ for negated literals. For clause packets corresponding to clauses with only two literals we can apply a similar reasoning: The two packets behave like the last two packets which are scheduled in a vertex v_Q where Q is a clause with three literals. □

Now we want to prove the implication in the opposite direction.

Lemma 3.3. *If there is a schedule whose makespan is at most 5 then ϕ is satisfiable.*

Proof. Assume we are given a schedule whose makespan is at most 5. We want to show that ϕ is satisfiable. In order to do this, we show how we can construct a satisfying variable assignment from the schedule. W. l. o. g. we assume that the schedule never delays a packet if it is the only packet that needs to use an edge at a time. Consider a variable x and its packets $x_1, x_2, \bar{x}_1$, and $\bar{x}_2$. Since

x_1 and $\bar{x}_1$ have the same start vertex, either x_1 is scheduled at time 0 and $\bar{x}_1$ is scheduled at time 1 or vice versa. The same holds for x_2 and $\bar{x}_2$. Since the length of the schedule is at most 5, we see that those of the packets which are scheduled at time 1 are not delayed later in the schedule. From this it follows that x_1 and x_2 are scheduled at the same time. If they are scheduled at time 0 we set the variable x to true, otherwise we set x to false. We do this with all variables. Now we want to show that this leads to a satisfying assignment for the formula.

Let Q be a clause. We discuss the case where Q contains only positive literals. The other cases can be proven similarly. Let $Q = (x \vee y \vee z)$. Consider the three packets $q_{x,i}$, $q_{y,j}$, $q_{z,k}$ for Q. They all originate at the same vertex v_Q. We consider the packet of the three which was scheduled last (i. e., at time $t = 2$). W.l.o.g. let $q_{x,i}$ be this packet. Since the length of our schedule is at most 5 we conclude that $q_{x,i}$ was never delayed after time $t = 2$. Thus, the packet x_i must have been scheduled at time $t = 0$. Otherwise both x_i and $q_{x,i}$ would reach the vertex $v_{x,6}$ (or $v_{x,7}$ respectively) at time $t = 4$ and the total makespan of the schedule would be 6. This would be a contradiction.

Hence, we set the variable x to true in our variable assignment. Therefore, the clause Q is satisfied. Doing this reasoning for all clauses shows that ϕ is satisfied. □

The two previous lemmas for our construction prove Theorem 3.1.

Proof of Theorem 3.1. Follows from Lemmas 3.2 and 3.3 and the construction defined above. □

As a corollary we obtain the same result of the packet routing problem on directed graphs.

Corollary 3.4. *It is NP-hard to approximate the packet routing problem on directed graphs with an approximation factor of* $6/5 - \varepsilon$ *for all* $\varepsilon > 0$.

Proof. It is straightforward to see that we can orientate the edges of the underlying graph in the construction of Theorem 3.1 such that the packets always move in the direction of the edges (see Figure 3.2).

□

3.3 Trees

Now we study the complexity of the packet routing problem on trees. Recall that in Chapter 1 we presented a 2-approximation algorithm for the packet routing problem on undirected trees. In the following theorem we prove that the best approximation guarantee we can hope for is 8/7 (unless $P = NP$), even on directed trees. In particular, this rules out the existence of a PTAS even on directed trees (unless $P = NP$).

Theorem 3.5. *It is NP-hard to approximate the packet routing problem on directed trees with an approximation ratio of* $8/7 - \varepsilon$ *for any* $\varepsilon > 0$.

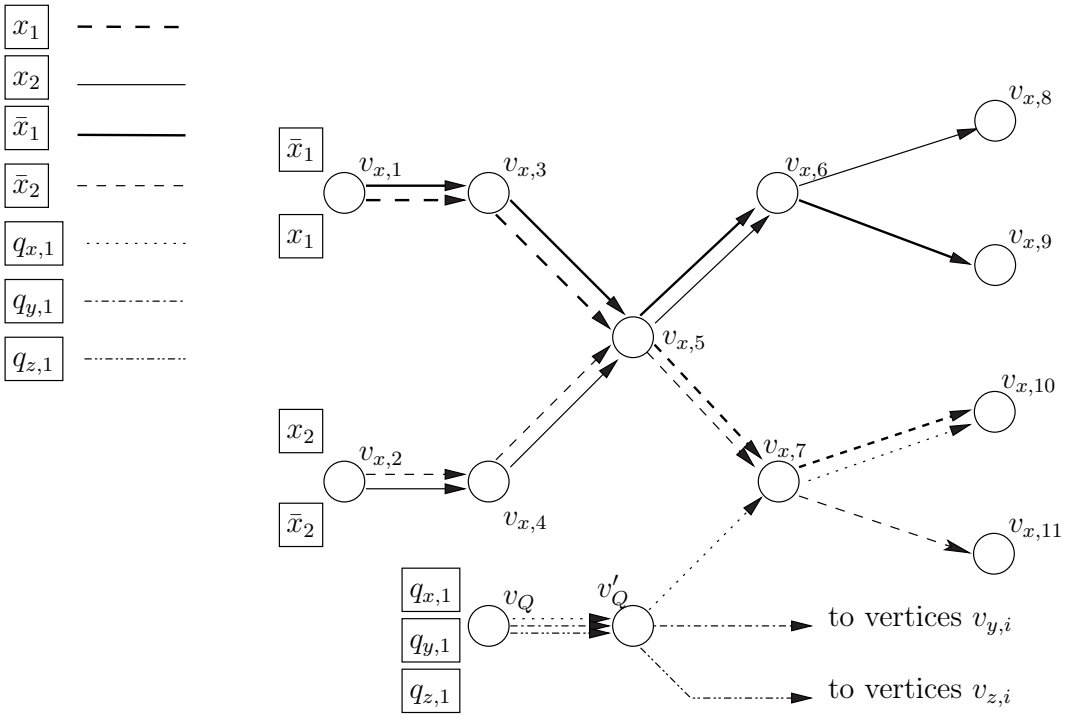


Figure 3.2: A part of the directed graph for the formula ϕ which consists only of the clause $(x \vee y \vee z)$.

Again, we reduce from 3-Bounded-3-Sat. Our construction is an adapted version of the construction for proving Theorem 3.1. Since the graph is a tree the approximation ratio that we can rule out (assuming $P \neq NP$) is slightly smaller.

Assume we are given a 3-Bounded-3-Sat formula ϕ. We construct an instance of the packet routing problem with fixed paths such that the underlying graph is a directed tree. In particular, the packets always move in the direction of the edges. We show that the formula is satisfiable if and only if the optimal schedule has length 7. As in the proof of Theorem 3.1, we assume that each clause contains at least two variables and that each variable occurs at most two times positive and at most two times negated in ϕ. The construction is sketched in Figure 3.3. The main difference in comparison with Theorem 3.1 is that now there is a central vertex v_r that all packets pass. For each variable x in the formula, there are the four variable packets x_1, x_2, $\bar{x}_1$ and $\bar{x}_2$. They can be scheduled at time $t = 0$ or $t = 4$. The intuition is that if x_1 and x_2 are scheduled at time $t = 0$ then we set the variable x true, if x_1 and x_2 are scheduled at time $t = 4$ then we set the variable x false. Note that now it could happen that this assignment is not well-defined, i. e., the packets x_1 and $\bar{x}_2$ go first or the packets x_2 and $\bar{x}_1$ go first. As in the proof of Theorem 3.1 our construction ensures that this leads to a schedule which is strictly longer than 7 timesteps. Moreover, for each literal in a clause Q there is one packet. Packets corresponding to literals in the same clause share the first edge on their path. Given a satisfying variable assignment for ϕ there is at least one literal in Q which satisfies the clause. The packet which corresponds to this literal is scheduled last.

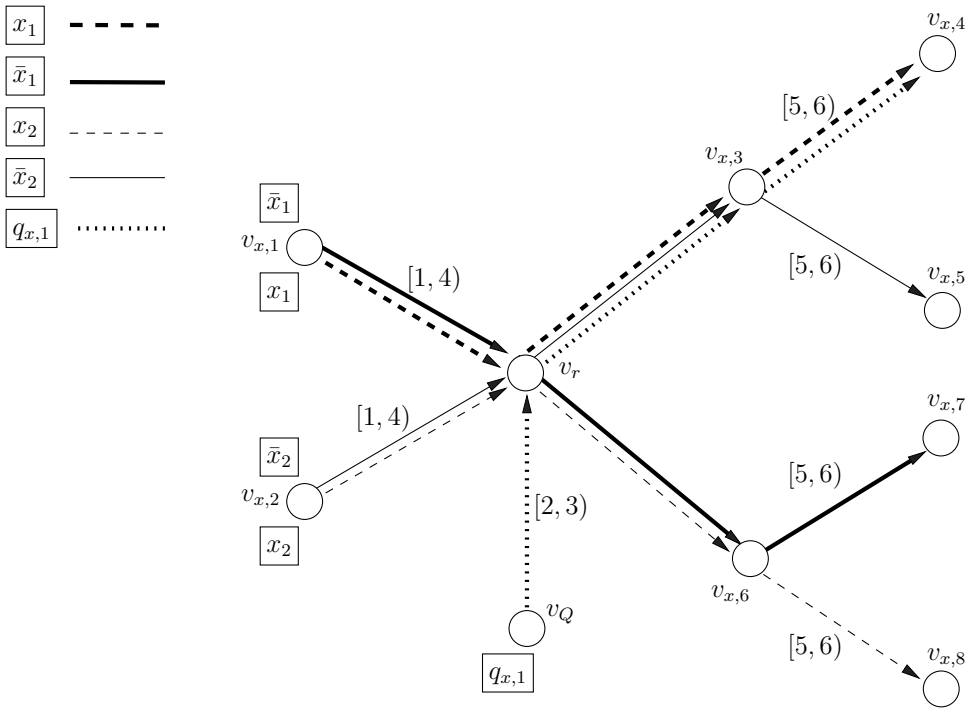


Figure 3.3: A part of the graph for the clause $Q = (x \vee y \vee z)$ (construction for Theorem 3.5). The parts corresponding to the variables y and z and their packets are omitted for brevity. The time intervals at the edges denote the time when the respective edge is blocked. Note that the first timestep is $t = 0$.

Packet	Path	
Variable packets for variable x		
x_1	$v_{x,1}, v_r, v_{x,3}, v_{x,4}$	for each variable x
x_2	$v_{x,2}, v_r, v_{x,6}, v_{x,8}$	for each variable x
$\bar{x}_1$	$v_{x,1}, v_r, v_{x,6}, v_{x,7}$	for each variable x
$\bar{x}_2$	$v_{x,2}, v_r, v_{x,3}, v_{x,5}$	for each variable x
Clause packets for clauses Q with three literals		
$q_{x,1}$	$v_Q, v_r, v_{x,3}, v_{x,4}$	where Q contains the first pos. literal x
$q_{x,2}$	$v_Q, v_r, v_{x,6}, v_{x,8}$	where Q contains the second pos. literal x
$q_{\bar{x},1}$	$v_Q, v_r, v_{x,6}, v_{x,7}$	where Q contains the first neg. literal $\bar{x}$
$q_{\bar{x},2}$	$v_Q, v_r, v_{x,3}, v_{x,5}$	where Q contains the second neg. literal $\bar{x}$

Table 3.2: The predefined paths of the packets for the proof of Theorem 3.5. Clause tasks of clauses Q with only two literals visit one additional vertex v'_Q between v_Q and v_r.

Now we describe the construction in detail. For each variable x we introduce vertices $v_{x,1}, v_{x,2}, ..., v_{x,8}$. We also introduce the four *variable packets* x_1, x_2, $\bar{x}_1$, and $\bar{x}_2$. In the formula, we fix an order of the clauses. For each clause Q there is a vertex v_Q. For each literal in Q there is one *clause packet*. We write $q_{x,i}$ for the packet corresponding to the i-th positive occurrence of x and $q_{\bar{x},i}$ for the packet corresponding to the i-th negative occurrence of x. The predefined paths for the packets are shown in Table 3.2 and sketched in Figure 3.3. Note that the underlying graph is a directed tree.

In order to make the construction work as desired we want to block some edges at certain times. We do this by introducing *blocking packets*. If we want to block an edge e at time $\bar{t}$ we introduce a blocking packet $b_{e,\bar{t}}$ which needs to use e at time $\bar{t}$ or otherwise would not reach its destination before time $t = 7$. Note that this does not destroy the tree structure of the underlying graph. In order to clarify matters we do not explicitly define the paths of the blocking packets. Instead, we state which edges are blocked at what times.

In our construction, the packets that "check" whether a schedule encodes a satisfying variable assignment for ϕ are the respective last packets that leave the vertices $v_{x,1}$, $v_{x,2}$ for each variable x and the vertex v_Q for each clause Q. If a schedule does *not* encode a satisfying assignment we want that two of these packets collide. The blocking packets help us to ensure this. In particular, they help separating the mentioned packets from the other packets.

Let x be a variable. The edges $(v_{x,1}, v_r)$ and $(v_{x,2}, v_r)$ are blocked during the time interval $[1, 4)$ (note that $t = 0$ is the first timestep). The edges $(v_{x,3}, v_{x,4})$, $(v_{x,3}, v_{x,5})$, $(v_{x,6}, v_{x,7})$, and $(v_{x,6}, v_{x,8})$ are blocked during the time interval $[5, 6)$. For each clause Q with three literals the edge (v_Q, v_r) is blocked during the time interval $[2, 3)$. For each clause Q' with two literals the edge $(v_{Q'}, v_r)$ is blocked during the time interval $[1, 3)$.

Now we want to prove that ϕ is satisfiable if and only if the optimal schedule has length 7. First assume that ϕ is satisfiable.

Lemma 3.6. *If ϕ is satisfiable then there is a schedule whose length is at most 7.*

Proof. Consider a variable assignment that satisfies the formula. We schedule the packets as follows: Whenever there is a packet M that is the only packet that needs to use the next edge on its path we move M. Let x be a variable. If x is set true in our variable assignment then we schedule the packets x_1 and x_2 at time $t = 0$ and the packets $\bar{x}_1$ and $\bar{x}_2$ at time $t = 4$. If x is set false then we schedule x_1 and x_2 at time $t = 4$ and $\bar{x}_1$ and $\bar{x}_2$ at time $t = 0$.

Since our variable assignment satisfies the formula in each clause Q there must be at least one literal y (or $\bar{y}$ respectively) which satisfies the clause. We schedule the packet corresponding to y (or $\bar{y}$ respectively) to leave v_Q at time $t = 3$. We schedule the other two packets to leave v_Q in arbitrary order at times $t = 0$ and $t = 1$ (if Q contains only two literals the other packet is scheduled at time $t = 0$). Blocking packets are never delayed by other packets. If there is a variable packet and a clause packet competing for using an edge at the

same time we give priority to the variable packet (this is only for simplification of the proof). Other ties are broken arbitrarily.

Now we show that all packets arrive at their destination vertices after at most 7 timesteps. First assume that x_1 started at time $t = 0$. Since the edge $(v_{x,2}, v_r)$ is blocked during the time interval $[1, 4)$ the packet $\bar{x}_2$ cannot reach v_r before $t = 5$. Thus, x_1 reaches $v_{x,6}$ at time $t = 2$ and $v_{x,8}$ at time $t = 3$. For the case that the packets x_2, $\bar{x}_1$, or $\bar{x}_2$ started at time $t = 0$ the proof works similarly.

Now assume that $\bar{x}_2$ started at time $t = 4$. The reasoning above implies that $\bar{x}_2$ cannot collide with x_1. Since variable packets have a higher priority than clause packets we conclude that $\bar{x}_2$ reaches $v_{x,6}$ at time $t = 6$ and $v_{x,8}$ at time $t = 7$. For the case that the packets x_1, x_2, or $\bar{x}_1$ started at time $t = 4$ the proof works similarly.

Next we discuss the clause packets. First consider a clause packet $q_{x,1}$ which is scheduled at time $t = 0$ or $t = 1$. In both cases it reaches $v_{x,6}$ at time $t = 4$ at the latest and $v_{x,8}$ at time $t = 5$ at the latest. (Note that either the packet x_1 or the packet $\bar{x}_2$ uses the edge $(v_r, v_{x,6})$ at time $t = 1$ and thus it does not make a difference for our reasoning whether $q_{x,1}$ was scheduled at time $t = 0$ or $t = 1$). We can do a similar reasoning for clause packets corresponding to negative literals.

Now assume that a clause packet $q_{\bar{x},2}$ was scheduled at time $t = 3$. This implies that in the satisfying variable assignment x was set false. Then $q_{\bar{x},2}$ reaches v_r at time $t = 4$. From the above reasoning it follows that it cannot collide with any other packet at v_r. Moreover, it reaches $v_{x,6}$ at time $t = 5$. The edge $(v_{x,6}, v_{x,8})$ is blocked during the time interval $[5, 6)$. The packet $\bar{x}_2$ is the only packet that $q_{\bar{x},2}$ competes with for using the edge $(v_{x,6}, v_{x,8})$. But since x was set false in our variable assignment $\bar{x}_2$ started at time $t = 0$ and the above reasoning shows that $\bar{x}_2$ reached $v_{x,8}$ at time $t = 3$. Thus, $q_{\bar{x},2}$ reached $v_{x,8}$ at time $t = 7$. We can do a similar reasoning for clause packets which correspond to positive literals.

This shows that all packets reach their respective destination vertices after at most 7 timesteps. □

Now we want to prove the opposite implication.

Lemma 3.7. *If there is a schedule whose length is at most* 7 *then* ϕ *is satisfiable.*

Proof. Assume that there is a schedule S of length at most 7. We want to show that there is a variable assignment which satisfies ϕ. Let x be a variable. If the packets x_1 and x_2 are both scheduled at time $t = 0$ then we set x to true, if the packet $\bar{x}_1$ and $\bar{x}_2$ are both scheduled at time $t = 0$ we set x to false. We will show in the sequel that this assignment is well-defined and that it satisfies ϕ. First we show that in S either x_1 and x_2 are both scheduled at time $t = 0$ or $\bar{x}_1$ and $\bar{x}_2$ are both scheduled at time $t = 0$. Assume on the contrary that x_1 and $\bar{x}_2$ are scheduled at time $t = 0$. Since the edges $(v_{x,1}, v_r)$ and $(v_{x,2}, v_r)$ are blocked during the time interval $[1, 4)$ and S has length 7 this implies that $\bar{x}_1$ and x_2 are scheduled at time $t = 4$. However, this implies that both packets

arrive at v_r at time $t = 5$. Since both need to use the edge $(v_r, v_{x,3})$ next this contradicts that S has length 7. The case that $\bar{x}_1$ and x_2 are scheduled at time $t = 0$ can be proven similarly. Now we prove that our variable assignment satisfies ϕ. Consider a clause Q with three literals (the case that Q contains only two literals can be proven similarly). Since the edge (v_Q, v_r) is blocked during the time interval $[2, 3)$ there must be at least one packet corresponding to Q which is scheduled at time $t = 3$ or later. Let $q_{x,1}$ be this packet (the cases for the other packets can be proven similarly). We want to show that the packet x_1 is scheduled at time $t = 0$ and thus x is set true in our variable assignment. Assume on the contrary that x_1 is scheduled at time $t = 4$ or later. Then it can arrive at $v_{x,6}$ at time $t = 6$ the earliest. Since $q_{x,1}$ is scheduled at time $t = 3$ it can arrive at the vertex $v_{x,6}$ at time $t = 5$ the earliest. The edge $(v_{x,6}, v_{x,8})$ is blocked during the time interval $[5, 6)$ and thus $q_{x,1}$ cannot have reached $v_{x,8}$ by time $t = 6$. Thus, in order to reach $v_{x,8}$ at time $t = 7$ the packets $q_{x,1}$ and x_1 must use the edge $(v_{x,6}, v_{x,8})$ at time $t = 6$. This is a contradiction. Thus, x_1 is scheduled at time $t = 0$ and we set x true. Doing this reasoning for all clauses Q shows that our variable assignment satisfies ϕ. □

The two previous lemmas for our construction prove Theorem 3.5.

Proof of Theorem 3.5. Follows from Lemmas 3.6 and 3.7 and the construction above. □

Note that the above result implies that it is also NP-hard to approximate the packet routing problem on undirected trees with a performance ratio of $8/7 - \varepsilon$ for all $\varepsilon > 0$.

3.4 Absolute Approximation

The two NP-hardness proofs above use reductions with a gap of one time unit between yes- and no-instances of 3-Bounded-3-Sat. This raises the question whether it is NP-hard to approximate the packet routing problem with a fixed *absolute* error, i. e., with a bound of $OPT + k$ (rather than $k \cdot OPT$) for some fixed value k. We answer this question in the following theorem.

Theorem 3.8. *For any $k \in \mathbb{N}$ it is NP-hard to approximate the packet routing problem with an absolute error of k.*

Proof. We give a reduction from 3-Bounded-3-Sat. Let ϕ be a 3-Bounded-3-Sat formula. We show by induction that for each k there is an integer α_k and an instance I_k of the packet routing problem with the properties that

- $OPT(I_k) = \alpha_k$ if ϕ is satisfiable and
- $OPT(I_k) = \alpha_k + k + 1$ if ϕ is not satisfiable.

In order to clarify matters, we construct I_k such that each packet has its own start and its own destination vertex. For I_0 we take the construction which was used in the proof of Theorem 3.1 and introduce two additional vertices for each packet such that the start and destination vertices of the packets are unique. This increases the makespan by two. We define $\alpha_0 = 8$. Observe that I_0 satisfies the two properties above. Let n_0 be the number of packets that are used in this construction.

For the inductive step we assume that the claim is true for all $i \leq k$. We want to construct a packet routing instance I_{k+1} with the above properties. A sketch of the construction is given in Figure 3.4. Let n_k be the number of packets used in I_k. We denote by α_{k+1} the length of the optimal makespan for I_{k+1} assuming that ϕ is satisfiable. In order to meet the properties stated above we want that an optimal schedule needs at least $\alpha_{k+1} + (k+1) + 1$ steps if ϕ is not satisfiable.

We use n_k copies of I_k and denote them as *formula gadgets*. We denote them by $F_1, ..., F_{n_k}$. The intuition is the following: If ϕ is satisfiable then all packets in all formula gadgets arrive at their destination vertices at time $t = \alpha_k$. If ϕ is not satisfiable then in each gadget there is at least one packet that reaches its destination vertex at time $t = \alpha_k + k + 1$. Thus, there is an absolute difference of $k + 1$ between yes- and no-instances. We want to increase this difference to $k + 2$.

In case that ϕ is not satisfiable we want to control which packet from a formula gadget is late, i. e., reaches its destination vertex at time $t = \alpha_k + k + 1$. In order to achieve this we introduce another type of gadget which we call the *sorting gadget*. We place one sorting gadget behind each formula gadget. A sorting gadget works as follows: Its input consists of all packets from a formula gadget and an additional *sorting packet*. If all packets of a formula gadget leave the formula gadget at time $t = \alpha_k$ then inside the sorting gadget no packets is delayed. If there is a non-empty set of packets P which leave the formula gadget at time $t = \alpha_k + k + 1$ then inside the sorting gadget either one packet in P is delayed once or the sorting packet is delayed once. If a packet in P is delayed inside the sorting gadget then we can already guarantee that the overall makespan has to be at least $\alpha_{k+1} + k + 2$. So for the remainder of the proof we assume that if ϕ is not satisfiable it will be the sorting packet that is delayed inside each sorting gadget.

Finally we place an additional copy $\bar{F}$ of I_k. We will call this the *final gadget*. The input of the final gadget consists of all sorting packets. The intuition is that if ϕ is satisfiable no sorting packet will be delayed inside the sorting gadgets. Moreover, all sorting packets will need only α_k additional steps inside the final gadget.

If ϕ is not satisfiable each sorting packet will be delayed once in the sorting gadget and one of them will need $\alpha_k + k + 1$ steps inside the final gadget. This causes an absolute difference of $k + 2$ between yes- and no-instances.

Now we describe the construction in detail. Let $F_1, ..., F_{n_k}$ be the n_k copies of I_k (the formula gadgets). We know that ϕ is satisfiable if and only if in an optimal schedule all packets leave the formula gadgets after at most α_k timesteps. We call the packets used in formula gadgets the *formula packets.*

For a formula packet m_j denote by s_j its start vertex and by u_j its first vertex behind in the formula gadget (i. e., the first vertex outside the gadget), see Figure 3.4. Behind each formula gadget F_j we place a sorting gadget S_j. The input of the sorting gadget S_j consists of all packets leaving F_j and an additional sorting packet q_j. Figure 3.5 depicts a sketch of a sorting gadget. For a packet p_j denote by w_j the first vertex outside of the sorting gadget. Inside the sorting gadget the formula packets move on horizontal lines consisting of $2n_k$ vertices without interfering with each other. The sorting packet first moves $\alpha_k + k + 1$ edges (this can be understood as an artificial delay). Then it intersects the paths of each formula packet in one edge. After having left the sorting gadget the formula packets move on a path of additional $\alpha_k + k + 2$ edges (there is one path for each packet, so the packets do not interfere with each other). The paths of the sorting packets are then connected to the final gadget $\bar{F}$ which is another copy of I_k. The entire construction is shown in Figure 3.4. We set $\alpha_{k+1} := 2n_k + 2 \cdot \alpha_k + k + 3$.

Now we want to prove that if ϕ is satisfiable then there is a schedule whose length is at most α_{k+1}. We construct a schedule with that length. In this schedule we never delay a packet if not necessary, i. e., if it is the only packet that needs to use its next edge. Since ϕ is satisfiable there is a schedule such that after α_k timesteps all formula packets p_j have reached their respective vertex u_j. Inside the sorting gadgets no packet encounters any delay. Therefore, the sorting packet reaches its vertex w_j after $(\alpha_k + k + 1) + (2n_k + 1) = 2n_k + \alpha_k + k + 2$ timesteps. After another timestep it reaches its last vertex before entering $\bar{F}$. Since ϕ is satisfiable there is a schedule such that after another α_k timesteps, the sorting packet reaches its destination vertex. Thus, after $2n_k+2\alpha_k+k+3 = \alpha_{k+1}$ timesteps all sorting packets have reached their destination vertices. A formula packet p_j reaches w_j after $\alpha_k + 2n_k + 1$ steps. Since after this it has to move another $\alpha_k + k + 2$ edges p_j has reached its destination vertex t_j after at most $2n_k + 2 \cdot \alpha_k + k + 3 = \alpha_{k+1}$ timesteps. Thus, the length of the overall makespan is at most α_{k+1}.

Next we want to show that if ϕ is not satisfiable then the optimal makespan is at least $\alpha_{k+1} + (k+1) + 1 = \alpha_{k+1} + k + 2$. If ϕ is not satisfiable, then in each copy F_j there is at least one formula packet p_j which reaches the vertex u_j after at least $\alpha_k + k + 1$ steps. Now consider the sorting gadget S_j. By construction inside S_j either the sorting packet q_j or p_j is delayed. If p_j is delayed then it needs at least another $(2n_k + 1)+1+\alpha_k+k+2$ steps to reach its destination after having reached u_j. This gives $2\alpha_k + 2n_k + 2k + 5 = \alpha_{k+1} + k + 2$ steps in total. So now assume that in each formula gadget F_j the sorting packet q_j is delayed. This implies that q_j reaches w_j after at least $(\alpha_k + k + 2) + (2n_k + 1) + 1$ steps. Then q_j needs an additional step in order to reach the vertex v_j. This reasoning applies to all sorting packets q_j. Since ϕ is not satisfiable there must be at least one sorting packet which needs another $\alpha_k + k + 1$ steps inside $\bar{F}$. This gives $2\alpha_k + 2n_k + 2k + 5 = \alpha_{k+1} + k + 2$ steps in total. Thus, if ϕ is not satisfiable, the length of an optimal schedule is at least $\alpha_{k+1} + k + 2$. The total number of packets used is $(n_k)^2 + n_k =: n_{k+1}$. The size of this construction is clearly

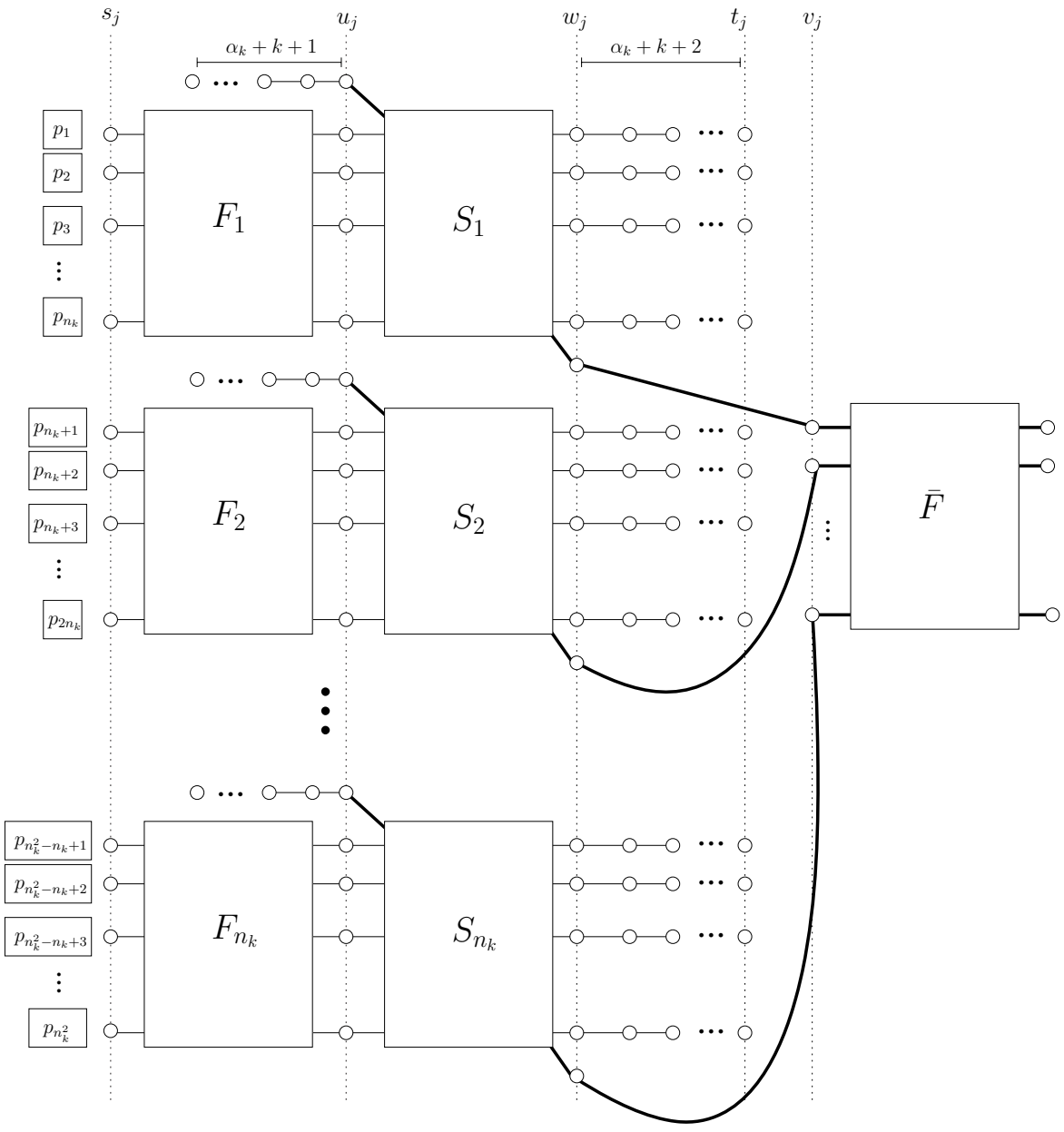


Figure 3.4: Construction for Theorem 3.8.

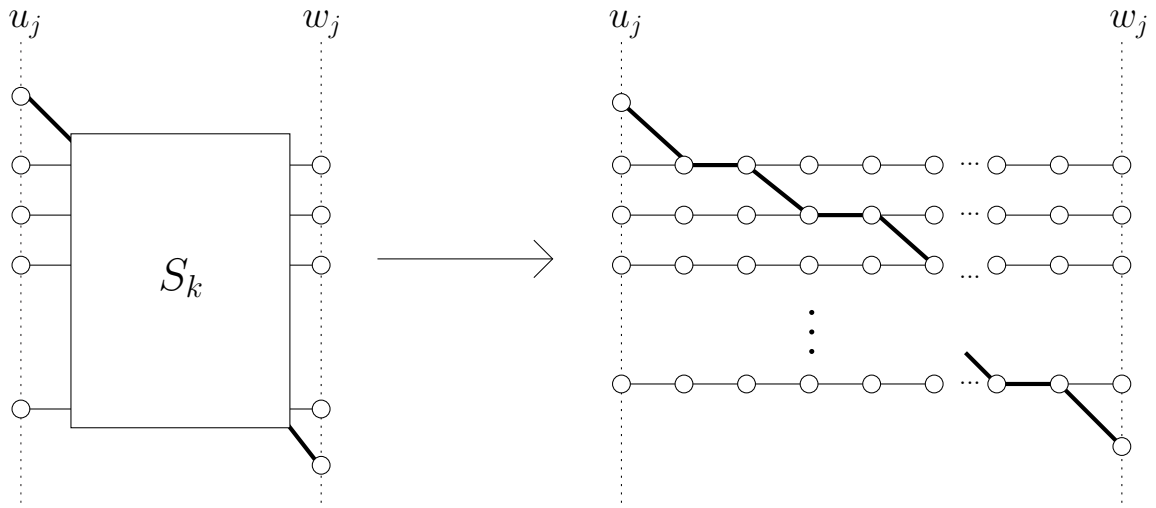


Figure 3.5: The sorting gadget used in the proof of Theorem 3.8.

bounded by a polynomial in the size of I_k. By induction, it is also bounded by a polynomial in the size of ϕ.

This proves that it is NP-hard to approximate the packet routing problem with an absolute error of k for any $k \in \mathbb{N}$. □

3.5 Conclusion

In this chapter we proved that the packet routing problem is APX-hard and hence there can be no PTAS for it, unless $P = NP$. This holds even for directed trees which have a relatively simple structure. On the other hand, the best known approximation algorithms for packet routing on directed and undirected trees achieve approximation factors of 2, see Chapter 1. This leaves a quite large gap to the approximation factor of $(8/7 - \varepsilon)$ which we can rule out (for any $\varepsilon > 0$). On general graphs, we showed non-approximability for factor $(6/5 - \varepsilon)$ for any $\varepsilon > 0$. There, the best known approximation factors are $O(1)$ [66] and – for sufficiently large $C + D$ – our algorithm presented in Chapter 2 which computes a schedule of length $23.4(C + D)$. It remains open to fill this gap.

All our reductions create a gap of one timestep between yes- and no-instances of the reduced 3-Bounded-3-Sat problem. The approximation factors in our theorems are given by the ratios between the two makespans of these types of instances (and substracting ε). Since packet routing instances with $D = 1$ are trivial, our strategy cannot rule out approximation factors which are larger than 3/2. Even showing NP-hardness for a factor of $3/2 - \varepsilon$ seems difficult to achieve this way. Hence, we believe that in order to prove better hardness results, more sophisticated techniques like the PCP-Theorem [6, 7] or the Unique Games Conjecture [59] are necessary.

Chapter 4

Periodic Packet Routing

4.1 Introduction

In the previous chapters, we studied the packet routing problem in the static setting: A finite set of packets is given and the packets have to be scheduled to reach their respective destination as fast as possible. When data are transmitted through a network, sometimes indeed only a very limited amount of information has to be transported, e. g., consider the download of a website. However, in applications like Voice over IP (VoIP) or live video-streaming new information is created continuously that has to be transported. In particular, in these settings a lot of packets have the same origin and destination. Also, the information that has to be transported is not known entirely at the very beginning but becomes available over time. In order to capture this behavior, it is not so suitable to think of single packets that need to be transported. It is much more appropriate to think of *tasks* that continuously create new packets that all follow the same source-destination-path. This leads to the *periodic packet routing problem*.

We assume that each task continuously creates new packets over an infinite time horizon, modeling for example a video-stream or a VoIP-connection. In a video-transmission packets are created in a steady continuous stream. Hence, a good model is that each tasks creates new packets at regular intervals of, e. g., p timesteps. We call this the *strict periodic setting*. However, in a VoIP-connection packets are created only when a user actually speaks. Therefore, it is not clear when exactly new packets are emitted. It is only known that the rate is bounded in which the packets are created. We model this in the *sporadic setting* where we demand only that each task emits at most one packet in each time interval of length p.

In periodic packet routing, we deal with an infinite time horizon. Also, in the sporadic setting it is not clear a priori when exactly each task emits its packets. Hence, one cannot describe the routing schedule explicitly. Instead, we define a *scheduling policy* which defines when the packets are transported. Such a policy should ensure a certain quality of service (QoS). This means that each

packet is delayed along its path by at most a certain, tolerable small time span. Our objective is to design a scheduling policy which ensures a good QoS. We will make this precise in the sequel. Moreover, since huge quantities of packets have to be transported through the network, policies with a small computational overhead for the routing decisions are desired.

In this chapter, we compare two paradigms for scheduling policies, priority schedules and template schedules. In *priority schedules* we transfer the concept of fixed-priority schedules in real-time scheduling to periodic packet routing. Each edge maintains an ordered priority list of the tasks using it. Whenever two packets are in conflict, a packet has priority which was created by a task with higher priority. In a *template schedule* at each point in time each edge is open to transfer packets of exactly one task. This exclusive openness permutes cyclically in time over all tasks that send their packets along the edge. In both paradigms it is very simple to compute what packet has priority in case of a conflict. Nevertheless, from an implementation point of view, priority schedules are much easier to handle since template schedules require a global clock (to synchronize the mentioned cyclic permutation on the edges).

In the sequel we evaluate the two paradigms. We design priority and template schedules and bound the maximum delay of the packets in the respective schedules. Also, we prove lower bounds for the potential of priority and template schedules for different graph classes. As a general statement, we can say that template schedules are more powerful than priority schedules. We make this precise in the rest of this chapter.

The results presented in this chapter are joint work with Britta Peis and Sebastian Stiller [86]. Before we give a detailed overview of our results, we summarize some definitions.

4.1.1 Definitions

Let $G = (V, A)$ be a directed tree, i. e., a directed graph such that the underlying undirected graph is a tree. Let T denote a set of tasks $\tau_i = (s_i, t_i)$ with $s_i, t_i \in V$ such that in G there is a directed path from s_i to t_i. Let $p \in \mathbb{N}$ denote the global period length. We call $I = (G, T, p)$ an instance of the *periodic packet routing problem* (PPRP). We assume a discrete time model. Each task τ_i repeatedly creates new packets which have to be transported from s_i to t_i by a routing schedule. We assume unit transit times (i. e., each packet needs one timestep to traverse an arc), unlimited storage in each vertex, and unit bandwidths (i. e., each arc can be used by at most one packet at a time).

We also consider undirected trees, where the paths of different tasks may include the same edge in opposite directions. Still, for *undirected* trees, the edge can only be used by one packet at time. In case an edge can be used by one packet in each direction at a time, we speak of a *bidirected* tree. Then, we interpret an edge as two (directed) antiparallel arcs.

We distinguish between the *strict* and the *sporadic PPRP*. For instances of the strict PPRP each task generates a new packet every p timesteps, starting at time $t = 0$. In instances of the sporadic PPRP it is not known a priori when the

tasks emit their packets. We require only that in each time interval of length p each task emits at most one packet. We call a specification of the release times for the packets a *realization*.

For each task τ_i we denote by $P_i \subseteq E$ the arcs on the unique path from s_i to t_i and define $D_i := |P_i|$. For an arc e let T_e denote the set of tasks which use e. Whenever we consider general graphs, we assume that we are given the paths P_i of the tasks explicitly.

Given an instance and a realization for it, a *schedule* S must (for an infinite time horizon) assign to each edge for each point in time which packet traverses it. We define a *limit* for a task τ_i in a schedule S to be a value k such that each packet which is ever created by τ_i needs at most k timesteps to reach t_i after it has been created. Denote by c the congestion, that is the maximum number of tasks which use a directed edge – or the maximum number of tasks which use an undirected edge in one direction. (We use a lower-case c here to distinguish it from the congestion in the static setting.) We say an instance I is *feasible* if there is a schedule for I such that for each task there is a finite limit. A schedule is a *direct schedule* if no packet waits in a vertex different from its start vertex (and its destination vertex). A schedule is called *indirect* if it is not necessarily direct.

Now we define the two main classes of scheduling paradigms under consideration, template schedules and priority schedules.

Template Schedules

One of the types of schedules studied in this chapter are template schedules [5]. Template schedules can be understood as a set of pin-wheels. For each edge there is one pin-wheel with $\bar{p}$ slices (for some integer $\bar{p}$). Each slice of the pin-wheel belongs to one task which uses this edge. Each pin-wheel has a pointer. At each timestep t the pointer points at a slice of the wheel. Only the packets of the task of this slice can use the edge at time t. After every timestep, each pin-wheel is turned such that the pointer points at the next slice.

Now we give a formal definition.

Definition 4.1 (Template schedules)**.** Let $I = (G, T, p)$ be an instance of the sporadic PPRP or the strict PPRP. A schedule for I is a *template* schedule if there exists an integer $\bar{p} \leq p$ and a map $\text{task} : E \times \{0, ..., \bar{p} - 1\} \to T \cup \{\text{none}\}$ such that in each realization each arc $e = (u, v)$ is used at time t by a packet M created by a task τ if and only if

- $\text{task}(e, t \bmod \bar{p}) = \tau$,
- M is located on u at time t, and
- no packet is located on u which was created by τ earlier than M.

We call the value $\bar{p}$ the *periodicity* of the map task. Note that each map $\text{task} : E \times \{0, ..., \bar{p} - 1\} \to T \cup \{\text{none}\}$ yields a template schedule, if its restriction to every arc e is surjective on T_e. Also note that a template schedule might delay

a packet even though there is no conflicting packet waiting. Apart from template schedules, this peculiarity is inevitable in a direct periodic schedule.

Global-priority and Edge-priority Schedules

The other types of schedules studied in this chapter are global-priority and edge-priority schedules. These schedules work with priority hierarchies for the tasks. If there are two packets which both need to use some edge we give priority to the packet which was created by a task with higher priority. Global-priority schedules have a global prioritization of the tasks (which is valid for each edge), edge-priority schedules might have different prioritizations for each edge.

Definition 4.2 (Edge-priority schedule)**.** Let $I = (G, T, p)$ be an instance of the sporadic or strict PPRP. A schedule for I is an *edge-priority* schedule if there exists a total order $\prec_e \subseteq T \times T$ for each edge e in G such that the following holds for every realization: Whenever there are several packets waiting to use e, a packet moves first such that for its corresponding task τ we have that $\tau \prec_e \tau'$ for each other task τ' that created a packet waiting to use e.

Note that the definition does not explicitly define what happens if the task τ with highest priority has created several packets waiting to use some edge. However, it makes sense for a schedule to give priority to the packet which was created first. A global-priority schedule is simply an edge-priority schedule in which all edges e have the same ordering $\prec_e$.

Definition 4.3 (Global-priority schedule)**.** Let $I = (G, T, p)$ be an instance of the sporadic or strict PPRP. A schedule for I is a *global-priority* schedule if it is an edge-priority schedule such that all relations $\prec_e$ for the edges are identical.

Both types of priority schedules are simpler to implement than template schedules: While priority schedules can be executed fully locally, template schedules require a "global clock". Edge-priority schedules are a strengthened version of global-priority schedules.

In this chapter we show that the more complicated template schedules are indeed significantly more powerful. Moreover, we give algorithms to efficiently construct such superior schedules.

4.1.2 Related Work

For related results for the non-periodic packet routing problem we refer to Chapter 1. Recall that for the non-periodic setting Leighton et al. [65] show that there is always a schedule of length $O(C + D)$ (where C denotes the maximum number of packets using an edge and D is the length of the longest path of a packet). This result is extended to the periodic setting by Andrews et al. [5] guaranteeing a bound of $O(D_i + 1/r_i)$ for each session i with a packet injection rate of r_i (corresponding to a task τ_i with a period length $p_i = 1/r_i$ in our notation). In particular, they introduce the template schedules which are also studied in this chapter.

Template Schedules				
	Indirect Schedules		Direct Schedules	
	Limit	Bound on c	Limit	Bound on c
Dir. trees	$c + D_i - 1$	p	$c + D_i - 1$	p
Bidir. trees	$2c - c\eta + D_i - 1$	p	$2c + D_i - 2$	$p/2$
Undir. trees	$4c - 2c\eta + D_i - 1$	$p/2$	$4c + D_i - 3$	$p/4$

Table 4.1: Overview of our results for template schedules with $\eta := (1/2)^{\lceil \mathrm{diam}(G)/2 \rceil - 1}$.

Priority Schedules				
	Type	Limit	Bound on c	Lower Bound
Dir. trees	Edge	$1.5c + D_i - 1$	$2p/5$	$1.25c + D_i - 1$
Dir. trees	Global	–		$2c + D_i - 1$
Bidir. trees	Global	$2c + D_i - 1$	$p/3$	–

Table 4.2: Overview of our results for priority schedules. All bounds hold for both the strict periodic and the sporadic setting. The entries marked with a "–" follow directly from the result for directed trees or bidirected trees, respectively.

Our task models are borrowed from classical real-time scheduling. Here, one studies real-time executable algorithms for distributing and scheduling (computational) jobs on a processor platform. However, as there is no graph involved the techniques are quite different. For an overview see [18]. Note that with arbitrary period lengths for the tasks (rather than all being identical to p) the PPRP with only one edge is identical to real-time scheduling on one machine.

4.1.3 Outline of the Chapter

We give a comprehensive characterization of the periodic packet routing problem in the various settings (direct/indirect schedules, bidirected/directed/undirected trees, template/priority schedules), Tables 4.1 and 4.2 give a complete overview. We present algorithms and prove limitations and relations of the different types of schedules. In particular, this compares the power of the two scheduling paradigms. It turns out that template schedules are strictly more powerful than priority schedules.

Most importantly, we show the following results:

- In the following Section 4.2 we prove a necessary and sufficient condition for the existence of a schedule. The affirmative part of the theorem rests on a structural insight for template schedules that allows to bound the backlog of the packets.
- In Section 4.3 we prove that template schedules can guarantee a limit of $c + D_i - 1$ on directed tree and a limit of $2c - c(1/2)^{\lceil \mathrm{diam}(G)/2 \rceil - 1} + D_i - 1$ on bidirected trees (for each task τ_i, respectively), where $\mathrm{diam}(G)$ denotes

the diameter of G. The latter is achieved by carefully distributing the necessary delays among the tasks.

- Then we study priority schedules in Section 4.4. For directed trees we give an edge-priority schedule that guarantees a maximal delay of $1.5c-1$ for each packet. We give a non-trivial construction that yields a lower bound which almost matches the quality achieved by the algorithm. For bidirected trees we give a global-priority schedule which assigns a maximal delay of $2c-1$ to each packet. A lower bound shows that this is best possible.

The above results already prove that template schedules are strictly more powerful than priority schedules.

- Then, in Section 4.5 we follow a more direct approach to compare the power of template schedules and priority schedules. We show that whenever a priority schedule achieves a certain quality of service, one can construct a template schedule imitating the priority schedule well enough to achieve the same or almost the same quality. Key to these results is to prove that after some time priority schedules show a periodic behavior.

We supplement our results for template schedules by giving algorithms for designing direct template schedules. Note that priority schedules cannot yield direct schedules due to their specific nature.

All our algorithms are designed for trees in the setting of identical period lengths p. This is so because we show that already on chain graphs (which have a quite simple structure) no edge-priority schedule can always guarantee non-trivial limits. Also, we prove that with arbitrary period lengths already on a path no edge-priority schedule can guarantee a delay for each packet which is bounded by a constant times the period length of its respective task. See Section 4.4 for the corresponding proofs.

Finally, in Section 4.6 we conclude and address open problems.

4.1.4 Comparison of Template- and Priority-Schedules

Our work in this chapter is dedicated to comparing the power of template and priority schedules. Our results show that on bidirected trees, template schedules are strictly better than global-priority schedules. We have a global-priority schedule which guarantees a bound of $2c+D_i-1$ and a matching lower bound. However, we present template schedules which guarantee the bound $2c-c\eta+D_i-1$ (with $\eta=(1/2)^{\lceil \mathrm{diam}(G)/2\rceil-1}$) which is strictly better than $2c+D_i-1$. For directed trees, we show that template schedules are more powerful than even edge-priority schedules. There are template schedules guaranteeing a bound of $c+D_i-1$ which is best possible. However, no edge-priority schedule can give a better bound than $1.25c+D_i-1$ for every task (and we have an algorithm which guarantees a bound of $1.5c+D_i-1$).

Note that on general graphs Andrews et al. [5] prove the existence of template schedules which guarantee a bound of $O(c+D_i)$ (in our notation) for each

task τ_i. (However, in contrast to our template model, they allow a larger periodicity than p.) In fact, they show this result even for arbitrary period lengths. However, for general graphs we prove in Section 4.4 that no edge-priority schedule can guarantee a better limit than $\Omega(c \cdot D)$ for every task. In the same section we show that with arbitrary period lengths p_i even on a path no edge-priority schedule can guarantee a better limit than $\alpha \cdot p_i + D_i$ for any $\alpha \geq 0$. This proves that on general graphs template schedules are clearly superior to priority schedules.

A question which we leave open is whether on bidirected trees edge-priority schedules can be as good as template schedules. In particular, for the latter is it not clear to us whether the bound of $2c - c(1/2)^{\eta-1} + D_i - 1$ is already best possible or whether further improvements are possible.

4.2 Necessary Bound on Congestion

In this section we prove that an instance of the periodic packet routing problem has a schedule with a finite limit for each task if and only if $c \leq p$. Hence, for the instances studied in the remainder of this chapter we will always require that $c \leq p$. Note that this result does not only hold for instances on trees but also for instances on general graphs. (Here we need to assume that the paths of the tasks are given as part of the input since otherwise c would not be defined.)

Theorem 4.4. *An instance I of the strict or sporadic periodic packet routing problem is feasible if and only if $c \leq p$.*

Before we can prove the theorem we need some preparation. First, we prove a useful insight for template schedules. It shows that in a (reasonable) template schedule no vertex can hold more than one packet created by each task τ (apart from the destination vertex of τ).

Lemma 4.5. *Let I be an instance of the sporadic PPRP and let S be a template schedule for I with periodicity $\bar{p}$. Assume that for each arc e we have $\{\text{task}(e,k) | 0 \leq k < \bar{p}\} = T_e$. If a packet M created by a task τ_i arrives on a vertex v at time t then no packet created by τ_i arrives on v during the time interval $[t+1, t+\bar{p})$.*

Proof. Let τ_i be a task. We prove the claim by induction over P_i. Since $\bar{p} \leq p$ the claim holds for s_i (due to our definition of the sporadic setting). Now let $\{s_i = v_0, v_1, ...v_{k-1}, v_k = t_i\}$ be the vertices on P_i. We assume by induction that the claim holds for the vertices $v_1, ..., v_\ell$. In particular, this implies that at any point in time t there is at most one packet created by τ_i on v_ℓ. Assume that at time t a packet M created by τ_i arrives on v_ℓ. Since S is a template schedule with periodicity $\bar{p}$ there must be a timestep $t' \geq t$ with $t' < t + \bar{p}$ at which M traverses the arc $e_\ell = (v_\ell, v_{\ell+1})$. In particular, this implies that during the time interval $[t + \bar{p}, t' + \bar{p})$ no packet created by τ_i traverses e_ℓ (due to the periodicity of S). We conclude that M arrives on $v_{\ell+1}$ at time $t' + 1$ and no packet created by τ_i arrives at $v_{\ell+1}$ during the time interval $[t' + 2, t' + \bar{p} + 1)$. □

Using the above insight, in the following lemma we prove a universal limit for template schedules.

Lemma 4.6. *Let S be a template schedule, given by a map* task *with periodicity $\bar{p}$, for a strict or sporadic periodic packet routing instance I. Assume that for each arc e we have $\{\text{task}(e,k)|0 \leq k < \bar{p}\} = T_e$. Then, for each task τ_i it holds that $D_i \cdot \bar{p}$ is a valid limit.*

Proof. From Lemma 4.5 it follows that there can be no two packets created by the same task τ_i waiting for an arc $e \in P_i$. From the definition of template schedules it follows that a packet M has to wait at most $\bar{p} - 1$ timesteps before it can be transferred over the next arc on its path. Thus, each packet created by τ_i needs at most $D_i \cdot \bar{p}$ steps to reach t_i. □

Now we prove that $c \leq p$ is a necessary and sufficient condition for the existence of a periodic routing schedule.

Proof of Theorem 4.4. First assume on the contrary that I is feasible but there is an arc $e = (u, v)$ which is used by more than p tasks. Since I is feasible there is a limit k_i for each task τ_i. We define $k := 1 + \max_i k_i$.

Assume that for each $k' \in \mathbb{N}_0$ each task τ_i creates a new packet at the timestep $t = k' \cdot p$. Hence, during the time interval $L = [0, p)$ each task $\tau_i \in T_e$ has created one packet. This implies that in L all tasks in T_e together have created $c > p$ packets. Since c and p are integral, this implies that $c \geq p + 1$. Since at most p packets can be transferred over e in p timesteps, there is at least one packet which has been created within L which has not been transferred over e. Within the time interval $L_k = [0, k \cdot p)$ each task $\tau_i \in T_e$ has created k packets. This implies that within L_k all tasks in T_e together have created $k \cdot c \geq k \cdot (p + 1)$ packets. However, at most $k \cdot p$ packets could possibly have used the arc e. That means that there are at least k packets in the network which still need to use e. Thus, there is one of those packets $M_{i,j}$ created by a task τ_i which does not reach v before time $t = k \cdot p + k$. Let $c_{i,j}$ be the time when $M_{i,j}$ was created. Since $c_{i,j} \leq k \cdot p$ we conclude that $k \cdot p + k \geq c_{i,j} + k > c_{i,j} + k_i$ and thus $M_{i,j}$ reaches t_i after strictly more than k_i timesteps. Thus, k_i is not a valid limit for τ_i which is a contradiction.

Now assume that $c \leq p$. We create a template schedule as follows: First, we define $\bar{p} := p$. Now let e be an arc. W. l. o. g. assume that $T_e = \{\tau_0, \tau_1, ..., \tau_{|T_e|-1}\}$. Note that $|T_e| \leq c \leq p$. We define $\text{task}(e, k) := \tau_k$ for $0 \leq k < |T_e|$ and $\text{task}(e, k') :=$ none for $|T_e| \leq k' < \bar{p} = p$. We do this procedure for each arc e. Then, for each arc e we have $\{\text{task}(e,k)|0 \leq k < \bar{p}\} = T_e$ since $|T_e| \leq \bar{p} = p$. Denote by S the (cheap) template schedule resulting from task. From Lemma 4.6 it follows that in S each task has a finite limit. Thus, I is feasible. □

4.3 Template Schedules

The main aim of this chapter is to compare the power of template- and priority schedules. In order to do so, in this section we study template schedules on

directed, bidirected, and undirected trees. Our schedules guarantee a limit for each task and hence a bound on the maximal delay for each task. Moreover, we distinguish between direct and not necessarily direct (indirect) schedules. For each algorithm computing a schedule we prove the limit that it guarantees for each task. Note that for some algorithms we require an upper bound on the congestion c which is stricter than the necessary condition $c \leq p$. This raises the question whether one can still obtain schedules with similar limits if c is larger (though still bounded by p). We answer this question by giving examples for the respective settings with higher congestion where no template schedule can achieve the limits guaranteed by our algorithms. For the case of direct schedules we even give counterexamples where there can be no direct template schedule at all.

The limits guaranteed by our algorithms are summarized in Table 4.1.

4.3.1 Directed Trees

In the sequel, we will study template schedules on directed, bidirected, and undirected trees. We start with directed trees and present a general technique which we will adapt later for the other two tree classes. Note that the case of directed trees is a special case of bidirected trees.

Let $I = (G, T, p)$ be an instance of the sporadic periodic packet routing instance on a directed tree G. We present an algorithm which constructs a template schedule which guarantees a limit of $c + D_i - 1$ for each task τ_i. This bound is best possible: There are instances where there can be no better limit for every task (e. g., consider an instance on a directed path in which all tasks have identical paths). Also, we will show below that it is NP-hard to determine whether there is a template schedule which guarantees a limit of $c + D_i - 2$ for an instance on a directed tree. Our algorithm transfers ideas from Chapter 1 to the periodic setting.

Algorithm $DTREE(I)$

1. Find feasible path-coloring $f : T \to \{0, ..., c-1\}$ for paths of tasks.
2. Define time-dependent edge-coloring $g : E \times \{0, ..., c-1\} \to \{0, ..., c-1\}$ with consecutive property (the latter ensures limit and direct routing).
3. Define map task from path-coloring and time-dependent edge-coloring.

Now we present the algorithm. We compute a *feasible path-coloring* for the paths of the tasks, i. e., we compute a map $f : T \to \{0, ..., c-1\}$ such that if the paths of two tasks τ_i, τ_j share an arc then $f(\tau_i) \neq f(\tau_j)$. Such a coloring exists since we assumed that at most c paths use each edge (see Lemma 1.3). The coloring can be obtained by first solving the problem for the subgraphs induced by each vertex together with its neighbors. This can be reduced to edge-coloring on bipartite multigraphs (bipartite graphs with possibly parallel edges). Then,

the solutions for the subproblems can be combined to a global coloring. The whole procedure can be accomplished in polynomial time, for details we refer to Section 1.3.1.

Similarly as in Section 1.3.2 we compute a time-dependent edge-coloring $g : E \times \{0, ..., c-1\} \to \{0, ..., c-1\}$ as follows: We start with an arbitrary arc e^* and define its coloring by $g(e^*, i) := i$ for $0 \leq i < c$. Then, we define the coloring of the remaining arcs such that the *consecutive property* holds:

- For two consecutive arcs $e = (u, v)$ and $e' = (v, w)$ we require that $g(e, i) = g(e', (i+1) \bmod c)$ for $0 \leq i < c$.
- For two adjacent arcs $e = (u, v)$ and $e' = (u, v')$ (or $e = (u, v)$ and $e' = (u', v)$) we require that $g(e, i) = g(e', i)$ for $0 \leq i < c$.

Note that after having defined the coloring of e^* the consecutive property implies the coloring of the other arcs. From the colorings f and g we compute the map task as follows. We define $\bar{p} := c$. Let e be an arc and let $k \in \{0, ..., c-1\}$. If there is a task $\tau_i \in T_e$ such that $f(\tau_i) = g(e, k)$ we define task$(e, k) := \tau_i$. Since f is a valid path coloring there can be at most one such task. If $g(e, k) \notin f(T_e)$ we define task(e, k) := none. We do this for all arcs e and all timesteps t. Denote by $DTREE(I)$ the resulting schedule.

Theorem 4.7. *Let I be an instance of the sporadic periodic packet routing problem on a directed tree with $c \leq p$. The schedule $DTREE(I)$ is a direct template schedule which guarantees a limit of $c + D_i - 1$ for each tasks τ_i.*

Proof. The (first) consecutive property of g is passed on to the map task. For two consecutive arcs $e = (u, v)$ and $e' = (v, w)$ we have that task(e, k) = task$(e', k + 1 \bmod c)$ for all k. Therefore, once a packet has left its start vertex it is never delayed until it reaches its destination vertex. Each packet has to wait for at most $c - 1$ timesteps in its start vertex. We conclude that for each task τ_i it holds that $c + D_i - 1$ is a valid limit. □

It is easy to see that there are instances of the strict PPRP where no schedule can guarantee a better limit than $c + D_i - 1$ for each task τ_i. For example, consider a path P of length D and c tasks τ_i such that $P_i = P$ for each task τ_i. However, there are many instances of the strict PPRP which allow a schedule with a better limit than $c + D_i - 1$ for each task τ_i. Nevertheless, it is not immediately clear how to determine for an instance I of the strict PPRP whether there is a schedule that guarantees a better limit than $c + D_i - 1$ for each task τ_i. Now we show that this is in fact NP-hard.

Theorem 4.8. *For instances of the strict PPRP on directed trees it is NP-hard to compute whether there is a template schedule which guarantees a limit of $c + D_i - 2$ for each task τ_i.*

We use a similar construction as used in Chapter 3 for showing that the non-periodic packet routing problem is NP-hard to approximate on directed trees. For the sake of completeness, we describe the reduction in detail.

Task	Path	
Variable tasks for variable x		
$\tau_{x,1}$	$v_{x,1}, v_r, v_{x,3}, v_{x,4}$	For each variable x
$\tau_{x,2}$	$v_{x,2}, v_r, v_{x,6}, v_{x,8}$	For each variable x
$\tau_{\bar{x},1}$	$v_{x,1}, v_r, v_{x,6}, v_{x,7}$	For each variable x
$\tau_{\bar{x},2}$	$v_{x,2}, v_r, v_{x,3}, v_{x,5}$	For each variable x
Clause tasks for clause Q with three literals		
$\tau_{q,x,1}$	$v_Q, v_r, v_{x,3}, v_{x,4}$	where Q contains the first pos. occurrence of x
$\tau_{q,x,2}$	$v_Q, v_r, v_{x,6}, v_{x,8}$	where Q contains the second pos. occurrence of x
$\tau_{q,\bar{x},1}$	$v_Q, v_r, v_{x,6}, v_{x,7}$	where Q contains the first neg. occurrence of x
$\tau_{q,\bar{x},2}$	$v_Q, v_r, v_{x,3}, v_{x,5}$	where Q contains the second neg. occurrence of x

Table 4.3: The predefined paths of the tasks. Clause tasks of clauses Q with only two literals visit one additional vertex v'_Q between v_Q and v_r.

We reduce from 3-Bounded-3-Sat. Assume we are given a 3-Bounded-3-Sat formula ϕ. We construct an instance I of the periodic packet routing problem such that the underlying graph is a directed tree. For I we define $p := 6$. The following definitions will impliy that also $c = 6$. We show that ϕ is satisfiable if and only if there is a schedule for I which guarantees a limit of $c + D_i - 2$ for each task τ_i. We call such a schedule a *short* schedule. We call a schedule *long* if it is not short.

We assume that in ϕ each clause contains at least two variables and that each variable occurs at most two times positive and at most two times negated. For each variable x in the formula, there are four tasks $\tau_{x,1}$, $\tau_{x,2}$, $\tau_{\bar{x},1}$, and $\tau_{\bar{x},2}$. The instance is defined such that in a short schedule S they can be scheduled at times $t \equiv 0 \bmod p$ or $t \equiv 4 \bmod p$. The intuition is that if x_1 and x_2 are scheduled at time $t \equiv 0 \bmod p$ then we set the variable x true, if x_1 and x_2 are scheduled at time $t \equiv 4 \bmod p$ then we set the variable x false.

Note that possibly none of the above cases apply, i. e., the tasks $\tau_{x,1}$ and $\tau_{\bar{x},2}$ are scheduled at time $t \equiv 0 \bmod p$ or the tasks $\tau_{x,2}$ and $\tau_{\bar{x},1}$ are scheduled at time $t \equiv 0 \bmod p$. However, our construction ensures that then the resulting schedule is long.

For each literal in a clause Q we introduce a task. Tasks corresponding to literals in the same clause share the first arc on their path. Given a satisfying variable assignment for ϕ there is at least one literal in Q which satisfies the clause. The intuition is that tasks which corresponds to these literals are scheduled last (at times $t \equiv 3 \bmod p$).

Now we describe the construction in detail. For each variable x we introduce vertices $v_{x,1}, v_{x,2}, ..., v_{x,8}$. We also introduce the four *variable tasks* $\tau_{x,1}$, $\tau_{x,2}$, $\tau_{\bar{x},1}$, and $\tau_{\bar{x},2}$. In the formula, we fix an order of the clauses. For each clause Q we introduce a vertex v_Q. For each literal in Q we introduce a *clause task*. We write $\tau_{q,x,i}$ for the task corresponding to the i-th positive occurrence of x and $\tau_{q,\bar{x},i}$ for the task corresponding to the i-th negative occurrence of x. The

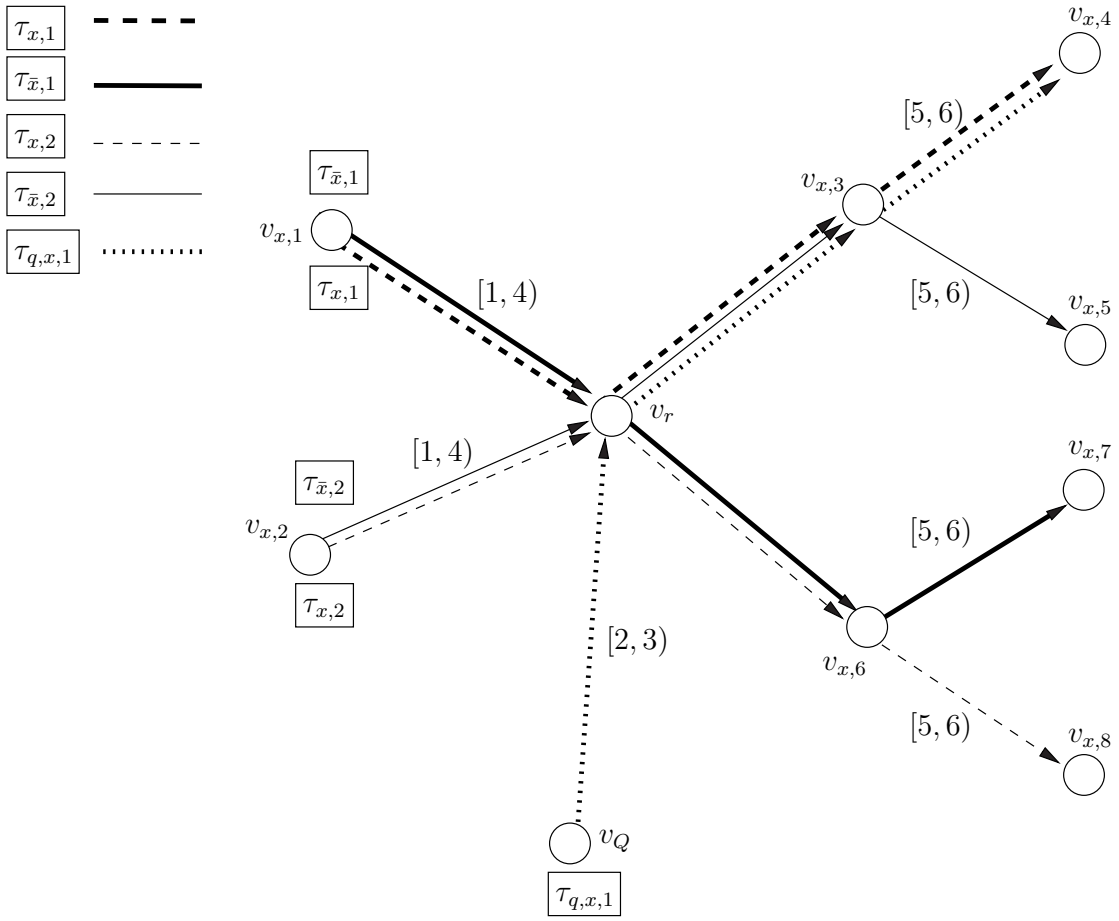


Figure 4.1: A part of the graph for the clause $Q = (x \vee y \vee z)$ (construction for Theorem 4.8). The parts corresponding to the variables y and z and their packets are omitted for brevity. The time intervals at the arcs denote the time when the respective arc is blocked. Note that the first timestep is $t = 0$.

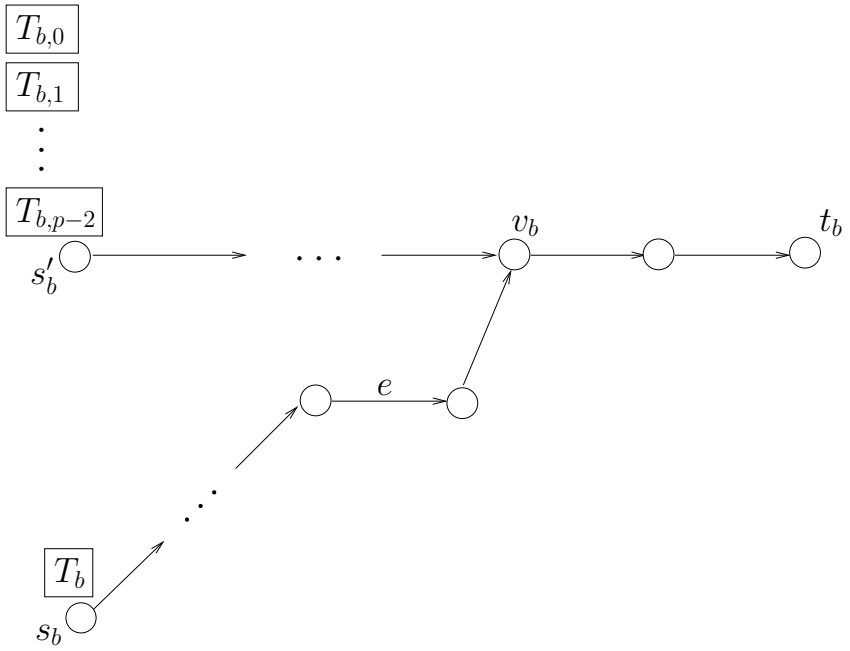


Figure 4.2: The paths of the tasks $\tau_b, \tau_{b,0}, \tau_{b,1}, ..., \tau_{b,p-2}$ are defined such that if τ_b is never delayed before having reached v_b then it will not collide with any delay task $\tau_{b,i}$. However, if τ_b is delayed once, then it will arrive at v_b at the same time as the fastest delay task. The latter implies that in the overall instance there must be task which is delayed $p-1$ times in total.

predefined paths for the task are shown in Table 4.3 and sketched in Figure 4.1. Note that the underlying graph is a directed tree.

In order to make the construction work as desired we want to block some edges at certain times. We do this by introducing *blocking tasks* (see a sketch in Figure 4.2). The blocking tasks take the role of the blocking packets in the proof of Theorem 3.5. However, this part of the construction is more complicated than in the non-periodic setting.

A blocking task τ_b is constructed such that it has to use an edge e at times t with $t \equiv \bar{t} \bmod p$, otherwise there would be a task in I with a total delay of at least 5 (as we will ensure below). The path of τ_b is defined such that if τ_b is not delayed it uses the edge e at times t with $t \equiv \bar{t} \bmod p$. After having passed e the path of τ_b moves on to an edge where it crosses the path of $p-1$ other tasks $T_b := \{\tau_{b,0}, ..., \tau_{b,p-2}\}$. We call the latter the *delay tasks of* τ_b. We define that all delay tasks of τ_b use the same path. There are $p-2$ delay tasks for τ_b. Hence, in order to ensure that the resulting schedule is short it is necessary that for each $k \in \{0, 1, ..., p-2\}$ there must be a task $\tau_k \in T_b$ which has an initial delay of k. The paths of τ_b and its delay tasks are designed such that τ_b does not interfere with its delay tasks if τ_b is not delayed at all. However, if τ_b is delayed once then it arrives at v_b at the same time as the task of T_b with zero initial delay. This implies that one of the tasks $\{\tau_b\} \cup T_b$ has a total delay of $p-1$. In order to clarify matters we do not explicitly define the paths of the blocking packets. Instead, we state which edges are blocked at what time. To simplify

notation we write $[i, i+j)$ if we want to state that an arc is blocked during the time intervals $[i+k\cdot p, i+j+k\cdot p)$ for all $k \in \mathbb{N}$.

Let x be a variable. The arcs $(v_{x,1}, v_r)$ and $(v_{x,2}, v_r)$ are blocked during the time interval $[1, 4)$ (note that $t = 0$ is the first timestep). The arcs $(v_{x,3}, v_{x,4})$, $(v_{x,3}, v_{x,5})$, $(v_{x,6}, v_{x,7})$, and $(v_{x,6}, v_{x,8})$ are blocked during the time interval $[5, 6)$. For each clause Q with three literals the arc (v_Q, v_r) is blocked during the time interval $[2, 3)$. For each clause Q' with two literals the arc $(v_{Q'}, v_r)$ is blocked during the time interval $[1, 3)$. Let M be a packet which was created by a variable or a clause task. Due to the above reasoning we assume that if M wants to use an arc e at a time when e is blocked then M is delayed.

Proof of Theorem 4.8. Now we show that ϕ is satisfiable if and only if there is a short periodic schedule. First we assume that ϕ is satisfiable. In the proof of Theorem 3.5 we showed – for the same construction in the non-periodic case – that ϕ is satisfiable if and only if the optimal makespan is 7. Denote by $\mathcal{M}_i$ all packets which were created at time $t = i \cdot p$. We can turn a non-periodic schedule S' of length 7 into a periodic schedule in which all packets in $\mathcal{M}_i$ arrive at their respective destination vertices at time $t = i \cdot p + 7$ the latest. We do this by infinitely repeating the packet movements defined in S'. It remains to argue why packets in $\mathcal{M}_i$ do not interfere with packets in $\mathcal{M}_{i-1}$. At time $t = i \cdot p$ the packets of $\mathcal{M}_{i-1}$ which have not reached their destination vertices yet are located on vertices $v_{x,3}$ and $v_{x,6}$ for respective variables x. Thus, they do not interfere with the newly created packets in $\mathcal{M}_i$. Since the paths of all tasks have length 3 we conclude that there is a (short) schedule with a limit of $7 = 3 + 6 - 2 = D_i + c - 2$ for each task τ_i. (We ignore the blocking tasks here since we discussed their delay already above.)

For the case that ϕ is not satisfiable it was shown in the proof of Theorem 3.5 that the length of each non-periodic schedule is at least 8. Since the paths of all packets have length 3 we conclude that in each non-periodic schedule there has to be at least one packet which is delayed at least 5 times. This implies the same statement for any periodic schedule. Hence, there can be no short periodic schedule. □

Note that we cannot tighten this construction any further (assuming that $P \neq NP$) since our template schedule $DTREE(I)$ guarantees a limit of $c - 1$ for each task and it can be computed in polynomial time.

The above result is stated only for the strict PPRP. Note that in the sporadic case no schedule can guarantee less than $c - 1$ delays for each task: The release times of the packets can be chosen such that there is an arc on which c packets arrive at the same time (if they are not delayed already before).

4.3.2 Bidirected Trees

In this section, we show how we can adapt the technique introduced for template schedules on directed trees to (direct and indirect) template schedules on bidirected trees. First, we describe the schedule $BTREE(I)$ which is indirect

and guarantees a limit of $2c - c\left(\frac{1}{2}\right)^{\lceil \mathrm{diam}(G)/2 \rceil - 1} + D_i - 1$ ($\mathrm{diam}(G)$ denotes the diameter of G). We make use of the technique which we developed above for directed trees. A straight forward adaption would lead to a limit of $2c + D_i - 2$ for each task τ_i. However, by carefully distributing the delays among the tasks we obtain a better bound. Then, we show that at least for the case $c = 2$ our limit cannot be improved: We describe a suitable lower-bound instance. However, we give a randomized algorithm which guarantees a limit of $1.5c + D_i - 1$ *in expectation* for each task τ_i.

Then we study direct schedules on bidirected trees. We show that one needs to require a certain bound on $\frac{c}{p}$ because there are instances with $c = \frac{3}{4}p$ for which there exists no direct schedule. However, we prove that given an instance with $c \leq \frac{p}{2}$ we can find a direct schedule $BTREE_{dir}(I)$ which guarantees a limit of $2c + D_i - 2$ for each task τ_i.

First, we introduce some structure on the tree which we will use for all algorithms in the sequel.

Definition 4.9 (Tree-structure)**.** Let G be a directed or bidirected tree. We define an arbitrary vertex v_r to be the root vertex. We call an arc e an *up*-arc if it is oriented towards v_r and a *down*-arc if it is oriented away from v_r. For a vertex v let $d(v)$ be the distance between v_r and v. For each task τ_i we define the vertex v_i which is closest to v_r to be the *peak vertex of* τ_i. For a task τ_i we define its *depth* $d(\tau_i)$ by $d(\tau_i) := d(v_i)$. We say a packet *moves up* if it is using an up-arc. A packet *moves down* if it uses a down-arc.

Deterministic Algorithm

Now we describe our indirect schedule $BTREE(I)$ which guarantees a limit of $2c - c\left(\frac{1}{2}\right)^{\lceil \mathrm{diam}(G)/2 \rceil - 1} + D_i - 1$. Intuitively, the schedule works as follows: on the way towards the root, the packets of each task are delayed only in their start vertex. This results in a delay of at most $c - 1$ in the start vertex. The delay of a task τ_i on a down-arc $e = (u, v)$ depends on the depth of e (defined by $d(e) := d(u)$) and on whether e is the first down-arc of τ_i. If e is the first down-arc on P_i then τ_i is delayed up to $c\left(1 - \left(\frac{1}{2}\right)^{d(e)}\right)$ times before it can use e. If e is not the first down-arc on P_i then τ_i is delayed at most $c\left(\frac{1}{2}\right)^{d(e)}$ times before it can use e. In total, on its way down a task τ_i can be delayed only up to

$$c\left(1 - \left(\frac{1}{2}\right)^{d(e)} + \sum_{k=d(e)+1}^{\ell} \left(\frac{1}{2}\right)^{k}\right) = c \cdot \sum_{k=1}^{\ell} \left(\frac{1}{2}\right)^{k} \leq c \cdot \left(1 - \left(\frac{1}{2}\right)^{d(G)}\right)$$

times, assuming that e is the first down-arc on P_i and P_i has $\ell + 1$ down-arcs in total (we denote by $d(G)$ the maximum depth of a vertex in G). By choosing the root v_r such that $d(G) \leq \lceil \mathrm{diam}(G)/2 \rceil$ we achieve the bound on the maximum delay stated above.

Now we describe the algorithm in detail. We choose the root v_r such that $d(G) \leq \lceil \mathrm{diam}(G)/2 \rceil$. For our template schedule we define $\bar{p} := c$. We define

Algorithm $BTREE(I)$

1. Define values for task from top to bottom. Start with root vertex v_r such that in v_r no task is delayed, like in $DTREE(I)$
2. On an intermediate vertex v
   (a) Tasks moving up from v are not delayed on v
   (b) Tasks moving down from v with peak vertex above v are delayed at most $c\left(\frac{1}{2}\right)^{d(v)}$ times
   (c) Tasks with peak vertex v are delayed at most $c\left(1-\left(\frac{1}{2}\right)^{d(v)}\right)$ times on v

the values of the map task(e,k) for all arcs e adjacent to v_r such that no task is delayed in v_r. The problem of finding such values can be reduced to finding a direct schedule in a directed tree, see Section 4.3.1.

We assume by induction that for all arcs e adjacent to each vertex v with $d(v) \leq n$ the respective values of the map task(e,k) have been defined. Now consider a vertex v with $d(v) = n+1$. Denote by G_v the subtree rooted at v. Let $e_v^{\uparrow}$ and $e_v^{\downarrow}$ be the two arcs on the two directed paths between v_r and v which are adjacent to v. Denote by $T_v^{(\text{up})}$ the tasks whose path uses v and $e_v^{\uparrow}$ and by $T_v^{(\text{down})}$ the tasks whose path uses v and $e_v^{\downarrow}$. Let $T_v^{(\text{peak})}$ be the tasks for which v is the peak vertex. Recall that for a down-arc $e=(u,v)$ we defined its depth $d(e)$ by $d(e)=d(u)$.

Due to the induction we have already defined the offsets when the tasks in $T_v^{(\text{down})}$ and $T_v^{(\text{up})}$ use the edges $e_v^{\downarrow}$ and $e_v^{\uparrow}$. Assume that these offsets are fixed, we need to determine offsets for the tasks $T_v := T_v^{(\text{up})} \cup T_v^{(\text{down})} \cup T_v^{(\text{peak})}$ to use the other arcs adjacent to v. We do this as follows: For the tasks T_v we define a schedule such that none of them is delayed on v. Note that this problem can be reduced to finding a direct schedule on a directed tree, see Section 4.3.2. In particular, there is a map task$'$ (defining such a schedule) such that for all $k \in \{0,...,c-1\}$ with task$\left(e_v^{\uparrow},k\right) \in T_v^{(\text{up})}$ we have that task$'\left(e_v^{\uparrow},k\right) = \text{task}\left(e_v^{\uparrow},k\right)$.

For all up-arcs e which are adjacent to v we define task$(e,k) :=$ task$'(e,k)$ for all $k \in \{0,...,c-1\}$. (Note that this includes $e_v^{\uparrow}$ as well as all up-arcs which are oriented towards v.) Now we need to define the values task(e,k) for down-arcs e which are adjacent to v. In general, for the tasks which use these arcs we cannot avoid giving them some delay on v. Let $e=(v,w)$ be a down-arc. Let $T_e^{(\text{down})}$ denote all tasks which use e and which also use $e_v^{\downarrow}$. Let $T_e^{(\text{peak})}$ denote all tasks which use e and for which v is their peak vertex. There are two cases: If $\left|T_e^{(\text{down})}\right| \geq c\left(1-\left(\frac{1}{2}\right)^{d(v)}\right)$ then we define the map task such that

the tasks $T_e^{(\text{peak})}$ are not delayed in v. Since $\left|T_e^{(\text{peak})}\right| < c\left(\frac{1}{2}\right)^{d(v)}$ this can be achieved by delaying each task $\tau \in T_e^{(\text{down})}$ at most $c\left(\frac{1}{2}\right)^{d(v)}$ times. Similarly, if $\left|T_e^{(\text{down})}\right| < c\left(1-\left(\frac{1}{2}\right)^{d(v)}\right)$ we define task such that tasks $T_e^{(\text{down})}$ are not delayed in v and each task $\tau \in T_e^{(\text{peak})}$ is delayed at most $c\left(1-\left(\frac{1}{2}\right)^{d(v)}\right)$ times. Denote by $BTREE(I)$ the schedule resulting from the map task.

Theorem 4.10. *Let I be an instance of the sporadic PPRP on a bidirected tree with $c \leq p$. The schedule $BTREE(I)$ is a template schedule which guarantees a limit of $2c - c\left(\frac{1}{2}\right)^{\lceil \text{diam}(G)/2\rceil - 1} + D_i - 1$ for each task τ_i.*

Proof. Let τ_i be a task with peak vertex v_i. Let M_i be a packet created by τ_i. From the definition of task it follows that on its way up M_i is delayed only in its start vertex (at most $\bar{p}-1 = c-1$ times). On v_i the packet M_i is delayed at most $c\left(1-\left(\frac{1}{2}\right)^{d(v_i)}\right)$ times. Now let $e = (v, w)$ be a down-arc on P_i which is not adjacent to v_i (i. e., $v \neq v_i$). By construction, M_i is delayed at most $c\left(\frac{1}{2}\right)^{d(v)}$ times on v. Denote by $P_i^{\downarrow}$ all vertices on the way down of τ_i, excluding v_i and t_i. Note that $d(v) \leq d(G) - 1$ for each vertex $v \in P_i^{\downarrow}$. We calculate that in total M_i is delayed at most

$$\begin{aligned} c\left(1-\left(\frac{1}{2}\right)^{d(v_i)}\right) + \sum_{v \in P_i^{\downarrow}} c\left(\frac{1}{2}\right)^{d(v)} &= c\left(\sum_{k=1}^{d(t_i)-1}\left(\frac{1}{2}\right)^k\right) \\ &\leq c\left(\sum_{k=1}^{d(G)-1}\left(\frac{1}{2}\right)^k\right) \\ &\leq c\left(1-\left(\frac{1}{2}\right)^{\lceil \text{diam}(G)/2\rceil - 1}\right) \end{aligned}$$

times. This proves a limit of $2c - c\left(\frac{1}{2}\right)^{\lceil \text{diam}(G)/2\rceil - 1} + D_i - 1$ for each packet created by τ_i. □

For the case $c = 2$ our analysis of $BTREE(I)$ guarantees a bound of $2c + D_i - 2$. Now we show that this is indeed best possible.

Proposition 4.11. *There is an instance I of the strict PPRP on a bidirected tree with $c = p = 2$ such that in any template schedule there is a task τ_i such that the packets created by τ_i reach t_i after at least $2c + D_i - 2$ steps.*

Proof. We say a task τ is *delayed* by a schedule if the packets created by τ are delayed. In our instance I we first ensure that there is one task which is delayed at least once: Define two tasks τ_1 and τ_2 such that $s_1 = s_2$ and P_1 and P_2 share the first arc $\bar{e}$ on their path. Starting with this setup, we let P_1 continue in a gadget G_1 and P_2 in a gadget G_2. See Figure 4.3 for a sketch. The gadgets are

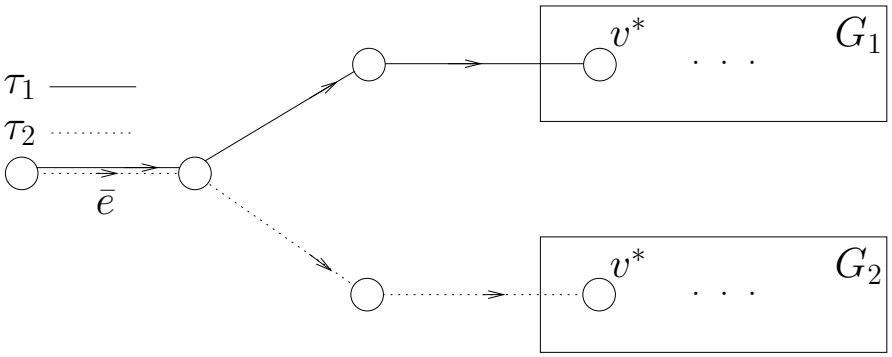


Figure 4.3: The graph used for the proof of Proposition 4.11. The boxes G_1 and G_2 denote the two copies of the gadget G shown in Figure 4.4.

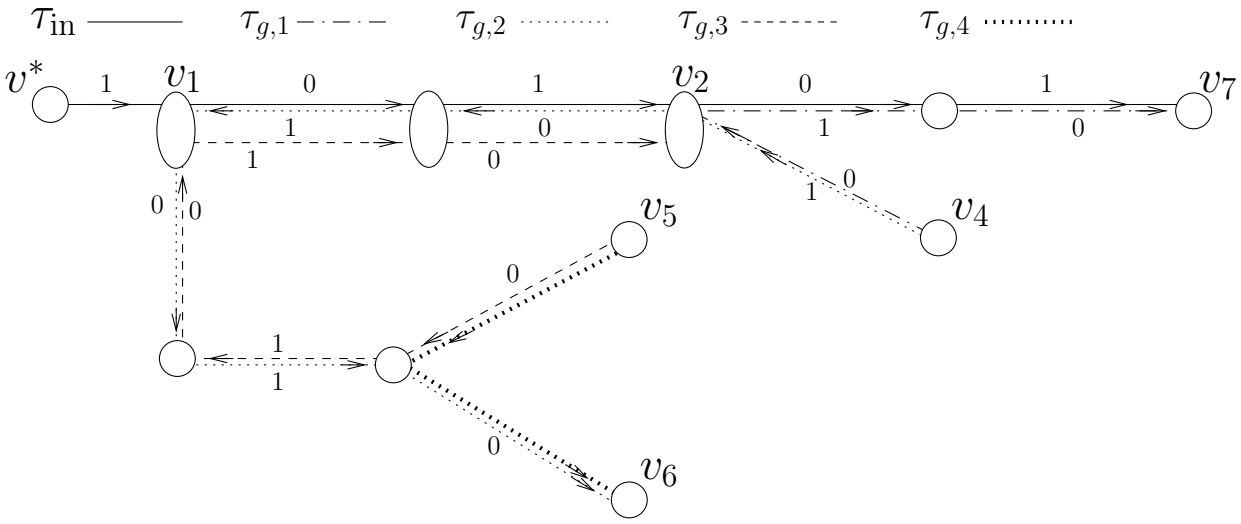


Figure 4.4: The gadget G_i used in the proof of Proposition 4.11. The numbers 0 and 1 represent whether the tasks use the corresponding arcs in even or odd timesteps, respectively.

defined below. We say the tasks τ_1 and τ_2 are the *ingoing tasks* of the respective gadgets. The gadgets G_1 and G_2 are identical and ensure the following: If the ingoing task τ was delayed before entering the gadget, then either τ is delayed again, or another tasks which is defined inside the gadget is delayed twice. This ensures that in the overall instance there is a task which is delayed at least twice and thus its packets reach their destination after at least $2c + D_i - 2$ steps.

We assume that the packets created by τ_1 and τ_2 would use $\bar{e}$ on even timesteps if they were not delayed. In our construction an ingoing task τ arrives on the first vertex v^* inside the gadget on even timesteps if it was delayed before entering the gadget. Now we describe the gadgets G_i. Their graph is depicted in Figure 4.4. Let τ_{in} denote the ingoing task of G_i. We introduce the tasks $\tau_{g,1}, \tau_{g,2}, \tau_{g,3}$ and $\tau_{g,4}$. The paths of these tasks are shown in Figure 4.4 and Table 4.4. Assume that τ_{in} was delayed once before entering the gadget. We show that then there is at least one task which is delayed twice. Since τ_{in} was delayed

Task	Path
τ_{in}	$v^*, v_1, ..., v_2, ..., v_7$
$\tau_{g,1}$	$v_4, v_2, ..., v_7$
$\tau_{g,2}$	$v_4, v_2, ..., v_1, ..., v_6$
$\tau_{g,3}$	$v_5, ..., v_1, ..., v_2$
$\tau_{g,4}$	$v_5, ..., v_6$

Table 4.4: The paths of the tasks defined inside the gadgets G_i.

Algorithm $RTREE(I)$

1. Split instance I into instance I_{up} (parts of the paths towards v_r) and instance I_{down} (parts of the paths away from v_r).
2. Compute maps task_{up} for $DTREE\,(I_{\text{up}})$ and $\text{task}_{\text{down}}$ for $DTREE\,(I_{\text{down}})$.
3. Choose offsets for task_{up} and $\text{task}_{\text{down}}$ uniformly at random.
4. Combine resulting maps to schedule.

once $\tau_{g,1}$ is never delayed (if it was delayed before reaching v_2 then it would be delayed again at v_2 by τ_{in}). Thus, $\tau_{g,2}$ is delayed once (by $\tau_{g,1}$). Since τ_{in} is not delayed inside G_i we conclude that $\tau_{g,3}$ is not delayed either. However, this implies that $\tau_{g,4}$ is delayed by $\tau_{g,3}$ and $\tau_{g,2}$ which yields a contradiction. □

Randomized Algorithm

Recall that the deterministic algorithm $BTREE(I)$ guarantees a limit of $2c - c\left(\frac{1}{2}\right)^{\lceil \text{diam}(G)/2 \rceil - 1} + D_i - 1$ for each task τ_i. Now we present a randomized algorithm for periodic packet routing on a bidirected tree. In expectation, this algorithm gives a much better limit for each task.

For the strict PPRP it guarantees a limit of $c + D_i - 1$ in expectation and for the sporadic PPRP a limit of $1.5c + D_i - 1$ in expectation for each task τ_i. Depending on the outcome of the random experiment the delay of each task differs. However, independent of the random experiment, no task has to suffer a higher delay than $2c - 2$. The main technique is the following. We split the bidirected tree into two directed trees, the edges which are oriented towards the root and the other edges. For both of them we compute the schedule $DTREE$. Then we define the offset of each of the two schedules randomly.

Now we describe the algorithm formally. Let I be an instance of the strict or sporadic PPRP on a bidirected tree with $c \leq p$. We denote by G_{up} and G_{down} the directed trees obtained by taking G and considering only the up- or the down-arcs, respectively. For each task $\tau_i = (s_i, t_i)$ we define two tasks $\tau_{\text{up},i} = (s_i, v_i)$ and $\tau_{\text{down},i} = (v_i, t_i)$. We define $T_{\text{up}} = \{\tau_{\text{up},i} | \tau_i \in T\}$ and $T_{\text{down}} = \{T_{\text{down},i} | \tau_i \in T\}$. Let $I_{\text{up}} = (G_{\text{up}}, T_{\text{up}}, p)$ and $I_{\text{down}} = (G_{\text{down}}, T_{\text{down}}, p)$. With-

out loss of generality we assume that I_{up} and I_{down} have the same congestion c. We compute the schedules $DTREE\,(I_{\text{up}})$ and $DTREE\,(I_{\text{down}})$ and the respective maps task_{up} and $\text{task}_{\text{down}}$ (see Section 4.3.1).

We observe that for each k the map $\text{task}_{\text{up}}^{(k)}$ defined by $\text{task}_{\text{up}}^{(k)}(e,t) := \text{task}_{\text{up}}(e, t+k \bmod c)$ yields a valid task assignment. We define the maps $\text{task}_{\text{down}}^{(k)}$ (for the different values of k) similarly. Our randomized algorithm works as follows: we choose integers k and k' independently and uniformly at random from the set $\{0, ..., c-1\}$.

Then we compute the map task from the maps $\text{task}_{\text{up}}^{(k)}$ and $\text{task}_{\text{down}}^{(k)}$: for each up-arc e we define $\text{task}\,(e,t) = \text{task}_{up}^{(k)}\,(e,t)$ for all $t \in \{0, ..., c-1\}$. For each down-arc e' we define $\text{task}\,(e',t) = \text{task}_{\text{down}}^{(k')}\,(e',t)$ for all $t \in \{0, ..., c-1\}$. Using the map task we define the template schedule $RTREE(I)$.

Theorem 4.12. *Let I be an instance of the PPRP on a bidirected tree with $c \leq p$. The schedule $RTREE(I)$ guarantees a limit of*

- $c + D_i - 1$ in expectation for each task τ_i if I is an instance of the strict PPRP.
- $1.5c + D_i - 1$ in expectation for each task τ_i if I is an instance of the sporadic PPRP.
- $2c + D_i - 2$ for each task τ_i independently of the outcome of the random experiment.

Proof. In the strict periodic setting, once k and k' are fixed all packets created by a task τ_i have the same delays at s_i and v_i. The values k and k' were chosen uniformly at random from the set $\{0, ..., c-1\}$. Thus, the packets of each task τ_i have an expected delay of $\frac{c-1}{2}$ in s_i and an expected delay of $\frac{c-1}{2}$ in v_i. This yields an expected limit of $c + D_i - 1$ in the strict periodic setting.

In the sporadic setting the packets created by τ_i have an expected delay of $\frac{c-1}{2}$ in v_i. However, they might be delayed up to $c-1$ times in s_i (depending in their release time). This yields an expected limit of $1.5c + D_i - 1$ in the sporadic setting.

Even in the worst possible outcome of the random experiment, each packet can be delayed at most $c-1$ times in its start vertex and at most $c-1$ times in its peak vertex. This gives a total limit of $2c + D_i - 2$ for each task τ_i (strict and sporadic setting). □

Direct Schedules

The schedules for bidirected trees presented so far are not direct, i. e., packets might wait in vertices different from their start vertex. Direct schedules have certain advantages: They are completely defined by the start offset of each task which makes them more compact. Also, they are easier to execute since on intermediate nodes no scheduling decision has to be taken. Additionally, the

Algorithm $BTREE_{\text{dir}}(I)$

1. Sort task non-ascendingly by their depth.
2. Consider the tasks in this order. Assign each task the smallest initial waiting time such that its packets do not collide with packets of previously considered tasks.

links in the network do not need any waiting queues. Only the devices which insert the packets into the network must be able to withhold packets for some time.

We present an algorithm for computing a direct schedule $BTREE_{\text{dir}}(I)$ for a packet routing instance on a bidirected tree. It requires that $c \leq p/2$. This might look restrictive at first glance. However, we will show afterwards that there are instances with $c = \frac{3}{4}p$ for which there does not exist a direct periodic schedule at all. Thus, we need a certain upper bound on the congestion c in order to guarantee that there actually is a direct periodic schedule which we can compute.

Let $I = (G, T)$ be an instance of the sporadic PPRP on a bidirected tree G such that $c \leq p/2$. We present our algorithm which finds a direct template schedule $BTREE_{\text{dir}}(I)$ with limit $D_i + 2c - 1$ for each task τ_i. It is based on the concepts presented in [17, Theorem 3.4].

We sort the tasks descendingly by their depth $d(\tau_i)$. W. l. o. g. let $\tau_1, \tau_2, ..., \tau_{|T|}$ be this order. Our schedule is a direct schedule, i. e., each packet is delayed for a certain number of steps and then moves to its destination without being delayed any further. We define the schedule via a template schedule with periodicity $\bar{p} := 2c$. We iterate over the tasks with $i = 1$ to $|T|$. Consider the i-th iteration. Let $P_i = \left\{v_0, v_1, ..., v_{|P_i|-1}\right\}$ be the path of τ_i and let $e_j = (v_j, v_{j+1})$ for all $j \in \{0, ..., |P_i| - 2\}$. Let k_i be the smallest positive integer such that $\text{task}\,(e_j, k_i + j \bmod \bar{p}) = \text{none}$ for all relevant values for j. We assign τ_i the initial waiting time k_i and define $\text{task}\,(e_j, k_i + j \bmod \bar{p}) = \tau_i$ for all respective values for j. We will show in Theorem 4.13 that there is always a value for k_i with $0 \leq k_i \leq 2c - 2 < \bar{p}$. We denote by $BTREE_{\text{dir}}(I)$ the resulting schedule.

Theorem 4.13. *Let I be an instance of the sporadic PPRP on a bidirected tree with $c \leq p/2$. The schedule $BTREE_{\text{dir}}(I)$ is a well-defined direct template schedule which guarantees a limit of $2c + D_i - 2$ for each task τ_i.*

Proof. Let τ_i be a task. First of all we show that we can always find a value k_i with $k_i \leq \bar{p} - 1$ which does not interfere with any previous assignment for tasks $\tau_{i'}$ with $0 \leq i' < i$. Let e and e' be the two arcs in P_i which are incident to v_i. (The case that there is only one such arc can be proven with a similar reasoning as below.) Recall that we considered the tasks in an order given by the values $d\,(\tau_i)$. Thus, when assigning the initial delay k_i to τ_i only tasks which use e or e' could possibly interfere. We say a task $\tau_{i'}$ with $0 \leq i' < i$ *blocks* a timeslot k if there is an arc e_j such that $\text{task}\,(e_j, k + j \bmod \bar{p}) = \tau_{i'}$. Since G

is a bidirected tree from the definition of task it follows that each task which uses e or e' can block at most one timeslot k with $0 \leq k < c$.

Excluding τ_i there are at most $2\,(c-1)$ tasks whose path uses e or e'. This implies that there is a value for k_i with $0 \leq k_i \leq 2c-2$ such that k_i is not blocked and thus $k_i \leq 2c-2$. Hence, $BTREE_{\text{dir}}(I)$ is well-defined. By definition of task each packet created by τ_i waits in s_i for k_i timesteps and then moves to t_i without being delayed any further. Thus, $BTREE_{\text{dir}}(I)$ is a direct periodic schedule and $2c + D_i - 2$ is a valid limit for each task τ_i. □

Limits of Direct Schedules

The algorithm $BTREE_{\text{dir}}(I)$ presented in the previous section needs the condition that $c \leq p/2$. This raises the question whether such a bound on the congestion is really necessary. Now we prove that this is indeed the case: we present an instance with $c = \frac{3}{4}p$ for which there can be no direct periodic schedule. Even worse, without a bound on c (apart from the necessary $c \leq p$) it is even NP-hard to determine whether there is a direct template schedule or not. This justifies our required bound on the congestion in $BTREE_{\text{dir}}(I)$.

First, we describe our instance with $c = \frac{3}{4}p$ for which there is no direct schedules. It is known that there exists an instance $I = (G, \mathcal{P})$ of the Path Coloring problem (i. e., the problem of coloring given paths such that two paths which share an arc are colored with different colors) with the following properties (see [58]):

- the underlying graph G is a bidirected tree,
- for each arc e there are at most three paths in $\mathcal{P}$ which use e, and
- any feasible path coloring for I needs at least five colors.

Starting from I we construct an instance $\bar{I} = \left(\bar{G}, T, p\right)$ of the periodic packet routing problem. The graph $\bar{G}$ is constructed as follows: starting with G, we replace each pair of anti-parallel arcs by a path with twelve anti-parallel arcs. We choose twelve arcs since 12 is the least common multiple of 1,2,3, and 4. We call all vertices which exist in both G and $\bar{G}$ the *old vertices*. For each path $P_i \in \mathcal{P}$ starting in a vertex u and ending in a vertex v we add a task $\tau_i = (u, v)$. Finally, we define $p := 4$.

Proposition 4.14. *For the instance $\bar{I}$ of the strict PPRP it holds that $c \leq \frac{3}{4}p$ but it has no direct periodic template schedule.*

Proof. Since in I each arc is used by at most three paths and $p = 4$ we have that $c \leq \frac{3}{4}p$ for our instance $\bar{I}$. We show that there can be no direct template schedule for $\bar{I}$ by using the fact that any path coloring for I needs at least five colors. Assume on the contrary that there is a direct template schedule for $\bar{I}$ with periodicity $\bar{p}$. We know that $1 \leq \bar{p} \leq 4$. Denote by $\text{task}(\tau_i, u)$ the offset when packets created by τ_j leave each vertex $u \in P_i$. Each arc in G corresponds to a directed path of length twelve in $\bar{G}$. We conclude that there

is a value $k_i \in \{0,1,2,3\}$ for each task τ_i such that $\text{task}(\tau_i, u) \equiv k_i \bmod \bar{p}$ for each old vertex $u \in P_i \setminus \{t_i\}$. Thus, for two tasks τ_i and τ_j whose paths share an arc it holds that $k_i \neq k_j$. This implies that the map $c(P_i) := k_i$ defines a path coloring for I with four colors. But this is a contradiction since each path coloring for I needs at least five colors. □

Now we show that if we do not impose any bound on c (apart from $c \leq p$) then it is even NP-hard to determine whether there is a direct template schedule (for an instance of the PPRP on a bidirected tree).

Theorem 4.15. *For instances of the strict or sporadic PPRP on bidirected trees it is NP-hard to decide whether there is a direct template schedule.*

Proof. We give a reduction from Directed-Path-Coloring on bidirected binary trees: Given a binary tree G, a set of directed paths $\mathcal{P}$ on G. The question is whether it is possible to color the paths in $\mathcal{P}$ with three colors such that each two paths which use an arc in the same direction have different colors. This problem is NP-hard [33].

The following construction is similar to the construction used for Proposition 4.14. Given an instance $(G, \mathcal{P})$ of the Directed-Path-Coloring such that $G = (V, E)$ is a bidirected tree we construct an instance (G', T') of the periodic packet routing problem as follows: We obtain $G' = (V', E')$ by taking G and replacing each pair of arcs between two vertices u and v by a path with anti-parallel arcs of length six between u and v. We call the vertices in V' which already existed in V the *old* vertices, all other vertices are called the *new* vertices. We note that G' is also a bidirected tree. For each vertex $v \in V$ there is a corresponding (old) vertex $v' \in V'$. For each path $P_i \in \mathcal{P}$ from $s_i \in V$ to $t_i \in V$ we introduce a task $\tau_i = (s_i', t_i') \in T'$. Since G' is a tree, the path for τ_i is implicitly given. We define $p := 3$.

Now we prove that the paths $\mathcal{P}$ can be colored with three colors if and only if there is a direct template schedule for (G', T', p). First assume that there is a valid coloring $f : \mathcal{P} \to \{0,1,2\}$ for the paths $\mathcal{P}$. We define a template schedule by setting $\text{task}(e_i, f(P_i)) := \tau_i$ for each task τ_i with e_i being the first arc on P_i. We define the remaining values for task such that the resulting schedule is direct. Also, we define $\bar{p} := p = 3$. For an old vertex $u \in P_i$ let $\text{leave}(\tau_i, u, j)$ be the time when the packet $M_{i,j}$ leaves u. We observe that $\text{leave}(\tau_i, u, j) \equiv f(P_i) \bmod 3$ for all old vertices $u \in P_i$ and all positive integers j. Two packets $M_{i,j}$ and $M_{i',j'}$ can collide only if there is an old vertex u and an arc $e = (u, v)$ which P_i and $P_{i'}$ have in common and if $\text{leave}(\tau_i, u, j) = \text{leave}(\tau_{i'}, u, j')$. But the latter implies that $f(P_i) = \text{leave}(\tau_i, u, j) \bmod 3 = \text{leave}(\tau_{i'}, u, j') \bmod 3 = f(P_{i'})$. But this cannot happen since f is a valid path coloring.

Now assume that we are given a valid direct template schedule S for (G', T', p). Note that $\bar{p} \leq p = 3$. For each task τ_i we define e_i to be the first arc on P_i. We define our path coloring $f : \mathcal{P} \to \{0,1,2\}$ such that $\text{task}(e_i, f(P_i)) = \tau_i$. We observe that $\text{task}(\bar{e}, f(P_i)) = \tau_i$ for each arc $\bar{e} \in P_i$ which points out of an old vertex in P_i. This holds since each (bidirected) edge in G was replaced by a path of anti-parallel arcs with length six in G' and $\bar{p} \in \{1,2,3\}$. Thus, if f was

not a valid path coloring then there would be an arc $\bar{e}$ pointing out of an old vertex and two paths $P_i \neq P_j$ with $f(P_i) = f(P_j)$ which both use $\bar{e}$. However, this would imply that $\tau_i = \text{task}(\bar{e}, f(P_i)) = \text{task}(\bar{e}, f(P_j)) = \tau_j$ which is a contradiction since $P_i \neq P_j \Rightarrow \tau_i \neq \tau_j$. □

4.3.3 Undirected Trees

Now we adjust the techniques introduced above for bidirected trees to undirected trees. Here, we need to take care that two packets which move towards each other do not interfere. Intuitively, the result is that the actual available bandwidth of the edges drops by a factor of 2. Hence, the constraint on the congestion for each algorithm needs to be by a factor of 2 stricter than in the setting of bidirected trees. We will also give example instances that show that this restriction is really necessary.

Let I be an instance of the sporadic PPRP on an undirected tree G such that $c \leq p/2$. We present an algorithm which finds an (indirect) schedule $UTREE(I)$ with limit $4c - 2c\left(\frac{1}{2}\right)^{\lceil \text{diam}(G)/2 \rceil - 1} + D_i - 1$ for each task τ_i. We will prove below that the bound for the congestion is necessary. In particular, we will show that for any $\alpha, \beta, \varepsilon > 0$ there is an instance I of the strict PPRP with $p/2 < c \leq (1+\varepsilon)\, p/2$ for which no template schedule can guarantee a limit of $\alpha p + D_i + \beta$ for every task τ_i.

After that we study the problem of finding directed schedules for the sporadic PPRP on undirected trees. We present an algorithm $UTREE_{\text{dir}}(I)$ which finds a direct template schedule with limit $4c + D_i - 3$ for each task τ_i if $c \leq \frac{p}{4}$. We justify the bound on the congestion by giving an instance on an undirected path with $p/2 < c \leq (1+\varepsilon)\, p/2$ for which there exists no direct template schedule at all.

Indirect Schedules

Now we present our algorithm for computing the indirect schedule $UTREE(I)$. It adapts the concepts of $BTREE(I)$ to undirected trees. In contrast to bidirected trees we need to avoid collisions by packets that use the same edge in opposite directions. For packets which have left their start vertex and which are located on a vertex v at a time t we ensure that $d(v) + t$ is always even. This implies that the packets which actually move are either all located on a vertex with even depth or all located on a vertex with odd depth. Then, no collision can occur. We call this the *parity property*. The parity property can be understood as halving the bandwidth. However, since $c \leq p/2$ there is still enough bandwidth available.

Now we give a formal definition. We define $\bar{p} := 2c$. Like in $BTREE(I)$ we start with the root v_r (chosen such that $d(G) \leq \lceil \text{diam}(G)/2 \rceil$). We define the map task such that no packet needs to wait in v_r and such that for an edge $e = \{v, v_r\}$ we have that $\text{task}(e, k) = \tau$ only if the packets of τ move from v to v_r and k is odd or the packets of τ move from v_r to v and k is even. Such values for task can be found since $c \leq p/2$. We continue with the same iterative

Algorithm $UTREE(I)$

1. Adapt algorithm $BTREE(I)$ to undirected trees.
2. Always ensure that moving packets are either all on vertices with even level or all on vertices with odd level (parity property).

procedure as in $BTREE(I)$. However, in order to ensure the parity property now if a packet is delayed it needs to wait twice as many times as in $BTREE(I)$. Again, the fact that $c \leq p/2$ ensures that there is enough available bandwidth.

Since $\bar{p} = 2c$ each packet is delayed at most $2c - 1$ times in its start vertex. After that, each packet is delayed at most $2c\left(1 - \left(\frac{1}{2}\right)^{\lceil \mathrm{diam}(G)/2 \rceil - 1}\right)$ times. This gives a limit of $4c - 2c\left(\frac{1}{2}\right)^{\lceil \mathrm{diam}(G)/2 \rceil - 1} + D_i - 1$ for each task τ_i. Denote by $UTREE\,(I)$ the schedule resulting.

Theorem 4.16. *Let I be an instance of the sporadic PPRP on an undirected tree with $c \leq p/2$. For each task τ_i the schedule $UTREE(I)$ guarantees a limit of $4c - 2c\left(\frac{1}{2}\right)^{\lceil \mathrm{diam}(G)/2 \rceil - 1} + D_i - 1$.*

Proof. Similar proof as for Theorem 4.10. □

For $UTREE(I)$ we required that $c \leq p/2$. This might look restrictive. However, we show with following proposition that this bound is really necessary for a schedule with a good limit for each task.

Proposition 4.17. *For any $\alpha, \beta, \varepsilon > 0$ there is an instances I of the strict PPRP with $p/2 < c \leq (1+\varepsilon)\,p/2$ on an undirected path for which there is no template schedule which guarantees a limit of $\alpha p + D_i + \beta$ for every task τ_i.*

Note that the same statement is implied for the sporadic PPRP (since the strict periodic setting is a special case of the sporadic setting). We will prove the proposition in the sequel.

Direct Schedules

We study the problem of computing a direct schedule for an undirected tree. We show how to adapt the algorithm $BTREE_{\mathrm{dir}}(I)$ to undirected trees. Given an instance I of the sporadic PPRP on an undirected tree with $c \leq \frac{p}{4}$. We show how to compute the schedule $UTREE_{\mathrm{dir}}(I)$ which guarantees a limit of $4c + D_i - 3$ for each task τ_i. Afterwards, we will justify the required bound of $c \leq \frac{p}{4}$: we give an example of an instance with $\frac{p}{2} < c \leq (1+\varepsilon)\frac{p}{2}$ on an undirected path for which there can be no direct schedule at all.

Now we describe the algorithm for computing $UTREE_{\mathrm{dir}}(I)$. We sort the tasks descendingly according to $d(\tau_i)$. W. l. o. g. let $\tau_1, ..., \tau_{|T|}$ be this order. We define a template schedule with periodicity $\bar{p} := 4c$ and iterate over the tasks from $i = 1$ to $|T|$. Let τ_i be the task considered in the i-th iteration. Like in

Algorithm $UTREE_{\text{dir}}(I)$

1. Sort task descendingly by the depth of the peak vertex of their path.
2. Consider the tasks in this order. Assign each task τ_i the smallest initial waiting time k_i such that
   (a) its packets do not collide with packets of previously considered tasks and
   (b) $d(s_i) + k_i$ is even.

$BTREE_{\text{dir}}(I)$ we call a value $k_i < \bar{c}$ *valid* if task $(e_j, k_i + j \bmod 4c) =$ none for all edges e_j on P_i. We assign τ_i the smallest valid value k_i such that $d(s_i) + k_i$ is even. We set task $(e_j, k_i + j \bmod 4c) = \tau_i$ for the respective edges e_j. We will show in Theorem 4.18 that there is always a valid value k_i such that $d(s_i) + k_i$ is even with $k_i \leq 4c - 3$. Denote by $UTREE_{\text{dir}}(I)$ the resulting schedule.

Theorem 4.18. *Let I be an instance of the sporadic PPRP on an undirected tree G with $c \leq p/4$. The schedule $UTREE_{\text{dir}}(I)$ is a well-defined direct template schedule which guarantees a limit of $4c + D_i - 3$ for each task τ_i.*

Proof. Let τ_i be task. First we show that there is always a valid value k_i with $k_i \leq 4c - 3$. Let e and e' be the two edges in P_i which are incident to v_i. (The case that there is only one such edge can be proven with a similar reasoning as below.) When assigning the initial delay w_i to τ_i only tasks which use e or e' could possibly interfere with τ_i. We say a task $\tau_{i'}$ with $0 \leq i' < i$ *blocks* a timeslot k if there is an edge e_j such that task $(e_j, k + j \bmod 4c) = \tau_{i'}$.

Since G is a tree it follows that if two paths P_i and $P_{i'}$ share an edge then either on all edges which they have in common they point in the same direction or on all these edges they point in opposite directions. For ease of notation we say that τ_i and $\tau_{i'}$ *move in the same direction* or they *move in opposite direction*.

First we discuss tasks $\tau_{i'}$ which move in the opposite direction of τ_i. For the analysis we define a time-dependent edge-coloring $c : E \times \mathbb{N} \to \{\text{red}, \text{green}\}$. Denote by $d(u, v)$ the length of a shortest path between two vertices u and v. Let $e = \{u, v\}$ be an edge such that $d(u, v_r) = i$ and $d(v, v_r) = i + 1$. For t such that $t + i$ is even we define $c(e, t) =$ green, for t such that $t + i$ is odd we define $c(e, t) =$ red. From the algorithm it follows that $d(s_{i'}) + k_{i'}$ is even. Since the periodicity $\bar{p}$ is even this implies that on their way up (in direction of v_r) packets created by $\tau_{i'}$ use only red edges, and on their way down they use only green edges. If $d(s_i) + k_i$ is even the same holds for packets created by τ_i. Since τ_i and $\tau_{i'}$ move in opposite directions, if two packets created by these tasks meet then one of them moves up and the other one moves down. Thus, they do not interfere since then they use differently colored edges. This implies that tasks which move in opposite direction of τ_i do not block any timeslots k such that $d(s_i) + k$ is even.

Now let $\tau_{i''}$ be a task which moves in the same direction as τ_i. Since G is a tree $\tau_{i''}$ can block at most one timeslot k with $0 \leq k < \bar{c}$. Excluding τ_i there are at most $2\,(c-1)$ tasks which use e or e' in the same direction as τ_i. Since $c \leq p/4$ this implies that there are at most $2\,(c-1)$ blocked timeslots k such that $k + d(s_i)$ is even. This implies that $k_i \leq 4c - 3 < \bar{p}$ and thus $UTREE_{\mathrm{dir}}(I)$ is well-defined and guarantees a limit of $4c + D_i - 3$ for each task τ_i. From its definition it follows that $UTREE_{\mathrm{dir}}(I)$ is direct. □

Limits of Direct Schedules

Now we show that for any $\varepsilon > 0$ there are instances on an undirected path with $c \leq (1+\varepsilon)p/2$ for which no direct schedule exists. Thus, we have to give some upper bound on the congestion of an instance in order to guarantee that an algorithm can compute a direct schedule for it (as we did for $UTREE_{\mathrm{dir}}(I)$).

Proposition 4.19. *For any $\varepsilon > 0$ there are instances of the strict PPRP on an undirected path with $p/2 < c \leq (1+\varepsilon)\,p/2$ for which there is no direct template schedule.*

Proof. Let $\varepsilon > 0$. We describe how to construct an instance with the described property. Let ℓ be an integer such that $\frac{2}{\ell} \leq \varepsilon$. The graph G is an undirected path with $2\ell + 1$ vertices. Denote by v_A and v_E the vertices at the ends of G. We define $p := 2\ell$. We introduce $\ell + 1$ tasks $\tau_1, ..., \tau_{\ell+1}$ with $\tau_i = (v_A, v_E)$ for all $i \in \{1, 2, ..., \ell + 1\}$ and one task $\tau_{\ell+2} = (v_E, v_A)$. We call the former tasks the *good tasks.* Since each edge is used by all tasks we have $c = \ell + 2 = \frac{p}{2} + \frac{p}{2} \cdot \frac{2}{\ell} \leq (1+\varepsilon)\frac{p}{2}$. Assume on the contrary that there is a direct template schedule for I. Denote by k_i the initial delay for the first packet of each task τ_i. We assume that $k_{\ell+2}$ is even (the case that $k_{\ell+2}$ is odd can be proven similarly). Then each good task τ_i must have an odd initial delay k_i with $0 \leq k_i < 2\ell$. Since there are $\ell + 1$ good tasks but only ℓ odd integers in the interval $[0, ..., 2\ell - 1]$ this yields a contradiction. □

Note that Proposition 4.19 implies the same statement for instances of the sporadic PPRP. Now we can prove Proposition 4.17.

Proof of Proposition 4.17: Let $\alpha, \beta, \varepsilon > 0$. We construct a new instance $I = (G, T, p)$ of the periodic packet routing problem on an undirected path such that $\frac{p}{2} < c \leq (1+\varepsilon) \cdot \frac{p}{2}$ but there can be no periodic schedule which guarantees a limit of $D_i + \alpha \cdot p + \beta$ for each task τ_i.

Let ℓ be an integer such that $\frac{2}{\ell} \leq \varepsilon$. The graph G is an undirected path with $(2\ell - 1)(\ell + 2)(\alpha \cdot p + \beta) + 2$ vertices. Denote by v_A and v_E the vertices at the two ends of G. We introduce $\ell + 1$ tasks $\tau_1, ..., \tau_{\ell+1}$ with $\tau_i = (v_A, v_E)$ for each $i \in \{1, 2, ..., \ell + 1\}$ and one task $\tau_{\ell+2} = (v_E, v_A)$. We set $p := 2\ell$. Like in the proof of Proposition 4.19 we have that $c \leq (1+\varepsilon)\frac{p}{2}$. Assume on the contrary that there is a template schedule S for I in which each task is delayed at most $\alpha \cdot p + \beta$ times. Thus, there can be at most $(\ell + 2)\,(\alpha \cdot p + \beta)$ edges on which

packets are delayed. Since the graph G has $(2\ell-1)(\ell+2)(\alpha \cdot p+\beta)+1$ edges, by the pigeonhole principle there must be 2ℓ consecutive edges on which no packet is delayed. Thus, the schedule S is a direct schedule on this subpath. However, with a similar reasoning as in the proof of Proposition 4.19 we conclude that this is impossible. □

In Theorem 4.15 we showed that for instances of the periodic PPRP on bidirected trees it is NP-hard to decide whether there is a direct template schedule. Now we prove that this holds also for instances of the PPRP on undirected trees.

Theorem 4.20. *For instances of the strict or sporadic PPRP on undirected trees it is NP-hard to decide whether there is a direct template schedule.*

Proof. This can be shown with a similar reduction as Theorem 4.15. Here, if a path is used in only one direction, we introduce an artificial task which uses the path in the opposite direction. This ensures that each path has the same bandwidth in each direction. □

4.4 Global- and Edge-priority Schedules

Now we turn to global-priority and edge-priority schedules in the sporadic setting. We present a global-priority schedule $GPRIO(I)$ and an edge-priority schedule $EPRIO(I)$. For both schedules we bound the maximal delay for each task. The bounds guaranteed by these schedules are worse than the limits we proved for the template schedules in the respective settings. However, with suitable lower bound instances we show that our two priority schedules are (almost) best possible. In particular, this shows that in the settings that we consider template schedules are more powerful than priority schedules.

Later, in Section 4.4.2 we give slightly stronger results for the strict periodic setting. Our priority schedules require a certain bound on the congestion in order to guarantee the desired limits. In the strict periodic setting we can relax these bounds a little. Table 4.2 summarizes the results presented in this section.

First we present a schedule $GPRIO(I)$ for bidirected trees. The schedule assigns the priorities to the tasks according to their depth. Then we study an edge-priority schedule $EPRIO(I)$ for directed trees. It is based on $GPRIO(I)$ but uses a more sophisticated tie-breaking procedure for tasks with the same depth.

Algorithm $GPRIO(I)$

1. Sort tasks by depth $d(\tau_i)$
2. Define $\tau_i \prec \tau_j$ if and only if $d(\tau_i) < d(\tau_j)$.
3. Ties in the prioritization are broken arbitrarily.

Now we present the algorithm for computing the schedule $GPRIO(I)$. Let I be an instance of the sporadic packet routing problem on a bidirected tree. Note that here we do not yet impose any conditions on the congestion or whether I is a strict or sporadic instance. (However, when we prove our limits for $GPRIO(I)$ in Theorems 4.21 and 4.30 we will need a certain bound on c.) We define that a task τ_i has a higher priority than a task τ_j if its peak vertex is closer to the root than the peak vertex of τ_j. Formally, $\tau_i \prec \tau_j$ if and only if $d(\tau_i) < d(\tau_j)$. This yields a partial order $\prec$. We complete $\prec$ to a total order arbitrarily. In particular, this implies that the prioritization of tasks with the same peak vertex are chosen arbitrarily.

Theorem 4.21. *Let I be a instance of the sporadic PPRP on a bidirected tree with $c \leq p/3$. The schedule $GPRIO(I)$ guarantees a limit of $2c + D_i - 2$ for each task τ_i.*

We will prove the theorem in the sequel. It is easy to see that there are instances – even in the strict periodic case – where $GPRIO(I)$ delays a packet $2c - 2$ times. We will show later that $GPRIO(I)$ is best possible in the sense that no global-priority schedule can guarantee a better limit for every task in every instance.

Algorithm $EPRIO(I)$

1. Sort tasks by depth $d(\tau_i)$
2. Define $\tau_i \prec \tau_j$ if and only if $d(\tau_i) < d(\tau_j)$.
3. Ties in prioritization are broken such that each task is delayed at most $1.5c$ times in total.

Now we study our edge-priority schedule $EPRIO(I)$ for directed trees. Let I be an instance of the strict or sporadic packet routing problem on a directed tree. Again, we do not require a bound on c here in the definition of the schedule but later in the theorem. We need to define a prioritization for each arc separately. First, like in $GPRIO(I)$, we define that $\tau \prec \tau'$ if $d(\tau) < d(\tau')$ for each arc prioritization. So now we focus on the tie-breaking for tasks τ, τ' with $d(\tau) = d(\tau')$. Consider a vertex v. Let $e_1, ..., e_r$ be the ingoing arcs of v which are up-arcs and let $e'_1, ..., e'_s$ be the outgoing arcs of v which are down-arcs. Denote by E_v the set of all these arcs. In case that v is not the root vertex let $\bar{e}$ be the remaining arc which is adjacent to v. See Figure 4.5 for a sketch. We define the prioritization for all tasks with peak vertex v.

We define T_v to be the set of tasks which use v. Let T'_v be the tasks in T_v which do not use $\bar{e}$. We compute a minimum path coloring for the paths of the tasks T_v. Exactly c colors are needed, for details see Section 1.3.1. Assume that the paths are colored with colors $\{1, 2, ..., c\}$. If there is an arc $\tilde{e} \in E_v$ such that more than $c/2$ paths use both $\tilde{e}$ and $\bar{e}$ then we assume w. l. o. g. that the paths which use $\bar{e}$ and $\tilde{e}$ use the colors $1, ..., m$ if $\bar{e}$ is an up-arc and the

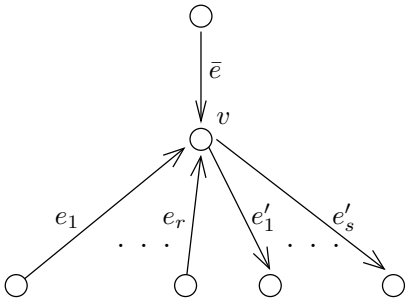


Figure 4.5: Sketch for the definition of $EPRIO(I)$.

colors $c-m+1, ..., c$ if $\bar{e}$ is an down-arc (assuming that there are m such paths). Note that there can be at most one arc $\tilde{e}$ with this property. Denote by $f(\tau)$ the color of the path of a task τ. Let $\tau, \tau' \in T'_v$ be a pair of tasks. Then, on each up-arc in the entire tree which is used by τ and τ' (not only the up-arcs in E_v) we define $\tau \prec \tau'$ if $f(\tau) < f(\tau')$. For each down-arc in the tree we define $\tau \prec \tau'$ if $f(\tau) > f(\tau')$. Denote by $EPRIO(I)$ the resulting schedule.

Theorem 4.22. *Let I be an instance of the sporadic PPRP on a directed tree with $c \leq 2p/5$. The schedule $EPRIO(I)$ guarantees a limit of $\frac{3}{2}c + D_i - 1$.*

We will prove the theorem later. We will also show later that $EPRIO(I)$ is almost best possible. First, we prove properties of $GPRIO(I)$ and $EPRIO(I)$ which we need in order to prove Theorems 4.21 and 4.22. First, we analyze the maximum number of delays which a packet can encounter on its way up (in the direction of the root). Recall that this is the part of P_i between s_i and v_i.

Lemma 4.23. *Let I and $\bar{I}$ be instances of the sporadic PPRP on a bidirected and directed tree, respectively. In the schedules $GPRIO(I)$ and $EPRIO(\bar{I})$ a packet created by a task τ_i can be delayed at most $c-1$ times between s_i and v_i.*

Proof. We prove the claim only for $GPRIO(I)$ since for $EPRIO(\bar{I})$ we can follow exactly the same argumentation. Consider a packet M created by a task τ_i and let $e \in P_i$ be the arc on P_i between s_i and v_i which is closest to v_r. Since we are only interested in the behavior of M between s_i and v_i it is sufficient to consider only the tasks $\tilde{T}$ which use e and whose priority on e is at least as high as the priority of τ_i. W. l. o. g. let $\tau_1 \prec \tau_2 \prec ... \prec \tau_i$ be these tasks.

We consider the following instance I' instead of I. (While doing this we assume that the timesteps when the respective tasks create new packets remain the same.) For each task τ_k let L_k denote the number of arcs on P_k below e. We replace each path P_k by a path P'_k which consists (in this order) of

- a new path of length L_k which is used only by τ_k,
- the arc e, and
- all arcs of P_k above e.

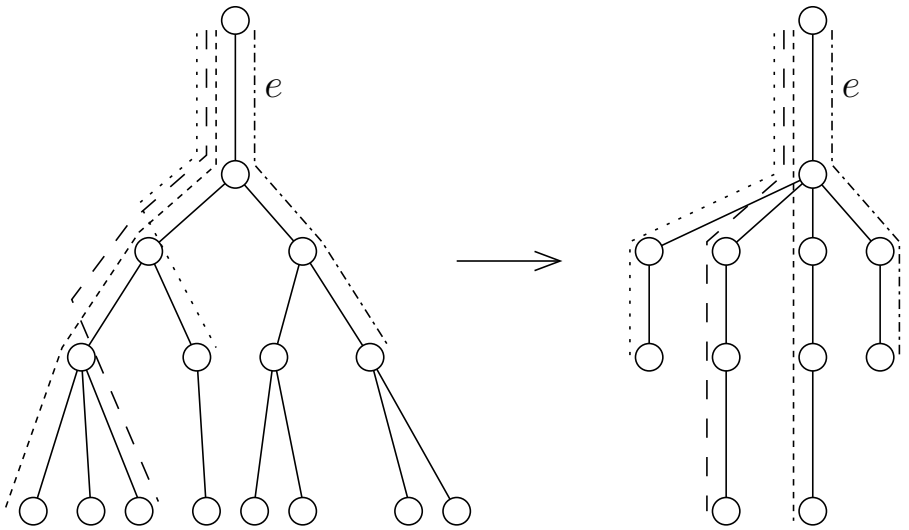


Figure 4.6: The transformation from I to I' in the proof of Lemma 4.23.

See Figure 4.6 for a sketch. In $GPRIO(I)$ and $GPRIO(I')$ we use a global prioritization and all tasks in $\tilde{T}$ use e. Thus, all packets created by tasks in $\tilde{T}$ traverse e in $GPRIO(I)$ at the same timesteps as in $GPRIO(I')$. (This can be shown by an induction which transforms I to I' stepwise.)

So now consider e in I'. All tasks have a period length of p. Before reaching e no packet is delayed. This implies that the packet M is delayed at most $i-1$ times. Since $i \leq c$ this proves the claim. □

Now we focus on the delay of the packets on the way away from the root.

Lemma 4.24. *Let I be an instance of the sporadic PPRP on a bidirected tree with $c \leq p/3$. In the schedule $GPRIO(I)$ on the way away from v_r each packet is delayed at most $c-1$ times.*

Proof. First observe that on the way down a packet is delayed only on the first down-arc. We prove the claim by induction over the depth of the respective tasks. First, consider a task τ_i with $d(\tau_i) = 0$. Let $e = (u, v)$ be the first down-arc on P_i. On the way up each packet is delayed at most $c-1$ times. Thus, two packets created by τ_i reach u with a time difference of at least $\lceil 2p/3 \rceil$. Since $c \leq \lfloor p/3 \rfloor$ in each time interval of length $\lceil 2p/3 \rceil$ there are less than c packets which use e and which have a higher priority than τ_i. Thus, each packet created by τ_i is delayed at most $c-1$ times on its way down.

Now assume the claim is true for all tasks τ_j such $d(\tau_i) \leq m$. Consider a task τ_i with $d(\tau_i) = m+1$. Let again $e = (u, v)$ be the first down-arc on P_i. From the induction hypothesis we conclude that for each task τ_j which has a higher priority than τ_i on e the following holds: Two packets created by τ_j reach u with a time difference of at least $\lceil p/3 \rceil$. Thus, in each interval of length $\lceil p/3 \rceil$ there are less than c packets which use e and have a higher priority than τ_i. Like above, two packets created by τ_i reach u with a time difference of at least $\lceil 2p/3 \rceil$. Thus, each packet created by τ_i is delayed at most $c-1$ times on its way down. □

Finally, we can proof the limits guaranteed by the schedule $GPRIO(I)$.

Proof of Theorem 4.21: From Lemma 4.23 it follows that packets are delayed at most $c-1$ times on their way up. Lemma 4.24 shows that on their way down they are delayed at most $c-1$ times as well. Hence, the schedule $GPRIO(I)$ guarantees a limit of $2c + D_i - 2$ for each task τ_i. □

Combining Lemma 4.23 with a similar reasoning as in Lemma 4.24 we can now prove that $EPRIO(I)$ guarantees a limit of $\frac{3}{2}c + D_i - 1$ for each task.

Proof of Theorem 4.22: Let τ_i be a task with peak vertex v. Assume that the path of τ_i was colored with color $f(\tau_i)$ when v was considered by the algorithm. First, we discuss the case that for no arc $\tilde{e} \in E_v$ there are more than $c/2$ paths which use both $\tilde{e}$ and $\bar{e}$. Then, there are at most $c/2+(f(\tau_i)-1)+(c-f(\tau_i)) = 3c/2-1$ tasks which share an arc with P_i and which have a higher priority than τ_i on these arcs. We say that such tasks *interfere* with τ_i.

Now we discuss the case that there is an arc $\tilde{e} \in E_v$ as described above. If P_i does not use $\tilde{e}$ we can show as above that at most $3c/2-1$ tasks interfere with τ_i. Now consider the case that P_i uses $\tilde{e}$. Due to our assumption on the coloring, for such tasks there are at most $(f(\tau_i)-1)+(c-f(\tau_i)) = c-1$ tasks which interfere with τ_i.

Since also $c \leq 2p/5$ one can show as in Lemma 4.23 that on its way up the packets of τ_i are delayed at most $c/2+(f(\tau_i)-1)$ times. As in Lemma 4.24 one can show inductively that on the way down each packet created by τ_i is delayed at most $c-f(\tau_i)$ times. In the inductive step one uses that two packets created by the same task have a minimum time difference of $2p/5$ and $c \leq 2p/5$. We conclude that each packet is delayed at most $\frac{3}{2}c-1$ times in total. □

Here we would like to comment that without a bound on the congestion c in the schedules $GPRIO(I)$ and $EPRIO(I)$ it might happen that a packet is delayed more than $c-1$ times on its way down. Even more, we will show in Proposition 4.27 that for any $\alpha \geq 0$ there are instances on a directed tree with $c = p$ where *no* edge-priority schedule (and hence also no global-priority schedule) can guarantee a limit of $\alpha c + D_i$ for every task τ_i.

4.4.1 Lower Bounds

Now we prove that the schedules $GPRIO(I)$ and $EPRIO(I)$ are (almost) best possible for their respective setting. We show that there are instances on directed trees for which no global-priority schedule can guarantee a better limit than $GPRIO(I)$. Also, we present instances on directed trees for which no edge-priority schedule can guarantee a better limit than $\frac{5}{4}c + D_i - 1$ for each task τ_i (in comparison, $EPRIO(I)$ guarantees a limit of $\frac{3}{2}c + D_i - 1$ for each task τ_i). Recall that for all our schedules we assumed that the underlying graph is a tree. We show for a slightly more general graph class that no edge-priority schedule can guarantee a better limit than $\Omega(c \cdot D)$ for every task. Finally, we show that

if the tasks have arbitrary period lengths we cannot guarantee any good limits either, not even when the graph is only a path.

First, we present our lower-bound instances for global-priority schedules on directed trees (which show that $GPRIO(I)$ is best possible).

Theorem 4.25. *For each period length $p \in \mathbb{N}$ and each congestion $c \in \mathbb{N}$ there exists a respective instance of the sporadic PPRP on directed trees for which no global-priority schedule can guarantee a better limit than $2c + D_i - 2$.*

Proof. Consider a star graph with one vertex v_r in the center with c ingoing arcs $E = \{e_1, ..., e_c\}$ and c outgoing arcs $E' = \{e'_1, ..., e'_c\}$. We introduce c^2 tasks. The path of each task uses one of the arcs in E and one of the arcs in E' such that no two tasks use the same two arcs. Denote by I the resulting instance. We claim that for I no global-priority schedule can guarantee a better limit than $2c = 2c + D_i - 2$.

Assume we have a global-priority schedule for I. Then, there must be a task τ which has the lowest priority. Assume w. l. o. g. that τ uses the arcs e_1 and e'_1. Denote by E_1 and E'_1 the other tasks which use e_1 and e'_1. We consider a realization in which only τ and the tasks in $E_1 \cup E'_1$ create packets. Let M_τ be a packet created by τ. Since we are in the sporadic setting it can happen that M_τ is delayed once by a packet of every task in E_1 and once by a packet of every task in E'_1. Thus, in this case M_τ needs $2c$ steps. Hence, the schedule cannot guarantee a better limit than $2c + D_i - 2$. □

Note that Theorem 4.25 implies the stated lower bound also for global-priority schedules on bidirected trees (since directed trees are a subclass of bidirected trees). Also note that in the instance described in Theorem 4.25 there can be a task which is delayed $2c - 2$ times by the schedule $GPRIO(I)$, even in the strict periodic setting (due to bad tie-breaking decisions).

Next, we present our lower-bound instances for edge-priority schedules on directed trees. They show that $EPRIO(I)$ is almost best possible.

Theorem 4.26. *For each period length p and each even congestion c there exist instances of the sporadic PPRP on directed trees for which no edge-priority schedule can guarantee a better limit than $\frac{5}{4}c + D_i - 1$.*

Proof. Consider a very long path along which $c/2$ tasks send their packets. Denote by $\hat{I}$ these tasks. Assume on the contrary that we have a schedule S which guarantees a better limit than $\frac{5}{4}c + D_i - 1$ for each task τ_i.

At every other vertex with degree 2 we introduce a gadget. Figure 4.7 shows the gadget: We have a star with the vertices v_1, v_2, v_3, v_4, v_r and the arcs $(v_1, v_r), (v_2, v_r), (v_r, v_3), (v_r, v_4)$. We introduce $c/2$ tasks with the path (v_2, v_r, v_3), $c/2$ tasks with the path (v_1, v_r, v_3), and $c/2$ tasks with the path (v_1, v_r, v_4). We denote these tasks by I_1, I_2, and I_3, respectively. We introduce $\frac{5}{8}c^2 + 1$ of these gadgets. For every task $\hat{\tau} \in \hat{I}$ there can be at most $\frac{5}{4}c$ gadgets in which a task $\tau \notin \hat{I}$ has a higher priority than $\hat{\tau}$. Since $|\hat{I}| = c/2$ by the pigeonhole principle there must be a gadget in which every task in $\hat{I}$ has a

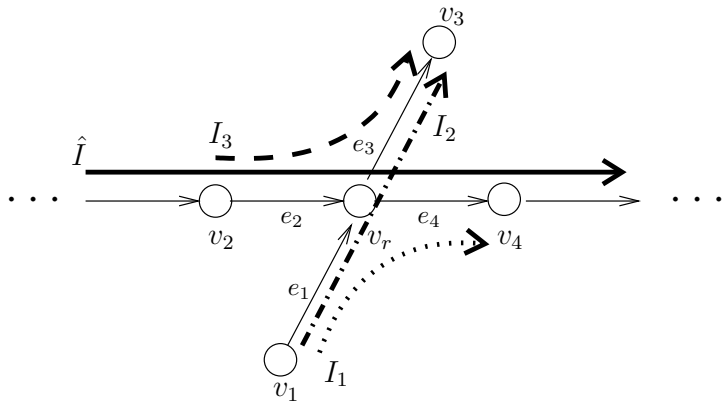


Figure 4.7: The gadget used in the proof of Theorem 4.26.

higher priority than each task not in $\hat{I}$ which uses this gadget. We prove in the sequel that depending on the release time of the packets in this gadget a packet created by one of the tasks in $I_1 \cup I_2 \cup I_3$ needs at least $\frac{5}{4}c + 1$ steps.

First, consider the arc $e_1 = (v_1, v_r)$. Denote by $\tau^{(1)} \in I_1$ the task in I_1 which has the lowest priority among all tasks in I_1. Denote by x the number of tasks in I_2 which have a higher priority than $\tau^{(1)}$ and by y the number of tasks in I_2 which have a lower priority than $\tau^{(1)}$. Now consider the arc $e_3 = (v_r, v_3)$. Let $\tau^{(3)} \in I_3$ be the task in I_3 with lowest priority. We define x' to be the number of tasks in I_2 with higher priority than $\tau^{(3)}$ and by y' the number of tasks in I_2 with lower priority than $\tau^{(3)}$ on e_3. Observe that $x + y = c/2 = |I_2| = x' + y'$.

Now we distinguish several cases: First assume that $x \geq y$. In particular, this implies that $x \geq c/4$. Then there is a realization in which a packet $M^{(1)}$ of $\tau^{(1)}$ is delayed once by a packet from each task in $\hat{I} \cup I_1 \setminus \{\tau^{(1)}\}$ and once by each of the x tasks in I_2 which have a higher priority on e_1 than $\tau^{(1)}$. Thus, $M^{(1)}$ is delayed $\frac{c}{2} + \frac{c}{2} + x - 1 \geq \frac{5}{4}c - 1$ times. With a similar reasoning we can handle the case that $x' \geq y'$. So now assume that $x < y$ and $x' < y'$.

If $x' < y$ then there are tasks $I_2' \subseteq I_2$ which have a lower priority than $\tau^{(1)}$ on e_1 and a lower priority than $\tau^{(3)}$ on e_3. Note that $|I_2'| \geq y - x'$. Let task $\tau'^{(2)} \in I_2'$ be the task which has the lowest priority on e_1 among all tasks in I_2'. There is a realization in which a packet of $\tau'^{(2)}$ is delayed on e_1 once by each of the x tasks in I_2 with higher priority than $\tau^{(1)}$, once by each task in I_1, and once by each task in $I_2' \setminus \{\tau'^{(2)}\}$, and on e_3 once by each task in I_3. In total, this gives

$$\begin{aligned} x + c/2 + |I_2'| - 1 + c/2 &\geq c + x + y - x' - 1 \\ &= \frac{3}{2}c - x' - 1 \\ &\geq \frac{5}{4}c - 1 \end{aligned}$$

delays where the last inequality follows since $x' + y' = c/2$ and $x' < y'$ and hence $x' < c/4$. We can handle the case that $x < y'$ similarly. So now assume

that $x \geq y'$ and $x' \geq y$. However, this is a contradiction since $x' \geq y > x \geq y' > x'$. □

For our schedules $GPRIO(I)$ and $EPRIO(I)$ we required a bound on the congestion of $c \leq p/3$ and $c \leq 2p/5$, respectively. This is stricter than the general condition $c \leq p$. Now we show that *any* global-priority or edge-priority schedule on a directed tree needs to require such a bound since otherwise it cannot guarantee any finite limit for each task.

Proposition 4.27. *Let $\alpha > 0$. There is an instance $I_\alpha = (G, T, p)$ of the sporadic PPRP on a directed tree G with $c = p$ such that for any edge-priority schedule ES there is a task $\tau \in T$ for which ES cannot guarantee a limit k with $k \leq \alpha \cdot c + D_i$.*

Proof. We describe how to construct the instance I_α. We introduce sets of tasks T_i. Each set T_i contains $p/2$ tasks which all have the same path. Let T_1 be the first of these sets. All tasks in T_1 are scheduled on a very long path P_1. For each arc e on P_1, there is a set of $p/2$ tasks T_e which use the arc e but no other arc on P_1. We assume that P_1 is so long that there must be an arc e such that all tasks in T_1 have higher priority than the tasks in T_e (otherwise we could define creation times for the packets such that a packet from a task in T_1 is delayed $\alpha \cdot c$ times). We define $T_2 := T_e$. We extend the path P_2 of the tasks in T_2 (actually, we do the following procedure for each arc $\bar{e} \in P_1$ but for our analysis later only the arc e will be important). For each arc e' on P_2 we introduce $c/2$ tasks $T_{e'}$ which use e' but no other arc no P_2. Like above, we define P_2 to be so long that there must be an arc $e' \in P_2$ such that the tasks T_2 all have a higher priority than the tasks $T_{e'}$. By continuing inductively, for each edge-priority schedule ES for I_α we obtain sets $T_1, ..., T_\ell$ with $c/2$ tasks each such that for each i we have that the tasks $T_i \cup T_{i+1}$ share an arc on which each tasks in T_i has a higher priority than any task in T_{i+1}. In the sequel, we show that there are creation times for the packets such in a set T_j there is a task whose packets are delayed at least αc times.

We denote by a *pack* of packets a set of $p/2$ packets which were all created at the same time by the tasks in a set T_i. We let T_1 create a pack $\mathcal{M}_1^{(1)}$ and we let T_2 create two packs $\mathcal{M}_1^{(2)}$ and $\mathcal{M}_2^{(2)}$. We define their creation times such that the pack $\mathcal{M}_1^{(1)}$ delays the pack $\mathcal{M}_1^{(2)}$ and thus the packets of packs $\mathcal{M}_1^{(2)}$ and $\mathcal{M}_2^{(2)}$ move in a row (after the delay). For induction, assume that T_i has created i packs $\mathcal{M}_1^{(i)}, ..., \mathcal{M}_i^{(i)}$ whose packets move in a row. We let T_{i+1} create $i+1$ packs $\mathcal{M}_1^{(i+1)}, ..., \mathcal{M}_{i+1}^{(i+1)}$ which collide with the packs $\mathcal{M}_1^{(i)}, ..., \mathcal{M}_i^{(i)}$. Since the tasks in T_i all have a higher priority than the tasks in T_{i+1} this results in $\mathcal{M}_1^{(i+1)}, ..., \mathcal{M}_{i+1}^{(i+1)}$ all moving in a row after the delay. By continuing inductively, we obtain that the tasks in $T_{2\alpha+1}$ create $2\alpha + 1$ packs (all moving in a row) with $(2\alpha + 1) \cdot \frac{c}{2}$ packets in total. Thus, by choosing the creation times appropriately, we can enforce that a packet M created by a task in $T_{2\alpha+2}$ is delayed by all these packets. Hence, M is delayed $(2\alpha + 1) \cdot \frac{c}{2} > \alpha \cdot c$ times. This implies that ES cannot guarantee a limit of $\alpha \cdot c + D_{2\alpha+2}$ for $T_{2\alpha+2}$. □

Note that Proposition 4.27 implies the same statement for global-priority schedules (since they are special cases of edge-priority schedules).

In all our algorithms for priority schedules we considered only trees as the underlying network topology. This might look very restrictive at first glance. However, in the following theorem we show that one cannot prove any similar results for more general graph classes, not even for the very simple class of chain graphs. We define a *chain graph* to be a path with possibly parallel edges. We assume that in this setting the paths of the tasks are given as part of the input (since they are not uniquely defined like in trees).

Theorem 4.28. *For any $D > 0$ and any $c, p \in \mathbb{N}$ there is an instance $I = (G, T, p)$ of the sporadic PPRP with congestion c and with given paths on a chain graph G such that*

- *$D_i = D$ for each task $\tau_i \in T$,*
- *all tasks have the same start and the same destination vertex, and*
- *for every edge-priority schedule there is at least one task $\tau \in T$ which has a limit in $\Omega(c \cdot D)$.*

Proof. The instance I is constructed as follows. We introduce c^D tasks. Intuitively, they can be visualized in a D-dimensional hypercube with edge-length c. We identify each task $\tau(k_0, k_1, ..., k_{D-1})$ by integers $k_i \in \{0, 1, ..., c-1\}$ for $i \in \{0, ..., D-1\}$. Our graph is a chain graph with vertices $v_0, ..., v_D$ and c^{D-1} parallel edges connecting each pair of adjacent vertices v_j, v_{j+1}. The paths are chosen such that on the connection between the vertices v_j, v_{j+1} two tasks $\tau(k_0, k_1, ..., k_{D-1}), \tau(k'_0, k'_1, ..., k'_{D-1})$ share an edge if and only if $k_i = k'_i$ for all $i \in \{0, 1, ..., j-1, j+1, ..., D-1\}$. We note that each edge is used by exactly c tasks.

Assume that there is an edge-priority schedule for I. For each task $\tau \in T$ let $d_{\tau,\ell}$ denote the number of tasks which use the same edge as τ between v_ℓ and $v_{\ell+1}$ and which have a higher priority on that edge. We calculate that

$$\sum_{\ell=0}^{D-1} \sum_{\tau \in T} d_{\tau,\ell} = Dc^{D-1} \sum_{k=0}^{c-1} k = Dc^D \frac{c-1}{2}.$$

Hence, there must be one task $\tau^* \in T$ such that $\sum_{\ell=0}^{D-1} d_{\tau,\ell} \geq D\frac{c-1}{2}$. We consider the following realization: The task τ^* creates packets and any other task τ' creates packets if and only if it shares an edge with τ^* and on this edge τ' has a higher priority than τ^*. We call the latter tasks the *blocking tasks*. By definition of I we note that if two blocking tasks share an edge then this edge is used by τ^* as well. In the realization we can find suitable timesteps for the creation of the packets such that a packet created by τ^* is delayed $D\frac{c-1}{2}$ times. Hence, the limit guaranteed for τ^* has to be at least $D\frac{c-1}{2} \in \Omega(c \cdot D)$. □

Finally, we prove that if the tasks have arbitrary period lengths p_i, in general we cannot obtain schedules guaranteeing good limits for all tasks like

in *EPRIO*(I). In an instance with arbitrary period lengths, each task τ_i has a period length p_i and we assume that in each time interval of length p_i the task τ_i creates at most one packet. The rest of the problem definition remains unchanged.

Theorem 4.29. *For every $\alpha, \varepsilon > 0$ there is an instance $I = (G, T)$ of the sporadic PPRP with arbitrary period lengths on a directed path such that for any edge-periodic schedule there is a task $\tau_i \in T$ whose limit is at least $\alpha \cdot p_i + D_i$. Furthermore, for each edge e it holds that $\sum_{\tau_i \in T_e} \frac{1}{p_i} \leq \frac{1}{\alpha} + \varepsilon$.*

Proof. We define $k := \frac{\alpha^5}{\varepsilon} + 1$ and assume w. l. o. g. that ε is chosen such that k is an integer. We define G to be a path with k vertices $v_0, v_1, ..., v_k$. There are k *big tasks* τ_i $(i = 0, ..., k-1)$ with start vertex $s_i = v_i$, destination vertex $t_i = v_{i+1}$, and period length $p_i = \alpha$. Also, we define a set of α^2 identical *small tasks*, all with start vertex $s_j = v_0$, destination vertex $t_j = v_k$, and period length $p_j = \frac{\alpha^2}{\varepsilon}$ (with $j = k, ..., k+\alpha^2-1$). We claim that for each edge-periodic schedule there is a task $\tau_i \in T$ whose limit is greater than $\alpha \cdot p_i + D_i$. Assume on the contrary that there is an edge-periodic schedule where this is not true. In this schedule, for each small task $\tau_j \in T$ there can be at most $\alpha \cdot \frac{\alpha^2}{\varepsilon}$ edges on which the respective big task has a higher priority than τ_j. Since $k = \frac{\alpha^5}{\varepsilon} + 1 = \alpha \cdot \frac{\alpha^2}{\varepsilon} \cdot \alpha^2 + 1$ and there are α^2 small tasks, there must be one big task τ_i which has a lower priority than all the small tasks on its edge. Hence, the limit of τ_i is at least $\alpha^2 + 1 = \alpha \cdot p_i + D_i$. Finally, for each edge e we have that $\sum_{\tau_i \in T_e} \frac{1}{p_i} = \frac{1}{\alpha} + \alpha^2 \cdot \frac{\varepsilon}{\alpha^2} = \frac{1}{\alpha} + \varepsilon$. □

Note that the value $\sum_{\tau_i \in T_e} \frac{1}{p_i}$ in the theorem can be understood as a generalized notion of congestion. Hence, no bound on this generalized congestion could help to guarantee good limits for all tasks.

4.4.2 Strict Periodic Setting

In this section we study priority schedules for the strict periodic packet routing problem. Since here each task creates a new packet *exactly* at timesteps $t = i \cdot p$ for every $i \in \mathbb{N}_0$, we have more control than in the sporadic setting. We use this control to weaken the bounds on the congestion which we require for *GPRIO*(I) and *EPRIO*(I) to guarantee the respective limits. In particular, we show that in the strict periodic setting the two schedules guarantee the bounds stated in Theorems 4.21 and 4.22 already if $c \leq p/2$ (and not only if $c \leq p/3$ or $c \leq 2p/5$, respectively).

First, we give our theorems for *GPRIO*(I) and *EPRIO*(I) in the strict periodic setting.

Theorem 4.30. *Let I be an instance of the strict PPRP on a bidirected tree with $c \leq p/2$. Then GPRIO(I) guarantees a limit of $2c + D_i - 2$ for each task τ_i.*

Theorem 4.31. *Let I be an instance of the strict PPRP on a directed tree with $c \leq p/2$. Then EPRIO(I) guarantees a limit of $\frac{3}{2}c + D_i - 1$ for each task τ_i.*

Before we can prove the two theorems, we need to study the behavior of packets on their way up in $GPRIO(I)$ and $EPRIO(I)$. First, we show in Lemma 4.32 that if an up-arc e is used by a packet M at a time t, then at time $t+p$ the arc e is used by a packet M' whose priority is not lower than the priority of M. This will allow us to prove a monotonicity property in the subsequent Lemma 4.33: We will show that any packet created by a task τ will suffer at least as much delay on its way up as any packet which was created by τ before. This additional structure in the delay of packets allows us to prove Theorems 4.30 and 4.31 afterwards.

Lemma 4.32. *Let I and $\bar{I}$ be instances of the strict PPRP on a bidirected tree and a directed tree, respectively. If in $GPRIO(I)$ or $EPRIO(\bar{I})$ an up-arc e is used by a packet created by a task τ_i at time t then at time $t+p$ it is used by a packet created by a task τ_j with $\tau_j \preceq \tau_i$.*

Proof. In order to prove the claim of the lemma, we show the following even stronger statement. Fix an arc e and a point in time t. Let $M_1 \preceq M_2 \preceq ... \preceq M_k$ be the packets which wait to use e at time t (we use the short notation $M \preceq M'$ for two packets M, M' if for their corresponding tasks τ, τ' it holds that $\tau \preceq \tau'$). Denote by $M'_1 \preceq M'_2 \preceq ... \preceq M'_{k'}$ the packets which wait to use e at time $t+p$. We show that $k' \geq k$ and $M'_i \preceq M_i$ for all i with $1 \leq i \leq k$. Observe that this statement implies the statement of the lemma since M_1 and M'_1 traverse e at times t and $t+p$, respectively.

W. l. o. g. we assume that the first p arcs of the path of each task τ are only used by τ. Note that on these arcs the claim holds trivially. In the sequel, we show the claim for the other arcs.

For an up-arc $e = (u, v)$ we define $d(e) := d(u)$. We prove the claim by induction over $d(e)$. If u is a leaf then the claim is clear. So now assume that the claim holds for all up-arcs e' with $d(e') \geq k$. Consider an up-arc e with $d(e) = k - 1$.

We show the claim by induction over t. The arc e is not used before time $t = p$ (since the first p arcs of each path are only used by one task and we now show the claim for the remaining arcs). Thus, the claim trivially holds for all timesteps $t < p$. So now assume that there is a value t^* such that the claim holds for e for all timesteps $t \leq t^*$. We need to show that the claim then also holds for timestep $t = t^* + 1$. We compare the waiting packets at timestep $t^* + 1$ with the corresponding packets at the timestep $t^* + p + 1$. As above, denote by $M_1 \preceq M_2 \preceq ... \preceq M_k$ and $M'_1 \preceq M'_2 \preceq ... \preceq M'_{k'}$ the packets waiting for using e at times t^* and $t^* + p$, respectively. Then, the packets M_1 and M'_1, respectively, traverse e.

Let $e_1, ..., e_\ell$ denote the ingoing arcs of u. Denote by $\bar{M}_1 \preceq \bar{M}_2 \preceq ... \preceq \bar{M}_m$ and $\bar{M}'_1 \preceq \bar{M}'_2 \preceq ... \preceq \bar{M}'_{m'}$ the packets which traverse $e_1, ..., e_\ell$ at times t^* and $t^* + p$, respectively, and which need to use e. Note that ℓ does not necessarily equal m or m' since it could be that some arcs e_i are not used by any packet at time t^*. Since we assumed the claim to be true for all arcs e' with $d(e') \geq k$ we have that $m' \geq m$ and $\bar{M}'_i \preceq \bar{M}_i$ for all i with $1 \leq i \leq m$. Recall that we prioritized the tasks by their depth and that here we consider only up-arcs. In

particular, this implies that if $\bar{M}_i$ needs to use e then so does $\bar{M}'_i$. Thus, at times $t^* + 1$ and $t^* + p + 1$ the packets $M_2, ..., M_k, \bar{M}_1, ..., \bar{M}_m$ and the packets $M'_2, ..., M'_{k'}, \bar{M}_1, ..., \bar{M}_{m'}$ wait to use e, respectively. From the above it follows that $M'_i \preceq M_i$ and $\bar{M}'_i \preceq \bar{M}_i$ for the respective values i. Hence, there is a one-one map f which maps each packet M waiting at time $t^* + 1$ to a packet $f(M)$ waiting at time $t^* + p + 1$ such that $f(M) \preceq M$. Therefore, if we consider the packets waiting at time $t^* + 1$ and one by one exchange each packet M by the packet $f(M)$, we observe that the claim holds after each exchange. This finishes the inductive step. □

Next, we show the monotonicity property for the delay of the packets which are created by the same task.

Lemma 4.33. *Let I and $\bar{I}$ be instances of the strict PPRP on a bidirected tree and a directed tree, respectively. If in GPRIO(I) or EPRIO($\bar{I}$) a packet M created by a task τ_i is delayed k times before traversing an up-arc $e \in P_i$ then each packet M' created by τ_i after M is delayed at least k times before traversing e.*

Proof. We show the claim by induction over the arcs of $P_i = (e_1, e_2, ..., e_m)$. In order to simplify the proof we assume w. l. o. g. that e_1 is used only by τ_i. Thus, for e_1 the claim is obvious. So now assume that the claim holds for all arcs $e_1, ..., e_\ell$. We need to show that it also holds for $e_{\ell+1}$. Consider the two packets M and M' as defined in the lemma statement. Assume that M was delayed k_ℓ times before traversing e_ℓ and $k_{\ell+1}$ times before traversing $e_{\ell+1}$. This implies that during the time interval $[j \cdot p + k_\ell + \ell, j \cdot p + k_{\ell+1} + \ell)$ the arc $e_{\ell+1}$ is used by packets created by tasks with higher priority than τ_i. Due to Lemma 4.32 this implies that the same statement holds for any interval $[j' \cdot p + k_\ell + \ell, j' \cdot p + k_{\ell+1} + \ell)$ with $j' \geq j$. From the induction hypothesis we conclude that M' is delayed at least k_ℓ times before traversing e_ℓ. We conclude that also M' is delayed at least $k_{\ell+1}$ times before traversing $e_{\ell+1}$. □

Now we can prove our bounds for *GPRIO*(I) and *EPRIO*(I) in the strict periodic setting, given that $c \leq p/2$. We begin with the bound for *GPRIO*(I).

Proof of Theorem 4.30: Consider a task τ_i with peak vertex v. From Lemma 4.33 we conclude that two packets created by τ_i have a minimum time distance of c when reaching v (without requiring a bound for c). With similar arguments as in Theorem 4.21 we can prove that then on the way away from v_r each packet is delayed no more than $c - 1$ times. The induction hypothesis is that on the way away from v_r two packets created by the same task have a minimum distance of $p - c \geq p/2$. Thus, in the strict periodic setting *GPRIO*(I) guarantees a limit of $2c + D_i - 2$ for each task τ_i if $c \leq p/2$. □

Similarly, we can prove the bound for *EPRIO*(I).

Proof of Theorem 4.31: Consider a task τ_i with peak vertex v. From Lemma 4.33 we conclude that two packets created by τ_i have a minimum time distance of c when reaching v (again without requiring a bound for c). We can argue that

then on the way down each packet is delayed no more than $c-1$ times and hence in the strict periodic setting $EPRIO(I)$ guarantees a limit of $\frac{3}{2}c + D_i - 1$ for each task τ_i if $c \leq p/2$. □

4.5 Imitation Theorems

In the previous sections we compared the power of template schedules and global-priority and edge-priority schedules. We did this by giving algorithms for computing schedules of the respective types and proving lower bounds.

In this section we follow a more direct approach to compare the capabilities of the two scheduling paradigms. We prove that any global-priority schedule on any graph can be *imitated* by a template schedule. Also, we show that any edge-priority schedule on a bidirected tree can be imitated by a template schedule. With "imitate" we mean that we can find a template schedule which guarantees the same limits for each task as the respective priority schedule in the strict periodic case and almost the same limits in the sporadic case. This shows that template schedules are at least as powerful as the respective priority schedules in the studies settings.

We prove these results by showing that in the strict-periodic setting global-priority schedules on general graphs and edge-priority schedules on bidirected trees behave periodically after a certain time (and thus operate like template schedules). This allows us to construct template schedules with the same periodic behavior. Finally, we show that there are edge-priority schedules on cycle graphs in the strict-periodic setting which never behave periodically. Thus, they cannot directly be imitated by template schedules.

4.5.1 Template Schedules vs. Global-Priority Schedules

First, we show that on general graphs template schedules can imitate global-priority schedule. Given an instance I of the strict periodic packet routing problem on an arbitrary graph. Let S be a global-priority schedule for I. We show how to construct a template schedule S_t which imitates the schedule S. First, we prove that S behaves periodically after a certain time.

Lemma 4.34. *Let I be an instance of the strict PPRP on an arbitrary graph. Let S be a global-priority schedule for I. For each task τ_i and each arc $e \in P_i$ there is a timestep k_i and an offset $a_{e,i}$ such that for all timesteps $k \geq k_i$ we have that*

- *S transports a packet of τ_i over e if $k \equiv a_{e,i} \bmod p$ and*
- *S does not transport a packet of τ_i over e if $k \not\equiv a_{e,i} \bmod p$.*

Proof. We prove the claim by induction over the tasks. Assume the tasks are ordered such that $\tau_1 \prec_S \tau_2 \prec_S \ldots \prec_S \tau_{|T|}$. W. l. o. g. we assume that the first arc of the path of each task is only used by this one task. For the task τ_1 we

can easily find a value k_1 (e. g., $k_1 := |P_1|$) and suitable offsets such that the claim of the lemma holds.

We assume for the induction hypothesis that for all tasks $\tau_1, ..., \tau_m$ we have values k_i and suitable time offsets with the claimed properties. Consider the task τ_{m+1}. We show the claim for τ_{m+1} by another induction over the arcs of $P_{m+1} = (e_1, ..., e_s)$. For e_1 the claim is trivial (since by assumption only τ_{m+1} uses this arc). Assume inductively that we have found a value k^r_{m+1} such that for all e_j with $j \leq r$ we have found offsets as stated in the lemma which hold for all timesteps k with $k \geq k^r_{m+1}$.

Consider the arc e_{r+1}. We define $\tilde{k} := \max\left\{k_1, ..., k_m, k^r_{m+1}\right\}$. First assume that there are $p-1$ other tasks which use e_{r+1} and which have a higher priority than τ_{m+1} in S. Then we define $k^{r+1}_{m+1} := \tilde{k}$ and $a_{e_{r+1},m+1}$ is defined such that $a_{e_{r+1},m+1} \neq a_{e_{r+1},i'}$ for all tasks $\tau_{i'}$ which also use e_{r+1}. Note that in this case it could be that at time $\tilde{k}$ a certain number of packets created by τ_{m+1} wait for using e_{r+1}. Then, this number will not decrease in the long run since in each time interval of length p one packet of τ_{m+1} will traverse e_{r+1} and a new one will arrive.

Now assume that on e_{r+1} there are at most $p-2$ tasks which have a higher priority than τ_{m+1}. Then, from time $\tilde{k}$ on we know from the induction hypothesis that exactly every p timesteps a new packet from τ_{m+1} arrives and needs to use e_{r+1}. However, there might be some other packets created by τ_{m+1} waiting to use e_{r+1} which could not use e_{r+1} yet. During the time interval $\left[0, \tilde{k}\right]$ exactly $\left\lfloor \frac{\tilde{k}}{p} \right\rfloor$ packets were created by τ_{m+1}. At least two packets created by τ_{m+1} can use e_{r+1} in each time interval $[\tilde{k} + v \cdot p, \tilde{k} + (v+1) \cdot p - 1]$, for $v \in \mathbb{N}$. Thus, if we define $k^{r+1}_{m+1} := 2\tilde{k}$ we can find a suitable offset $a_{e_{r+1},m+1}$ which satisfies the statement in the lemma. Finally, we define $k_{m+1} := k^s_{m+1}$. □

Now we can design a template schedule S_t which emulates S and thus guarantees the same limits as S.

Theorem 4.35. *Let I be an instance of the strict PPRP on an arbitrary graph. For each global-priority schedule S for I there is a template schedule S_t which guarantees the same limit as S for each task.*

Proof. We construct the template schedule S_t from the offsets which are given by Lemma 4.34. Consider a task τ_i. The lemma implies that all packets created by τ_i after timestep k_i cannot reach their respective destination faster in S than in S_t. □

Corollary 4.36. *Let $I = (G, T, p)$ be an instance of the sporadic PPRP on an arbitrary graph G. Let S be a global-priority schedule for I which guarantees a limit of k_i for each task τ_i. There is a template schedule which guarantees a limit of $k_i + p$ for each task τ_i.*

Note that for the sporadic setting we have the additive value p in the above corollary. This comes from the fact that in the sporadic setting a newly created

packet might just "miss" its timeslot to use its first edge in the template schedule. However, in a priority schedule the packet might not need to wait at all.

4.5.2 Template Schedules vs. Edge-Priority Schedules

We prove that on bidirected trees template schedules can imitate edge-priority schedules. After that, we show that on more general graph classes than trees edge-priority schedules do not necessarily behave periodically. Hence, they cannot directly be emulated by template schedules.

Given an instance I of the strict periodic periodic packet routing problem on a bidirected tree G. Let S be an edge-priority schedule for I. We show how to construct a template schedule S_t which guarantees the same limits for the tasks as S. As in Lemma 4.34 we show that S behaves periodically after a certain time.

Since our tree is bidirected here we will interpret each connection as two directed arcs. We assign each arc a *level*. For an arc $e = (u, v)$ which is oriented *away from* v_r we define level(e) to be the distance between v_r and v. For an arc $e = (u, v)$ which is oriented *towards* v_r we define level(e) to be the distance between u and v_r multiplied by -1. Figure 4.8 shows a sketch which yields some intuition for the definition. Note that if an arc e is the antiparallel arc to an arc e' then level$(e) = -$level(e'). Moreover, there is no arc e with level$(e) = 0$.

Lemma 4.37. *Let I be an instance of the strict PPRP on a bidirected tree. Let S be an edge-priority schedule for I. For each arc $e \in P_i$ and each task τ_i there is a timestep k_e and an offset $a_{e,i}$ such that for all timesteps $k \geq k_e$ we have that*

- *S transports a packet of τ_i over e if $k \equiv a_{e,i} \bmod p$ and*
- *S does not transports a packet of τ_i over e if $k \not\equiv a_{e,i} \bmod p$.*

Proof. We assume w. l. o. g. that the first arc on the path of each task is used only by this one task. We prove the lemma by induction over the levels of the arcs, starting with the arcs in the level with lowest index. Let e be such an arc. Due to our assumption above there is exactly one task τ_i which uses e and thus we can choose $k_e = 0$ and $a_{e,i} = 0$.

Now assume that by induction the claim is true for all arcs e with level$(e) \leq m$. Now let $e = (u, v)$ be an arc with level$(e) = m + 1$. If u is a leaf then there can be at most one task using e. The claim is then shown as in the base case. So now assume that u is not a leaf. Let $e_1, ..., e_s$ denote all ingoing arcs of u. Due to the induction hypothesis the claim is true for each of these arcs. Let $\tilde{k} := \max\{k_{e_1}, ..., k_{e_s}\}$. Assume w. l. o. g. that e is used by the tasks $T_e := \{\tau_1, ..., \tau_r\}$ with the priority order $\tau_1 \prec \tau_2 \prec ... \prec \tau_r$. We show the claim by induction over these tasks. Since τ_1 has highest priority among the tasks in T_e there is an offset $a_{e,1}$ with the properties stated above for all k with $k \geq \tilde{k} =: k_1^e$. Assume for induction that we have such offsets for all tasks $\tau_i \in T_e$ with $i \leq m$ such that the claim holds for all timesteps $k \geq k_m^e$. Now consider the task τ_{m+1}.

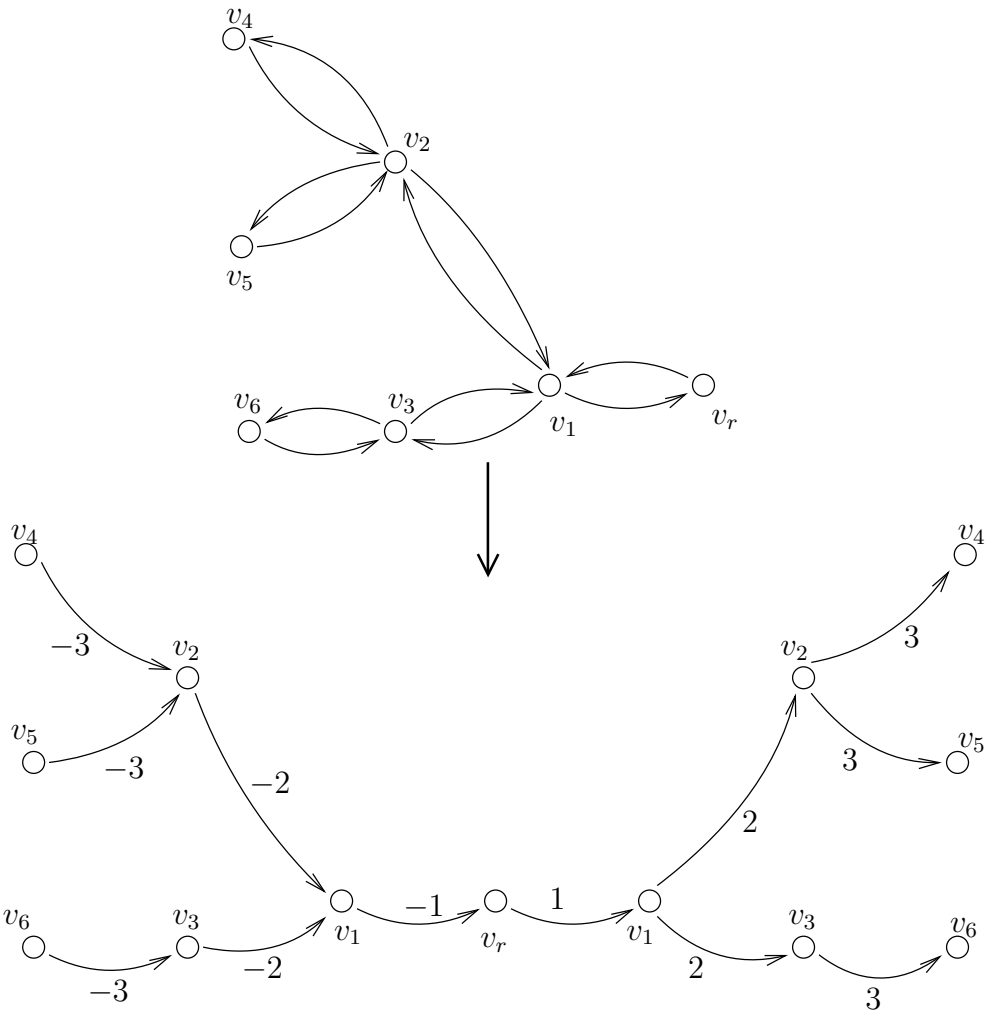


Figure 4.8: The lower figure depicts the levels of the arcs of the tree above. Note that in the lower figure all vertices (but v_r) appear twice.

If $m+1 = p$ then all other tasks in T_e have a higher priority and there is only one offset $a_{e,m+1} \in \{0, ..., p-1\}$ left for τ_{m+1}. Thus, the lemma statement holds with $a_{e,m+1}$ for all timesteps $k \geq k^e_{m+1} := k^e_m$. Now assume that $m + 1 < p$. Then, in each interval $[k^e_m + i \cdot p, k^e_m + (i+1) \cdot p - 1]$ (for all $i \in \mathbb{N}$) at least two packets created by τ_{m+1} can use e. In the first k^e_m timesteps $\left\lfloor \frac{k^e_m}{p} \right\rfloor$ packets were created by τ_{m+1}. With the induction hypothesis we conclude that there is an offset $a_{e,m+1}$ such that the lemma holds for all timesteps $k \geq k^e_{m+1} := 2k^e_m$. Finally, we define $k_e := \max_i k^e_i$. □

Theorem 4.38. *Let I be an instance of the strict PPRP on a bidirected tree. For each edge-priority schedule S for I there is a template schedule S_t which guarantees the same limit for each task as S.*

Proof. Follows from Lemma 4.37 (similar proof as for Theorem 4.35). □

Corollary 4.39. *Let $I = (G, T, p)$ be an instance of the sporadic PPRP on a bidirected tree G. Let S be an edge-priority schedule for I which guarantees a limit of k_i for each task τ_i. There is a template schedule which guarantees a limit of $k_i + p$ for each task τ_i.*

Now we investigate whether edge-priority schedules on more general graph classes can be imitated by template schedules. In the following theorem we show that there are edge-priority schedules for the strict PPRP on a cycle graph which do not behave periodically after any point in time. Hence, we cannot prove a lemma similar to Lemma 4.34 or Lemma 4.37 for this setting. In particular, this implies that the schedule cannot be directly emulated by a template schedule.

Theorem 4.40. *There is an instance I of the strict periodic PPRP on a cycle graph and an edge-priority schedule S for I with the following property: there is no timestep t such that there is a template schedule for I which is identical to S after time t.*

Proof. The instance I is depicted in Figure 4.9: we have a cycle graph with four vertices v_1, v_2, v_3, v_4 and two tasks τ_1, τ_2 with $P_1 = (v_1, v_2, v_3, v_4)$ and $P_2 = (v_3, v_4, v_1, v_2)$. We define $p := 2$. In the schedule S the task τ_1 has a higher priority than τ_2 on the arc (v_3, v_4). The opposite prioritization holds on the arc (v_1, v_2).

Assume on the contrary that there is a time t such that there is a template schedule S_t which behaves identically to S after time t. Since $p = 2$ it suffices to describe S_t by specifying whether a task uses an arc at even or odd timesteps. We distinguish two cases: First assume that the arc (v_2, v_3) is used by τ_1 at odd timesteps. Since the construction is symmetric, this implies that the arc (v_4, v_1) is used by the task τ_2 at odd timesteps. Due to the definition of S this implies that τ_1 uses the arc (v_3, v_4) at even timesteps and thus, every packet of τ_2 is delayed once in v_3. However, this implies that τ_2 uses the arc (v_4, v_1) at even timesteps which is a contradiction. Now we assume that the arc (v_2, v_3) is used by τ_1 at even timesteps. By symmetry, this implies that (v_4, v_1) is used by τ_2 at

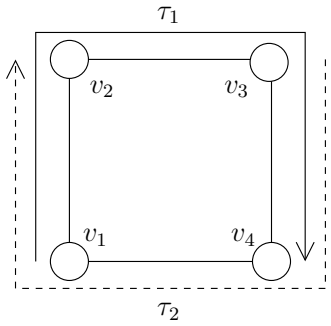


Figure 4.9: The instance I described in the proof of Theorem 4.40.

even timesteps. Hence, τ_2 uses (v_1, v_2) at odd timesteps. We conclude that τ_1 uses (v_2, v_3) at odd timesteps which is a contradiction. Thus, there can be no template schedule for I which is identical to S after any time t. □

4.6 Conclusion

In this chapter we studied the periodic packet routing problem. In particular, we compared priority schedules and template schedules. Template schedules have proven to be more powerful than priority schedules. Hence, for further research for proving better limits of periodic schedules, template schedules seem to be the right paradigm.

For directed trees, there are template schedules which guarantee a limit of $c + D_i - 1$. This is best possible. However, for bidirected trees it is not clear to us whether our bound of $2c - c\left(\frac{1}{2}\right)^{\lceil \mathrm{diam}(G)/2 \rceil - 1} + D_i - 1$ is already best possible or whether it can be improved further. Another interesting line of research would be template schedules with arbitrary period lengths on directed or bidirected trees. In contrast, recall that we showed that edge-priority schedules cannot guarantee a limit of the form $\alpha \cdot p_i + D_i$ for every task τ_i and any $\alpha > 0$, even on a directed path (where p_i equals the period length of τ_i). However, template schedules have this ability, due to the result by Andrews et al. [5]. Even for general graphs, they generalized the result by Leighton et al. [65] to the periodic packet routing problem, guaranteeing a limit of $O(p_i + D_i)$ for each task τ_i (expressed in our notation).

Our results presented in Chapter 2 give a bound $23.4(C + D)$ for the static setting. The bound of $O(p_i + D_i)$ in the periodic setting hides a very large constant. Therefore, it would be interesting to see what constants in front of $(p_i + D_i)$ can be obtained in the periodic case. Additionally, we believe that a lot of structural properties even for the static case are not yet fully understood. In particular, all known algorithms and bounds for general graphs highly rely on the Lovász Local Lemma. Since it is very general, it cannot exploit the entire structure of the underlying problem. Hence, we believe that there is space for further insights in the static setting which can also be very beneficial for the periodic setting.

Part II

Scheduling

Chapter 5

Increasing Speed Scheduling and Flow Scheduling

5.1 Introduction

Two corner stones of discrete optimization are network flows and scheduling. Flows model the movement of particles through a network. The well-studied flows over time even model varying flow rates over some time horizon. In scheduling, one is interested in computing a plan when certain actions (jobs) are carried out by certain entities (machines). Many problems have a flow as well as a scheduling aspect. For instance, a natural problem arising in computer networks is the packet routing problem studied in Part I *without* the given paths for the packets. There, one has to compute paths for the packets which can be understood as a static flow. Also, one needs to compute the actual routing schedule for the packets. There are also other examples in logistics where flows and scheduling interact. For instance, consider the container terminal of a modern harbor where containers are carried from the storage area to the loading cranes by automatically guided vehicles (see [45]).

Individually, network flows and scheduling are widely studied. However, the interaction of the two in a combined model is not well-understood yet. In particular, on general graphs the only known constant factor approximation for packet routing without given paths due to Srinivasan and Teo [101] decouples the scheduling from the routing aspect. It first computes the paths for the packets such that $C+D$ is small. Then, it invokes the algorithm by Leighton, Maggs, and Richa [66] to compute a schedule of length $O(C+D)$. In particular, in the scheduling part the algorithm does not take advantage of the freedom to choose the paths. Also, real-world applications like the mentioned harbor usually surpass the algorithmic means developed separately for scheduling and flows. In this chapter we take a first step towards joint combinatorial optimization of flows and schedules.

We study the following setting. Given a dynamic network, i. e., a network with static transit times and static capacities on the edges, a source s and a sink t. Also, consider a demand which is subdivided into k jobs, each characterized by its own flow demand and weight. We want to compute a flow over time which transports the demand from s to t. The objective is to minimize the sum of weighted completion times. The completion time of a job is the time when the entire demand of the job has reached the sink t. We call this the *flow scheduling problem*. We show a strong connection of the problem to the concept of *earliest-arrival-flows* (EAFs). Also, we show that the scheduling aspect of the problem reduces to the *increasing speed scheduling problem* (ISS) which to our knowledge has not been studied in its own right so far. In this problem, we are given a set of jobs with lengths and weights and a machine whose speed is not constant but might increase over time. The goal is to find a schedule which minimizes the sum of weighted completion time.

The results presented in this chapter are joint work with Sebastian Stiller [102].

5.1.1 Definitions

Before we can define the two problem studied in this chapter – the flow scheduling problem and the increasing speed scheduling problem – we need to define flows over time. First, we define single-commodity flows over time. Based on this, we then define the multicommodity flows over time which are important for this chapter.

Definition 5.1 (s-t-flow over time [99])**.** Let $G = (V, E)$ be a directed graph with a capacity u_e and a transit time τ_e for each arc $e \in E$. Let T be a time horizon. Let $f_e : [0, T) \to \mathbb{R}_{\geq 0}$ be a Lebesgue-integrable function for each arc $e \in E$. We call the family of functions f a *flow over time* if

- $f_e(\theta) = 0$ for all $\theta \geq T - \tau_e$,
- $f_e(\theta) \leq u_e$ for each arc $e \in E$ and all $\theta \in [0, T)$, and
- for the *excess* $\mathrm{ex}_f(v, \theta)$ *for node* v *at time* θ defined by

$$\mathrm{ex}_f(v, \theta) := \sum_{e \in \delta^-(v)} \int_0^{\theta - \tau_e} f_e(\xi) d\xi - \sum_{e \in \delta^+(v)} \int_0^{\theta} f_e(\xi) d\xi$$

  it holds that $\mathrm{ex}_f(v, \theta) \geq 0$ for each $v \in V \setminus \{s\}$ and all $\theta \in [0, T)$. Moreover, $\mathrm{ex}_f(v, T) = 0$ for each $v \in V \setminus \{s, t\}$.

Definition 5.2 (Multicommodity flow over time)**.** Let $G = (V, E)$ be a directed graph with a capacity u_e and a transit time τ_e for each arc $e \in E$. Let T be a time horizon. Assume we are given a set of commodities. Let $f_e^i : [0, T) \to \mathbb{R}_{\geq 0}$ be an s-t-flow over time for each commodity i. If additionally $\sum_i f_e^i(\theta) \leq u_e$ for each arc $e \in E$ and all $\theta \in [0, T)$ we call the family of functions f a *multicommodity flow over time*.

Now we are ready to define the flow scheduling problem.

Definition 5.3 (Flow scheduling problem)**.** Consider a directed graph G with two distinct nodes s and t. For each arc $e \in E$ we are given a static capacity u_e and a static transit time τ_e. Also, we are given a set of jobs J where each job $j \in J$ has a weight w_j and a demand ℓ_j. The goal is to find a multi-commodity s-t-flow over time with $|J|$ commodities such that $\sum_{j \in J} w_j C_j$ is minimized where C_j denotes the time when the flow value corresponding to commodity j has reached ℓ_j.

We call a solution for the flow scheduling problem a *flow schedule*. It turns out that a subproblem of the flow scheduling problem is the problem to schedule jobs on a single machine with possibly increasing speed to minimize $\sum_j w_j C_j$. This is the *increasing speed scheduling* problem as defined below.

Definition 5.4 (Increasing speed scheduling problem with release dates (ISS))**.** Given a machine M whose speed is given as a Lebesgue-integrable, weakly monotonically increasing function $s : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ and a set of jobs J. Each job $j \in J$ is characterized by demand ℓ_j, a weight w_j, and a release time r_j. The goal is to compute a schedule which minimizes the weighted sum of completion times. This means that we look for $|J|$ integrable indicator functions $\chi_j : \mathbb{R}_0^+ \to \{0, 1\}$ with $\chi_j(x) \cdot \chi_{j'}(x) = 0$ for all $x \in \mathbb{R}^+$ and $j \neq j'$ such that $C_j := \inf_{T \in \mathbb{R}^+} \{\int_{r_j}^T \chi_j(x) s(x) \mathrm{d}x \geq \ell_j\}$ exists for each $j \in J$ and $\sum_{j \in J} w_j C_j$ is minimized.

For the ISS problem without release dates, we assume that all $r_j = 0$. For a job (w_j, ℓ_j) we call w_j/ℓ_j its *Smith's ratio*. A schedule processing the jobs successively with non-increasing Smith's ratio is called a *Smith's rule algorithm*. An *efficient PTAS* (EPTAS) is a family of $(1+\varepsilon)$-algorithms for all $\varepsilon > 0$ with running time in $O\left(f(\varepsilon) \cdot poly(n)\right)$ for a function f depending only on ε.

5.1.2 Related Work

To the best of our knowledge *flow scheduling* has not been considered in the literature so far. It has some far resemblance to flow shop problems. There is a close relation to dynamic multi-commodity and earliest arrival flows (EAFs). An earliest arrival flow is a flow over time that has a maximal flow value (i.e., total excess at the sinks) at every point in time

For the single source, single sink case earliest arrival flows always exist [42], and can be found by pseudopolynomial successive shortest path algorithms [78, 107]. There are instances [111] where these algorithms take exponentially many steps in a binary encoded input. For multiple sinks and sources EAFs need not exist [15, 36]. In [55] a fully polynomial-time approximation scheme for the earliest arrival s-t-problem is given. These results have been extended in [15] to solve EAFs with multiple sources.

The solution of a flow scheduling instance is a multicommodity flow over time—where all commodities have a common source and a common sink—which optimizes an objective function which is unusual for flows, namely, the weighted sum of completion times. Whether a multicommodity flow over time with a

given time horizon exists is NP-hard even in the fractional case, and even for strongly restricted graph classes [50]. See also reference therein and [99] for a survey on flows over time in general.

The *increasing speed scheduling* problem with release dates clearly contains $1|r_i, pmtn|\sum w_jC_j$ as an NP-hard special case. For this problem, Goemans, Wein, and Williamson present a 1.47-approximation algorithm [46]. This is improved to 4/3 by Schulz and Sktuella [94]. Finally, Afrati et al. present an EPTAS [3]. For scheduling with arbitrary varying speed, in particular, when the machine stops, we can argue that the $1|pmtn|\sum w_jC_j$ problem is weakly NP-hard by a reduction from Partition similar to that in [62]. If the speed of the machine might increase *and* decrease but all release dates are equal, Epstein et al. [31] give a deterministic 4-approximation and a randomized e-approximation that does not even consider the actual machine when computing the ordering of the jobs. As mentioned above, if the machine runs constantly with unit speed then it is well-known that the Smith's rule algorithm is optimal [100].

Another closely related problem is scheduling with rejection (see [54] and references therein), i. e., jobs can be excluded from the schedule at a fixed penalty cost. For minimizing the weighted sum of completion times Engels et al. proved this problem to be weakly NP-hard for a single machine and arbitrary rejection costs and job weights [30] (reduction from Partition). Moreover, they showed that the case of unit weights and the case of unit lengths are polynomial time solvable. To the best of our knowledge, the case where rejection costs are proportional to job weights is open. This is equivalent to a special case of increasing speed scheduling, notably, when the machine has constant speed until time t, and infinite (or sufficiently large) speed after t.

5.1.3 Outline of the Chapter

First, in Section 5.2 we establish the connection between flow scheduling and increasing speed scheduling. We show that dynamic flows that are maximal for a given set of deadlines can be found in polynomial time (Theorem 5.7). Later, we will use such flows instead of earliest arrival flows (EAFs) which are maximal for *any* deadline. This technique is important for obtaining polynomial time algorithms since there is no polynomial encoding known for EAFs (and the function which gives the inflow-rate of the EAF into the sink can have exponentially many break-points [111]). In particular, we will use this to approximate EAFs.

Next, we extend the EPTAS of [3] for preemptive single machine scheduling with release dates to the case of machines with increasing speed and release dates. Together with Theorem 5.7 this yields an EPTAS for the flow scheduling problem. In Section 5.4 we study structural properties of the ISS problem. These properties allow us to give exact, polynomial time algorithms for the ISS problem in similar special cases as considered in [54] for scheduling with rejection. Moreover, we device a dynamic program in case the speed function is a step function with constantly many steps.

In Section 5.5 we show that Smith's rule is a $(\sqrt{3}+1)/2$-approximation algorithm for ISS. We prove this using a tight analysis: we constructively char-

acterize worst instances for the Smith's rule algorithm. In the final sections we study online algorithms and algorithms that have no knowledge of the speed function (blind algorithms). Note that Smith's rule is such a blind algorithm. For both cases we show a lower bound for the best achievable approximation factor. For the online case we present an algorithm with a competitive ratio of 2.

Finally, we conclude with Section 5.8 and address some open problems.

5.2 From Flows to Scheduling

We consider flows over time with single source and sink. For these it is known that an earliest arrival flow (EAF) exists [42], i. e., a flow over time that has a maximal flow value at every point in time. In particular, one can compute in pseudo-polynomial time its inflow rate into the sink s_{EAF} which is a non-decreasing, stepwise constant function of time. For some instances this function has (in the input size of the network) pseudo-polynomially many breakpoints [111].

Therefore, any (single source, single sink) flow scheduling problem has an optimal solution with the following structure. Let I be the smallest interval in time such that all flow arrives at the sink during I. In I the inflow rate is always strictly positive and the interval can be partitioned into k consecutive intervals $I_i = [T_i, T_{i+1})$ such that during each of these intervals all inflow to the sink belongs to the same job. Interpret the inflow rate to the sink as the speed of a machine. Then we can rephrase: In an instance without release dates, to minimize the weighted sum of completion times it is best to process the jobs without preemption in a certain order on the machine. See Figure 5.1 for a sketch.

An EAF is by definition maximal at all points in time. Thus, to solve the flow scheduling problem one may calculate an EAF and solve the ISS problem with the speed function given by the EAF. As for EAFs no strongly polynomial encoding is known (and seems unlikely to exist) this leads to over-complicated flow schedules. This pseudo-polynomial blow up is unnecessary for an optimal solution of the flow scheduling problem: The flow value in any optimal solution for a flow scheduling instance needs to equal that of an EAF only at the completion time of each job. So, assuming that the optimal order of the jobs is known one can find in strongly polynomial time an optimal flow schedule using a *multiple deadline flow*. An MDF is a flow over time which is maximal at a certain set of deadlines $\mathcal{T} = \{T_1, \dots, T_k\}$. Later, these deadlines will equal the completion times of the jobs. We give a formal definition of MDFs below.

Definition 5.5 (Multiple Deadline Flows (MDF))**.** Given an s-t-digraph $G = (V, E)$ with non-negative, constant transit times τ and capacities u on the arcs, and a finite set of deadlines $\mathcal{T} = \{T_1, \dots, T_k\}$. A s-t-flow over time for (G, τ, u) is called a *multiple deadline flow (MDF)*, if for $1 \leq i \leq k$ its value at time T_i is maximal among all feasible s-t-flows over time on (G, τ, u).

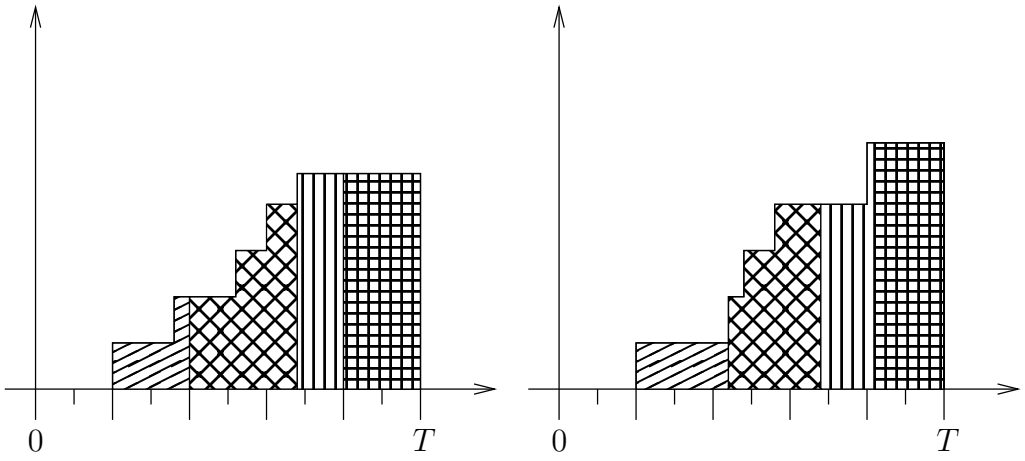


Figure 5.1: Left: the function s_{EAF} obtained from the inflow-rate into the sink of an EAF for some flow scheduling instance and a solution of the flow scheduling problem with four jobs. Right: the respective function for a flow which is not maximal at every point in time. Note that this function yields a worse objective function value even though the order of the jobs is the same: The first and the third job finish slightly later in the right solution than in the left solution.

Recall here that the *value* of an s-t-flow over time at a time T_i is the excess $\text{ex}_f(t, T_i)$ (inflow minus the outflow) at the sink t at time T_i.

We have seen that for our purposes EAFs – which are maximal at any point in time – can be unnecessarily complicated. MDFs are only maximal at a fixed set of points in time. This allows us to compute an MDF in time which is polynomial in the size of the input.

Theorem 5.6. *Given an s-t-digraph G with transit times τ and capacities u for the edges. Let $\mathcal{T}$ denote a set of deadlines. An MDF for G, τ, u, and $\mathcal{T}$ can be found in time polynomial in the input length.*

Proof. First we calculate in polynomial time the value of a maximal flow over time for each deadline T_i (e. g., with Ford-Fulkerson algorithm, see [99]). Then computing the MDF is equivalent to computing a quickest dynamic transshipment in the following graph: Replace the sink t by k sink nodes t_i each connected to t by its own arc of transit time $T_k - T_i$ and infinite capacity. The demand at each sink equals the maximal flow value for the corresponding deadline T_i minus the maximal flow value for T_{i-1}. A dynamic transshipment can be found in polynomial time [56]. □

The theorem above allows us to compute an optimal flow schedule if an optimal (or some fixed) order of the jobs is given. Assume that the order of the jobs is given by their indices. For each $i \in \{1, ..., k\}$ we calculate the minimal time horizon T_i to transport $\sum_{1 \le j \le i} \ell_j$ units of flow from the source to the sink. This can be done in polynomial time with a quickest flow computation [38]. Then we solve the MDF problem for the set of deadlines $\{T_i\}$. In the resulting flow we assign the first ℓ_1 flow units that reach the sink to the first job, the

following ℓ_2 flow units to the second job, and so on. This yields an optimal flow schedule for the fixed order. From this we immediately get:

Theorem 5.7. *There is a polynomial time algorithm for flow scheduling given that the optimal order of the jobs is known.*

To summarize: We are in a chicken and egg situation. Given an optimal order of the jobs, one can find an optimal flow schedule in strongly polynomial time. To compute the optimal order of the jobs, one needs to know the optimal (single-commodity) flow over time which corresponds to the optimal flow schedule. Still, one can use the pseudo-polynomially sized inflow rate of an EAF as the speed function of an instance of the ISS problem. Its optimal solutions give optimal orders of the jobs for the original flow scheduling problem. However, computing the actual EAF is computationally very costly.

There are two main approaches to overcome this dilemma: *Either* approximate the earliest arrival flow by a flow with a bounded number of speed changes and work with the machine given by this flow — *or* calculate an order which fulfills some approximation factor independent of the actual machine speed (blind algorithms). We will pursue both of these approaches, the first leading to an EPTAS and the second to an approximation factor of $\frac{\sqrt{3}+1}{2}$. Both approaches require to study the ISS problem only for step functions as speed functions. Nevertheless, we will treat the ISS problem as a problem in its own right and therefore allow any integrable function as speed function.

5.3 Polynomial Time Approximation Scheme

In this section we present an EPTAS for the increasing speed scheduling problem with arbitrary release dates, i. e., an approximation algorithm with approximation ratio $1 + \varepsilon$ and running time $O\left(f(\varepsilon) \cdot poly(n)\right)$ for a function f depending (exponentially) only on ε. We show later that this yields also an EPTAS for the flow scheduling problem.

Our strategy is the following. First, we derive a couple of properties which we can assume for the instance and the schedules without losing more than a factor of $1+O(\varepsilon)$ in the objective function in comparison to the optimum. Some of the techniques used here are borrowed from [3] and adapted to our problem. Then, we show how to compute the optimal schedule with these properties by a dynamic program.

Let $\varepsilon > 0$. We describe an algorithm which guarantees an approximation ratio of $1 + O(\varepsilon)$. By abuse of notation whenever we use the term $O(\varepsilon)$ we refer to a function bounded by $\ell \cdot \varepsilon$ for some positive ℓ. For technical reasons, we assume that $\varepsilon < 1$. W. l. o. g. we assume that at any time the speed of the machine is at least 1. We will use the notions "with $1+\varepsilon$ loss" and "with $1+O(\varepsilon)$ loss", meaning that by requiring a certain property for the schedule we might lose at most a factor of $1 + \varepsilon$ or $1 + O(\varepsilon)$, respectively, in the optimal objective function value. In each of the following lemmas we assume that all adjustments established by all previous lemmas have already been done.

We define $R(w)$ to be the timestep when the total work that the machine has done so far equals w. As short notation we use $R_x := R\left((1+\varepsilon)^x\right)$. Note that since we assume that the machine always runs with at least unit speed we have that $R_{x+1} \leq (1+\varepsilon) R_x$. We split the time scale into intervals of the form $I_x := [R_x, R_{x+1})$. In order to simplify notation we will use the notion I_x for the interval as well as for the work that the machine does within I_x. Note that $I_x = \varepsilon (1+\varepsilon)^x$.

In our first adjustment, we ensure that the demand of each job is a power of $1+\varepsilon$. This is a common simplification that reduces the complexity of the instance. Also, we shift the release times of the jobs such that $r_j \geq R(\varepsilon \ell_j)$ for each job j. Intuitively, this means that large jobs are not released very early. This does not change the objective value too much since large jobs finish late anyway.

Lemma 5.8. *With $1+O(\varepsilon)$ loss, we can assume for each job j that $r_j \geq R(\varepsilon \ell_j)$ and that ℓ_j is a power of $(1+\varepsilon)$.*

Proof. Assume we are given an optimal schedule OPT. We construct a new schedule OPT' as follows: whenever a job j is processed within a time interval $[x, y]$ in OPT then it is processed in the interval $[(1+\varepsilon) x, (1+\varepsilon) y]$ in OPT'. We say we *scale time* by a factor of $1+\varepsilon$. We have that $OPT' \leq (1+\varepsilon) OPT$. The resulting schedule is feasible for an instance in which j has a processing demand of $(1+\varepsilon) \ell_j$. Thus, we can safely shift the release time of j to $\max\{r_j, R(\varepsilon \ell_j)\}$ and still obtain a valid schedule for processing demand ℓ_j.

Scaling time again by a factor of $1+\varepsilon$ yields that a demand of $(1+\varepsilon) \ell_j$ is processed for each job j. We increase the demand of j to the largest value $(1+\varepsilon)^x$ (for an integer x) such that $(1+\varepsilon)^x \leq (1+\varepsilon) \ell_j$. □

To make the instance even simpler, we show that we can assume that each job is released at a time R_x. For proving this we use a technique called *interval-hopping* which we will use in the subsequent lemmas as well.

Lemma 5.9. *Assume the adjustments of Lemma 5.8. With $1+\varepsilon$ loss, we can additionally assume that each job is released at time R_x for some integer x.*

Proof. We employ interval-hopping: Consider any schedule S. For each interval I_x, we take the work that is done in I_x in S and process it in the interval I_{x+1} rather than in I_x. This results in a loss of at most $1+\varepsilon$. Now each job j which was originally released within an interval $I_x := [R_x, R_{x+1})$ is not processed before R_{x+1}. Thus, we can safely move its release time to R_{x+1}. □

In order to simplify the complexity of the problem we want that to calculate the objective function as if each job finished at the end of an interval I_x. Therefore, for the remainder of this section we do not consider the objective function $\sum_{j \in J} w_j C_j$ but the objective function $\sum_{j \in J} w_j \min\{R_x : C_j \leq R_x\}$. As an effect, we can assume that the machine has constant speed within each interval.

Lemma 5.10. *Assume the adjustments of the previous lemmas. For the completion times C_j which result from any schedule we have that $\sum_{j\in J} w_j C_j \leq \sum_{j\in J} w_j \min\{R_x : C_j \leq R_x\} \leq (1+\varepsilon)\sum_{j\in J} w_j C_j$.*

Proof. The first inequality is obvious. For the second inequality, for all values C_j we have that $\min\{R_x : C_j \leq R_x\} \leq (1+\varepsilon)\, C_j$. □

In the algorithm we will distinguish between *large* and *small jobs*. Whether a job is small or large depends on the amount of work that the machine does in the interval in which the job is released. A job j is small if it is released at a time R_x such that $\ell_j \leq \varepsilon I_x$. Otherwise it is large. We denote by H_x and T_x the large and small jobs, respectively, which were released at time R_x. We introduce the following lemma in order to show that there is a $(1+\varepsilon)$-approximate schedule which has a certain simple structure.

Lemma 5.11. *Assume the adjustments of the previous lemmas. With $1+O(\varepsilon)$ loss, we can additionally assume that*

- *no small job is ever preempted,*
- *no small job is processed in more than one interval,*
- *the order in which the small jobs are executed obeys Smith's rule,*
- *each large job $j \in H_x$ is preempted only if there is an integer $k \leq \frac{1}{\varepsilon^3}$ such that a fraction of exactly $k \cdot \frac{\varepsilon I_x}{\ell_j}$ of the job has already been processed, and*
- *at any point in time in each set H_x there is at most one job which has already been processed but which has not been finished yet.*

Proof. We can assume the last claim without any loss. For the other claims we again use the technique of interval-hopping. For the sake of analysis, we first consider a relaxation of our instance I. i. e., an instance I' for which we have that $OPT(I') \leq OPT(I)$. Starting with $OPT(I')$, we construct a schedule S for I with the property that $S \leq (1+\varepsilon)^2\, OPT\,(I')$.

The instance I' is defined as follows: Let $\delta > 0$ be a constant which divides the demands of all small jobs. Then we replace each small job j by $\frac{\ell_j}{\delta}$ jobs with demand δ and weight $\delta w_j/\ell_j$, all with the same release date as j. We call those new jobs the *tiny jobs*. With an exchange argument one can show that in this instance it is optimal to schedule the small jobs according to Smith's rule, i. e., whenever a small job is scheduled the available job with the highest Smith's ratio is scheduled. W. l. o. g. we can assume that if in $OPT\,(I')$ a tiny job is processed which corresponds to a small job j then for all other small jobs j' we have that either all of their tiny jobs are already scheduled or none of them (i. e., the tiny jobs corresponding to the different small jobs do not mix). Also, w. l. o .g. we assume that in $OPT\,(I')$ jobs are only preempted at the end of intervals (since jobs are released only at the beginning of intervals).

We now perform an interval-hop of two intervals: For a loss of $(1+\varepsilon)^2$ we take the work that is done in each interval I_x in the schedule $OPT\,(I')$

and process it in the interval I_{x+2} rather than in I_x. We call the resulting schedule $OPT\,(I')_{hop}$. However, now in each interval I_{x+2} (which processes jobs which were processed in I_x by $OPT\,(I')$) there is a spare space of at least $2\varepsilon I_x$. We have that $OPT\,(I')_{hop} \leq (1+\varepsilon)^2\, OPT\,(I')$. We define the instances I_{hop} and I'_{hop} by taking I and I', respectively, and for each small/tiny job which was released at time R_x we move its release time to R_{x+2}. Note that $OPT\,(I')_{hop}$ is a valid schedule for I'_{hop} in which the tiny jobs are ordered according to Smith's rule.

Now we construct a schedule S for I_{hop} based on $OPT\,(I')_{hop}$ which has the properties stated in the lemma. First, we process each small job j whenever its corresponding tiny jobs were processed within $OPT\,(I')_{hop}$. Now there can be at most one small job j which starts within one interval I_{x+2} and does not finish within it. If there is such a job j then we use at most εI_x of the spare space to finish j. If at the end of I_{x+2} a large job $j \in H_y$ is preempted then – using again at most εI_x of the spare space – we can continue processing it until a fraction of exactly $k \cdot \frac{\varepsilon I_y}{\ell_j}$ of the job has been processed for some integer k. Due to our slightly changed objective function we have that $S \leq OPT\,(I')_{hop} \leq (1+\varepsilon)^2\, OPT\,(I') \leq (1+\varepsilon)^2\, OPT(I)$. □

We simplify the instance even further by simplifying each set T_x and H_x. During I_x the machine can process only small jobs with a total demand of I_x and at most $\frac{1}{\ell} I_x$ large jobs for each (large) demand ℓ. For the small jobs we already know that they are scheduled according to Smith's rule. For large jobs with equal demand it is clear that it is optimal to order them by non-increasing weight. Hence, some jobs in $T_x \cup H_x$ will definitely *not* be processed in I_x by an optimal solution. Thus, we can safely shift their release date to R_{x+1}.

For a set of jobs $J' \subseteq J$ we denote by $p\,(J')$ their total demand.

Lemma 5.12. *Assume the adjustments of the previous lemmas. Without any loss we can assume that*

- *the number of distinct job sizes in H_x is bounded by $|H_x| \leq 3\log_{1+\varepsilon}\frac{1}{\varepsilon} + 1$,*
- *the number of jobs in each distinct size is bounded by $1/\varepsilon$, and*
- $p\,(T_x) \leq I_x$.

Proof. Let $j \in H_x$. Since j is large we know that $\ell_j > \varepsilon I_x = \varepsilon^2(1+\varepsilon)^x$. Due to Lemma 5.8 we have that $r_j = R_x \geq R\,(\varepsilon \cdot \ell_j)$ which implies that $(1+\varepsilon)^x \geq \varepsilon \cdot \ell_j$. Since all demands are powers of $1+\varepsilon$ the number of distinct job sizes equals the number of integers y such that

$$\begin{array}{rccccc} & \varepsilon^2\,(1+\varepsilon)^x & < & (1+\varepsilon)^y & \leq & \frac{1}{\varepsilon}\,(1+\varepsilon)^x \\ \Leftrightarrow & 2\log_{1+\varepsilon}\varepsilon + x & < & y & \leq & \log_{1+\varepsilon}\frac{1}{\varepsilon} + x \\ \Leftrightarrow & 2\log_{1+\varepsilon}\varepsilon & < & y - x & \leq & \log_{1+\varepsilon}\frac{1}{\varepsilon}. \end{array}$$

This implies that $|H_x| \leq 3\log_{1+\varepsilon}\frac{1}{\varepsilon} + 1$. At most $\frac{I_x}{\varepsilon I_x} = 1/\varepsilon$ jobs of each particular size can be scheduled in I_x. Within each size the jobs are ordered by their weight. Thus, we can safely move the release times of all other jobs of

each size to R_{x+1}. Note that here we need that $\varepsilon < 1$ since otherwise it would not hold that $\varepsilon^2 (1+\varepsilon)^x < \frac{1}{\varepsilon} (1+\varepsilon)^x$.

For the small jobs recall that due to Lemma 5.11 we can assume that they are ordered by Smith's rule. Also, we assume that no small job is processed in more than one interval. Thus, we can assume that $p(T_x) \leq I_x$: We sort the jobs in T_x non-decreasingly by their Smith's ratio and then pick jobs according to this order until the next job would not fit in I_x anymore. The release time of all other jobs can safely be moved to R_{x+1} since we will not process them in R_x anyway. □

For getting better control on the instance we want to bound the flow time of each job, i. e., the time between its release date and its completion time. The following lemma establishes such a bound by introducing at most a loss of $1+\varepsilon$.

Lemma 5.13. *Assume the adjustments of the previous lemmas. With $1+\varepsilon$ loss, we can additionally assume that each job which is released at time R_x finishes in the interval $I_{x+s(\varepsilon)}$ the latest, where $s(\varepsilon)$ is a constant which depends only on ε.*

Proof. We use interval-hopping again and shift the work being done in each interval I_x to the interval I_{x+1}. From Lemma 5.12 we conclude that $p(T_x) + p(H_x) \leq I_x + \frac{1}{\varepsilon} \cdot \left(3 \log_{1+\varepsilon} \frac{1}{\varepsilon} + 1\right) \cdot \frac{1}{\varepsilon} (1+\varepsilon)^x$. We define

$$s(\varepsilon) := \left\lceil \log_{1+\varepsilon} \left(\frac{1}{\varepsilon} + \frac{1}{\varepsilon^4} \left(3 \log_{1+\varepsilon} \frac{1}{\varepsilon} + 1 \right) \right) \right\rceil + 1.$$

Our interval-hop creates a spare space of size εI_x in each interval I_{x+1}. Thus, in the interval $I_{x+s(\varepsilon)}$ there is now a spare space of at least $\varepsilon I_{x+s(\varepsilon)-1}$. We calculate that

$$\begin{aligned} I_x + \frac{1}{\varepsilon} \cdot \left(3 \log_{1+\varepsilon} \frac{1}{\varepsilon} + 1 \right) \cdot \frac{1}{\varepsilon} (1+\varepsilon)^x &= (1+\varepsilon)^x \cdot \left(\varepsilon + \varepsilon^2 \left(3 \log_{1+\varepsilon} \frac{1}{\varepsilon} + 1 \right) \right) \\ &\leq (1+\varepsilon)^x \cdot \varepsilon \cdot I_{s(\varepsilon)-1} \\ &= \varepsilon \cdot I_{x+s(\varepsilon)-1} \end{aligned}$$

and thus we can process all jobs $T_x \cup H_x$ in the spare space in the interval $I_{x+s(\varepsilon)}$. This implies that there is a $(1+\varepsilon)$-approximative solution in which all jobs $T_x \cup H_x$ finish in the interval $I_{x+s(\varepsilon)}$ the latest. □

Now we want to glue small jobs to *packs* together. The reason is that later in the dynamic program we can treat these packs like big jobs. The next lemma shows that this loses at most $1+\varepsilon$ in the objective value.

Formally, we partition the ordered list of the jobs in each set T_x into at most $2/\varepsilon^2$ packs, each with size at most $\varepsilon^2 \cdot I_x$. Denote by $P_{x,i}$ the i-th pack of small jobs which are released at time R_x.

Lemma 5.14. *Assume the adjustments of the previous lemmas. With $1+\varepsilon$ loss, we can additionally assume that*

- *in each interval I_x either all or none of the jobs in a pack $P_{x',i}$ are scheduled and*
- *each job which is released at time R_x finishes in the interval $I_{x+s(\varepsilon)+2}$ the latest.*

However, the ordering of the small jobs does not necessarily obey Smith's rule anymore.

Proof. We use interval-hopping and shift the work which is done in each interval I_x to the interval I_{x+2}. This gives us a free space of εI_{x+1} in each interval I_{x+2}. Now consider all packs $P_{x',i}$ such that some but not all of the jobs in $P_{x',i}$ are scheduled within I_x. Due to the original ordering by Smith's rule, this holds for at most one pack from each release time. The total demand of these packs is upper bounded by

$$\begin{aligned}\sum_{i=x-s+1}^{x} \varepsilon^2 \cdot I_i &\leq \varepsilon^3 \cdot \sum_{i=0}^{x} (1+\varepsilon)^i \\ &= \varepsilon^2 \cdot \left((1+\varepsilon)^{x+1} - 1\right) \\ &\leq \varepsilon \cdot I_{x+1}.\end{aligned}$$

Thus, in the gained free space we can schedule all jobs from packs $P_{x',i}$ which have partly but not fully been processed. □

Before we describe the dynamic program we summarize our adjustments on the instance:

- for each job j we have that r_j and ℓ_j are powers of $(1+\varepsilon)$,
- for each job j it holds that $r_j \geq R(\varepsilon \ell_j)$,
- the number of jobs in H_x is bounded by a constant for each release time R_x, and
- $p(T_x) \leq I_x$.

We showed that all these adjustments lose at most a factor of $(1+O(\varepsilon))$ in the optimal objective value. Further, we showed that for an adjusted instance as above there is a $(1+O(\varepsilon))$-approximative solution with the following properties:

- no small job is ever preempted,
- the small jobs are grouped into packs and no pack is processed in more than one interval,
- each large job $j \in H_x$ is preempted only if there is an integer $k \leq \frac{1}{\varepsilon^3}$ such that a fraction of exactly $k \cdot \frac{\varepsilon I_x}{\ell_j}$ of the job has already been processed,

- at any point in time in each set H_x there is at most one job which has already been processed but which has not finished yet,
- the value of the solution is measured according to the objective function $\sum_{j \in J} w_j \min\{R_x : C_j \leq R_x\}$ (which results in a higher value than the original objective function), and
- each job with release time R_x finishes before time $R_{x+s(\varepsilon)+1}$.

Now we describe the dynamic program which finds the best solution with the above properties. Each table entry is identified by a combination of

- an interval I_x,
- for each interval I_y with $x - s(\varepsilon) \leq y < x$,
  - the subset of jobs in H_y which have already been fully processed,
  - a job $j \in H_y$ and an integer $k \leq \frac{1}{\varepsilon^3}$ such that a fraction of exactly $k \cdot \frac{\varepsilon I_x}{\ell_j}$ of j has been processed, and
  - the subset of the packs $P_{y,i}$ which have already been fully processed.

Since we need to consider at most $s(\varepsilon) \cdot |J|$ intervals in total, the number of table entries is bounded by

$$\begin{aligned} & (s(\varepsilon)+2) \cdot |J| \cdot \left(2^{\frac{1}{\varepsilon} \cdot 3 \log_{1+\varepsilon} \frac{1}{\varepsilon} + \frac{1}{\varepsilon}} \cdot \left(\frac{1}{\varepsilon} \cdot 3 \log_{1+\varepsilon} \frac{1}{\varepsilon} + \frac{1}{\varepsilon} \right) \cdot \frac{1}{\varepsilon^3} \cdot 2^{\frac{2}{\varepsilon^2}} \right)^{s(\varepsilon)+2} \\ \in \quad & O\left(|J| \, 2^{poly(1/\varepsilon)} \right). \end{aligned}$$

In order to compute the value for each table entry which corresponds to an interval I_x we need to enumerate all possibilities to schedule the available large jobs and packs of small jobs in I_x. Note that due to our changed objective function it does not matter at what exact time within the interval a job finishes: each job which finishes within I_x is charged R_{x+1}. For one table entry this computation can be done in time $O\left(2^{s(\varepsilon) \cdot \left(\frac{1}{\varepsilon} 3 \log_{1+\varepsilon} \frac{1}{\varepsilon} + \frac{1}{\varepsilon} + \frac{2}{\varepsilon^2}\right)}\right)$.

The preprocessing of the jobs can be done in $O(n \cdot \log n + s(\varepsilon) \cdot n)$. The following operations are needed.

- Round the release times and demands of all jobs: $O(n)$.
- Partition the jobs into small and large jobs: $O(n)$.
- Define the packs of small jobs and adjust their release times. This requires $O(n \log n + s(\varepsilon) \cdot n)$ since we need to sort the small jobs and define the sets T_x for each interval I_x.
- Define the adjusted release times of the large jobs. This requires $O(n \log n + s(\varepsilon) \cdot n)$ again since we need to sort the jobs and define the sets H_x for each interval I_x.

This yields the following theorem:

Theorem 5.15. *There is an efficient polynomial time approximation scheme for the increasing speed scheduling problem with release dates with a running time of* $O\left(2^{poly(1/\varepsilon)}n + n\log n\right)$.

In order to do the computation in the dynamic program it is not necessary to know the exact speed function. It is sufficient to know the points in time when a total demand of $(1+\varepsilon)^x$ has already been processed for the relevant values of x. Recall that at most $s(\varepsilon)\cdot|J|$ intervals are relevant for us. Thus, we obtain the following corollary:

Corollary 5.16. *There is an EPTAS for the flow scheduling problem.*

Proof. We need to determine the start and end points of the at most $s(\varepsilon)\cdot|J|$ intervals I_x. This (approximation of the EAF) can be computed in $O(s(\varepsilon)\cdot poly(n))$ time: each value R_x can be determined in polynomial time by a quickest-flow computation (see [38]). The remainder follows from Theorem 5.15. □

5.4 Tractable Cases of ISS

In this section we analyze the structure of the increasing speed scheduling problem. We identify some properties which allow efficient algorithms for certain special cases. Moreover, we provide insights which are necessary for our analysis of the Smith's rule algorithm in Section 5.5. Throughout this section we assume that all jobs are released at time $t = 0$. Accordingly, we can restrict ourselves to non-preemptive schedules.

If all jobs have unit weight a simple exchange argument shows that it is optimal to order the jobs ascendingly by demand. However, we can prove a slightly more general result:

Theorem 5.17. *If in an instance of ISS there is an ordering for the jobs such that* $\frac{w_j}{\ell_j} \geq \frac{w_{j+1}}{\ell_{j+1}}$ *and* $\ell_j \leq \ell_{j+1}$ *for each index* j *then it is optimal to order the jobs non-descendingly by demand (or non-ascendingly by ratios* $\frac{w_j}{\ell_j}$*, respectively).*

The theorem can be shown using the following lemma repeatedly.

Lemma 5.18. *Assume that in a schedule there are two jobs* j, j' *with* $\frac{w_j}{\ell_j} \leq \frac{w_{j'}}{\ell_{j'}}$ *and* $\ell_j \geq \ell_{j'}$ *and* j' *is executed directly after* j*. Then the objective value does not increase if we swap* j *and* j'*. If additionally* $\frac{w_j}{\ell_j} < \frac{w_{j'}}{\ell_{j'}}$ *then swapping* j *and* j' *strictly decreases the objective value.*

Proof. We denote the original schedule by $\langle j, j'\rangle$ and the schedule obtained by swapping j and j' by $\langle j', j\rangle$. Denote by t_0 the time when j starts in $\langle j, j'\rangle$, by t_1 the time when j' terminates in $\langle j', j\rangle$, by t_2 the time when j terminates in $\langle j, j'\rangle$ and by t_3 the time when j' terminates in $\langle j, j'\rangle$. W. l. o. g. we assume that the machine has constant speed within the interval $[t_i, t_{i+1})$ for each $i \in \{0, 1, 2\}$ (see

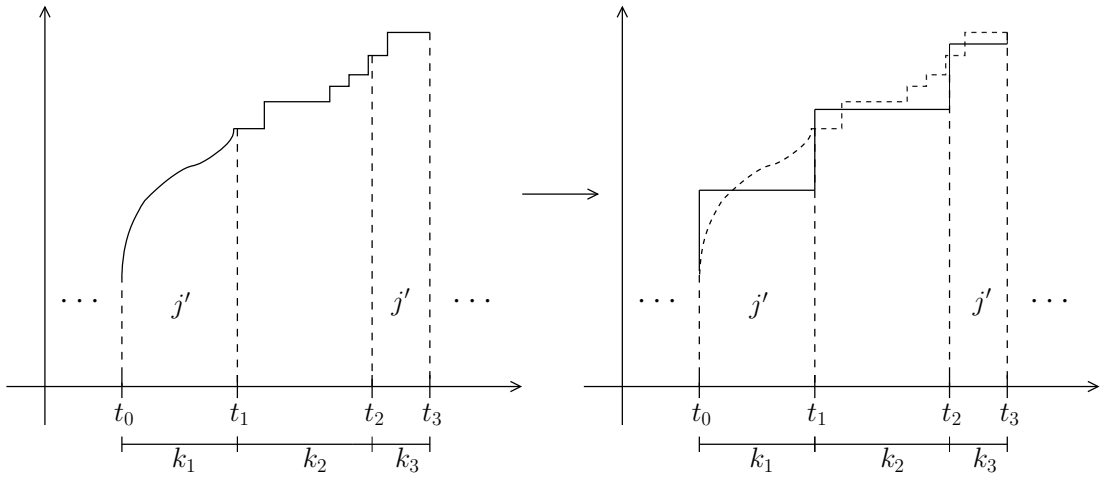


Figure 5.2: Proof of Lemma 5.18: We can assume w. l. o. g. that our machine has three different speeds within the interval $[t_0, t_3)$.

Figure 5.2). Denote the respective speeds by s_1, s_2, s_3. W. l. o. g. we assume that $s_3 = 1$. For ease of notations we define $k_i := t_i - t_{i-1}$ for $i \in \{1, 2, 3\}$.

We calculate that

$$\begin{array}{rcc}
cost\left(\langle j, j'\rangle\right) - cost\left(\langle j', j\rangle\right) & \geq & 0 \\
\Leftrightarrow \quad t_2 \cdot w_j + t_3 \cdot w_{j'} - t_1 \cdot w_{j'} - t_3 \cdot w_j & \geq & 0 \\
\Leftrightarrow \quad t_2 \cdot w_j + t_3 \cdot w_{j'} & \geq & t_1 \cdot w_{j'} + t_3 \cdot w_j \\
\Leftrightarrow \quad (k_1 + k_2)\, w_j + (k_1 + k_2 + k_3)\, w_{j'} & \geq & k_1 \cdot w_{j'} + (k_1 + k_2 + k_3)\, w_j \\
\Leftrightarrow \quad (k_2 + k_3)\, w_{j'} & \geq & k_3 \cdot w_j \\
\Leftrightarrow \quad \frac{w_{j'}}{\ell_{j'}} & \geq & \frac{w_j}{k_2 + \ell_{j'}}
\end{array}$$

The latter holds since $\frac{w_{j'}}{\ell_{j'}} \geq \frac{w_j}{\ell_j} = \frac{w_j}{\ell_{j'} + s_2 \cdot k_2} \geq \frac{w_j}{\ell_{j'} + k_2}$. If additionally $\frac{w_j}{\ell_j} < \frac{w_{j'}}{\ell_{j'}}$ then the above calculation gives strict inequality. □

For machines which run constantly with unit speed there is a well known exchange argument showing that Smith's rule yields an optimal schedule [100]. In our setting, this argument can easily be applied to jobs starting and finishing within an interval A in which the machine has constant speed. We show that the statement also holds for the set of all jobs which end in such an interval A (and do not necessarily start in A).

We split the time axis into intervals in which the speed function does not change its value. We denote by $s_1, s_2, ..., s_k$ the different speeds of the machine and by $A_1, A_2, ..., A_k$ the corresponding intervals, i. e., $A_i := s\left(s_i\right)^{-1}$. Assume a schedule S is given. We say a job j is *in an interval* A_i if the finishing time of j in S lies within A_i. Denote by J_i the jobs which lie in the interval A_i.

Lemma 5.19. *In an optimal schedule the jobs in J_i are ordered according to Smith's rule.*

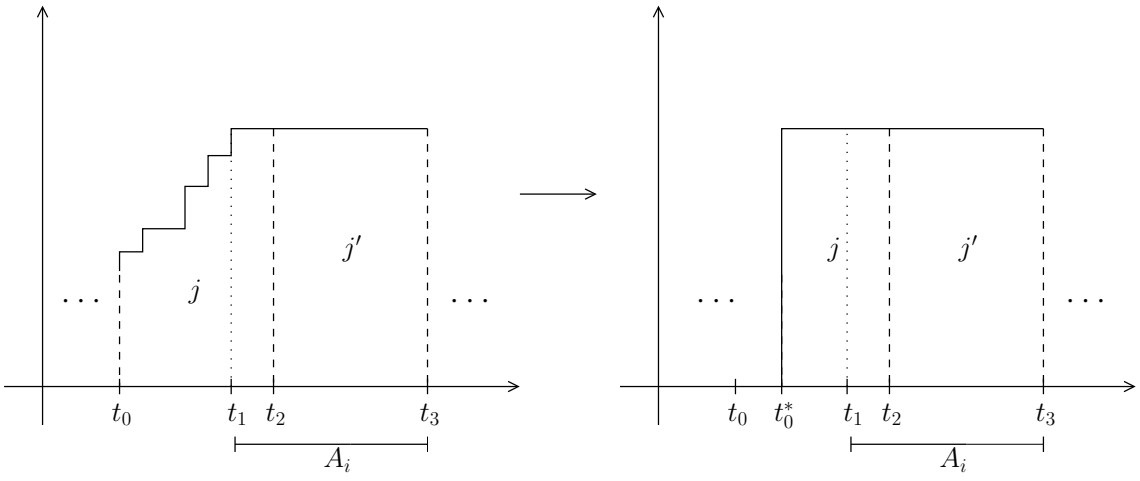


Figure 5.3: The adjustment of the machine speed in the proof of Lemma 5.19.

Proof. This can be shown by an exchange argument. Assume on the contrary that there is an optimal schedule and for two jobs $j, j' \in J_i$ we have that j is scheduled before j' but $\frac{w_j}{\ell_j} < \frac{w_{j'}}{\ell_{j'}}$. W. l. o. g. we can assume that j' is scheduled directly after j. We call this schedule $\langle j, j' \rangle$ and compare it with the schedule '$\langle j', j \rangle$ which is obtained by exchanging j and j'. If $\ell_{j'} \leq \ell_j$ then Lemma 5.18 proves the claim. So now assume that $\ell_{j'} > \ell_j$.

We denote by t_0 the starting time of j in $\langle j, j' \rangle$, by t_1 the starting time of interval A_i, by t_2 the finishing time of j in $\langle j, j' \rangle$, and finally by t_3 the finishing time of j' in $\langle j, j' \rangle$. See Figure 5.3 for a sketch. We observe that the finishing times of j and j' in both schedules ($\langle j, j' \rangle$ and $\langle j', j \rangle$) do not change if we adjust the machine as follows: Denote by t_0^* the timestep with $\int_{t_0}^{t_1} s(x)dx = (t_0^* - t_0)\, s_i$. We redefine our machines by setting $s(x) := 0$ for $x \in [t_0, t_0^*)$ and $s(x) := s_i$ for $x \in [t_0^*, t_1)$. Now in both schedules j and j' start and finish within interval A_i. Thus, the claim can be shown with the same argument which shows that Smith's rule is optimal for $1||\sum w_j C_j$:

$$\begin{array}{rcc} & \langle j', j \rangle & > & \langle j, j' \rangle \\ \Leftrightarrow & w_{j'} \frac{\ell_{j'}}{s_i} + w_j \frac{\ell_j + \ell_{j'}}{s_i} & > & w_j \frac{\ell_j}{s_i} + w_{j'} \frac{\ell_j + \ell_{j'}}{s_i} \\ \Leftrightarrow & w_j \cdot \ell_{j'} & > & w_{j'} \cdot \ell_j \\ \Leftrightarrow & \frac{w_j}{\ell_j} & > & \frac{w_{j'}}{\ell_{j'}} \end{array}$$

This is contradicts our assumption that $\frac{w_j}{\ell_j} < \frac{w_{j'}}{\ell_{j'}}$. □

Knowing the property above we are ready to introduce our dynamic program for the special case that the number of speed changes of s is bounded by a constant. The dynamic program starts with a list of the jobs ordered by Smith's rule. It successively removes a job j from the list and chooses the interval $A_i = [a_i, a_{i+1})$ in which j finishes. Inside A_i, the job j is scheduled right after the last job which finishes within A_i. If j is the first job assigned to A_i we try all start offsets less or equal a_i for which j finishes within A_i. Thus, in the

dynamic programming table we need to encode how many jobs have already been removed from the list and how much space (at the beginning and at the end) of each interval is already occupied by jobs.

Now we describe the dynamic program in detail. First, we order the jobs in a list according to Smith's rule. Ties are broken arbitrarily. We define $L := \sum_j \ell_j$. Note that there are at most L possible start times for a job. Each entry in the dynamic programming table is identified by

- the number of jobs which are still in the job list, i. e., an entry equal to q means that the last q jobs in the job list are still unscheduled and
- for each interval $A_i = [a_i, a_{i+1})$ there are values $t_i \leq L$ and $t'_i \leq L$ such that the intervals $\left[a_i, a_i + \frac{t_i}{s_i}\right)$ and $\left[a_{i+1} - \frac{t'_i}{s_i}, a_{i+1}\right)$ are already used by some jobs.

We store in each table entry the best possible objective value for the remaining jobs which is possible with the given constraints. Note that the number of entries in the table is bounded by $n \cdot L^{2k}$. The value for an entry $(k, t_1, t'_1, ..., t_k, t'_k)$ is computed as follows: Let j denote the $(n-k+1)$-th job. We try each interval A_i for scheduling j: if $t_i > 0$ we try to schedule j with start time $a_i + \frac{t_i}{s_i}$. If $t_i = 0$ we try to schedule it with each start time $a_i - \frac{m}{s_i}$ with $m < \ell_{n-k+1}$. By "try" we mean that we check carefully if scheduling the job at the respective position contradicts the fact that other intervals are already occupied by some jobs. We pick the position for j which yields the minimum total objective value.

Theorem 5.20. *If the number of different values for the speed function is bounded by a constant there is a pseudopolynomial dynamic program which solves the ISS problem optimally.*

Proof. Computing an entry in the dynamic programming table can be done in pseudopolynomial time. The number of entries in the dynamic programming table is bounded by $n \cdot L^{2k}$. Lemma 5.19 implies that our procedure finds the optimal solution. □

Note that this pseudopolynomial algorithm cannot be combined with the pseudopolynomial algorithm for earliest arrival flows [78, 107] to achieve an exact, pseudopolynomial algorithm for the entire problem. An EAF corresponds to a machine with pseudopolynomially many speeds. Our result requires a *constant* number of speeds.

Now we study the special cases where all jobs have the same demand or the same Smith's factors. We will benefit from these insights in the analysis in Sections 5.5 and 5.7, respectively. The following lemma holds not only for increasing speed functions but also for speed functions which might increase or decrease.

Lemma 5.21. *If all jobs have the same demand then it is optimal to order the jobs non-ascendingly by their weight. This is still true if the speed of the machine can increase* and *decrease.*

Proof. This can be shown by an exchange argument. Assume on the contrary that there is an optimal schedule in which a job j is scheduled before a job j' but $w_j < w_{j'}$. W. l. o. g. we can assume that j' is scheduled directly after j. Then the objective value strictly decreases if we swap j and j'. This contradicts that the original schedule was optimal. □

Particularly important for the next section will be the following proposition.

Proposition 5.22. *If in an instance I all jobs have the same Smith's ratio then*

- *there is an optimal schedule which orders the jobs non-descendingly by demand and*
- *there is a worst possible schedule which orders the jobs non-ascendingly by demand.*

Proof. Follows from Lemma 5.18 and its proof. □

5.5 A Tight Analysis of Smith's Rule

In this section we present a tight analysis which shows that the Smith's rule algorithm is exactly a $\frac{\sqrt{3}+1}{2}$-approximation. We achieve this by explicitly characterizing worst-case instances. Let $I = (J, M)$ be an instance of the increasing speed scheduling problem. We denote by $SR(I)$ the worst possible schedule which obeys Smith's rule (i. e., tie-breaking decisions are taken such that the total weight of the schedule is maximized). We show how to transform I into an instance with a special structure without decreasing $SR(I)/OPT(I)$. Then we show that on instances with this structure the Smith's rule algorithm is exactly a $\frac{\sqrt{3}+1}{2}$-approximation.

Assume on the contrary that there is an instance I such that $\frac{SR(I)}{OPT(I)} > \frac{\sqrt{3}+1}{2}$, again with $SR(I)$ being the objective value of the *worst* possible schedule which obeys Smith's rule. (We will show later by perturbing the weights of the jobs that one cannot prove a better approximation factor for the Smith's rule algorithm even if all tie breaking decisions are done optimally.)

As already mentioned above, in the following lemmas we show how one can transform I into an instance with a special structure without decreasing $SR(I)/OPT(I)$. First, we define a partition of the jobs in I into classes C_i such that two jobs belong to the same class if and only if they have the same Smith's ratio. The classes are sorted descendingly by the Smith's ratios of their elements, i. e., for two jobs $j \in C_i$ and $j' \in C_{i'}$ it holds that $\frac{w_j}{\ell_j} \geq \frac{w_{j'}}{\ell_{j'}}$ if and only if $i \leq i'$. Now we show that there are instances with the worst possible fraction $SR(I)/OPT(I)$ in which all jobs have a Smith's ratio of 1.

Lemma 5.23. *For any instance I there is an instance $I' = (J', M)$ such that $w_j/\ell_j = 1$ for all jobs $j \in J'$ and $\frac{SR(I')}{OPT(I')} \geq \frac{SR(I)}{OPT(I)}$. Moreover, if in I the demands of all jobs are integral then in I' the demands of all jobs are integral as well.*

Proof. Let $\tilde{I}$ be an instance with as few classes C_i as possible such that $\frac{SR(\tilde{I})}{OPT(\tilde{I})} \geq \frac{SR(I)}{OPT(I)} =: \alpha$. If in $\tilde{I}$ there is only one class then we can scale the weights of the jobs such that $w_i/\ell_i = 1$ for all jobs and we are done. So now assume that there are at least two classes in $\tilde{I}$. Denote by $SR(C_i)$ and $OPT(C_i)$ the amount that a class C_i contributes towards $SR\left(\tilde{I}\right)$ and $OPT\left(\tilde{I}\right)$, respectively. Since $\frac{SR(\tilde{I})}{OPT(\tilde{I})} \geq \alpha$ there must be a class C_k such that $\frac{SR(\mathcal{C}_k)}{OPT(\mathcal{C}_k)} \geq \alpha$. Now there are two cases:

- The class C_k is the class with the highest Smith's ratio (i. e., $k = 1$). We create I' by removing the jobs of all other classes from $\tilde{I}$. We have that $SR(I') = SR(C_k)$ and $OPT(I') \leq OPT(C_k)$ since in $SR(\tilde{I})$ the jobs in C_k are the first jobs which are scheduled. This implies that $\frac{SR(I')}{OPT(I')} \geq \frac{SR(C_k)}{OPT(C_k)} \geq \alpha$.
- The class C_k is not the class with the highest Smith's ratio (i. e., $k \neq 1$). Then we increase the weights of the jobs in C_k until they all have the Smith's ratio of C_{k-1}. Denote by I' the resulting instance. Clearly, I' has one class less than I. Let $\beta > 1$ denote the factor by which we increased the weights of the jobs in C_k. Then we calculate that $OPT(I') \leq OPT\left(\tilde{I}\right) + (\beta - 1)\, OPT(C_k)$ and $SR(I') = SR(\tilde{I}) + (\beta - 1)\, SR(C_k)$. We calculate that

$$\begin{aligned}
\frac{SR(I')}{OPT(I')} &\geq \frac{SR(\tilde{I}) + (\beta - 1)\, SR(C_k)}{OPT\left(\tilde{I}\right) + (\beta - 1)\, OPT(C_k)} \\
&\geq \frac{\alpha\left(OPT\left(\tilde{I}\right) + (\beta - 1)\, OPT(C_k)\right)}{OPT\left(\tilde{I}\right) + (\beta - 1)\, OPT(C_k)} \\
&= \frac{SR(I)}{OPT(I)}
\end{aligned}$$

□

Recall from Proposition 5.22 that if all Smith's ratios are identical then $OPT(I)$ orders the jobs non-descendingly by demand and the worst Smith's rule schedule $SR(I)$ orders the jobs non-ascendingly by demand. Now we want to study the demands of the jobs. W. l. o. g. we assume that all jobs have integral demands. Assume that the jobs are ordered such that for each pair of jobs j, j' with $j < j'$ we have that $\ell_j \leq \ell_{j'}$. Let k denote the largest integer such that $\sum_{i=1}^{k} \ell_i \leq \frac{1}{2} \sum_{i=1}^{|J|} \ell_i$. We define $J_{small} := \{j | j \leq k\} \subset J$. For an instance I' we denote the respective set by J'_{small}. In the following lemma we show that w. l. o. g. all small jobs in J_{small} have demand 1.

Lemma 5.24. *For any instance $I = (J, M)$ such that all jobs in J have a Smith's ratio of 1 there is an instance $I' = (J', M)$ such that*

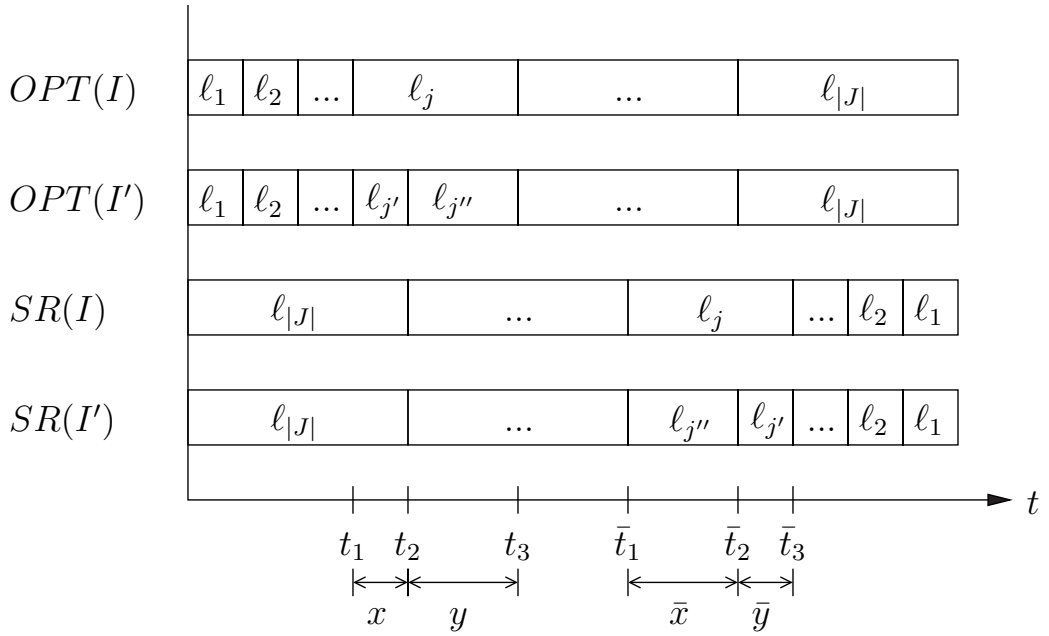


Figure 5.4: The sketch shows the jobs in I and I' in the orders in which they are sorted in SR and OPT. To simplify the readability of the sketch we assume that the machine runs constantly with unit speed.

- *J'_{small} consists only of jobs with demand 1 and*
- *$\frac{SR(I')}{OPT(I')} \geq \frac{SR(I)}{OPT(I)}$.*

Moreover, if in I the demands of all jobs are integral then in I' the demands of all jobs are integral as well.

Proof. If in J_{small} there are only jobs of demand 1 then there is nothing to show. So now assume that J_{small} contains a job whose demand is at least two. Let j denote the job with smallest index such that $\ell_j \geq 2$.

For $I' = (J', M)$ we use the machine M without any changes and we define $J' := J \setminus \{j\} \cup \{j', j''\}$ where j' is a job with demand $\ell_{j'} := 1$ and j'' is a job of demand $\ell_{j''} := \ell_j - 1$. Since we want the Smith's ratios of all jobs to be 1 we define $w_{j'} := 1$ and $w_{j''} := \ell_j - 1$. Note that $\ell_1 \leq \ell_2 \leq ... \leq \ell_{j-1} \leq \ell_{j'} \leq \ell_{j''} \leq \ell_{j+1} \leq ... \leq \ell_{|J|}$. Figure 5.4 shows a sketch of the modification. Now we show that $\frac{SR(I')}{OPT(I')} \geq \frac{SR(I)}{OPT(I)}$.

Denote by t_1 the start time of j in $OPT(I)$. Denote by t_2 and t_3 the finish time of j' and j'' in $OPT(I')$, respectively. Similarly, let $\bar{t}_1$ be the start time of j in $SR(I)$. Denote by $\bar{t}_2$ the finish time of j'' in $SR(I')$ and by $\bar{t}_3$ the finish time of j'' in $SR(I')$. Figure 5.4 shows a sketch. We define $SR_\Delta := SR(I) - SR(I') = \bar{t}_3 \cdot \ell_j - \bar{t}_3 \cdot \ell_{j'} - \bar{t}_2 \cdot \ell_{j''}$ and $OPT_\Delta := OPT(I) - OPT(I') = t_3 \cdot \ell_j - t_2 \cdot \ell_{j'} - t_3 \cdot \ell_{j''}$. In order to show that $\frac{SR(I')}{OPT(I')} \geq \frac{SR(I)}{OPT(I)}$ we first prove that $SR_\Delta \leq OPT_\Delta$ (in other words: the optimal solution saves more than the Smith's rule solution when we replace j by j' and j''). W. l. o. g. we assume that the machine M runs constantly with speed s in the time interval $[t_2, t_3)$. Similarly, we assume

that M runs with speed $\bar{s}$ in the time interval $[\bar{t}_2, \bar{t}_3)$. Note that $s \leq \bar{s}$ since $j \in J_{small}$ and thus $t_3 \leq \bar{t}_2$. We define $x := t_2 - t_1$ and $y := t_3 - t_2$. Similarly, we define $\bar{x} := \bar{t}_2 - \bar{t}_1$ and $\bar{y} := \bar{t}_3 - \bar{t}_2$.

We calculate that

$$\begin{aligned} OPT_\Delta &= (t_1 + x + y)\,\ell_j - (t_1 + x)\,\ell_{j'} - (t_1 + x + y)\,\ell_{j''} \\ &= y\ell_j - y\ell_{j''} \end{aligned}$$

and

$$\begin{aligned} SR_\Delta &= (\bar{t}_1 + \bar{x} + \bar{y})\,\ell_j - [(\bar{t}_1 + \bar{x})\,\ell_{j''} + (\bar{t}_1 + \bar{x} + \bar{y})\,\ell_{j'}] \\ &= \bar{y}\ell_j - \bar{y}\ell_{j'} \end{aligned}$$

and thus

$$\begin{aligned} OPT_\Delta - SR_\Delta &= y\ell_j - y\ell_{j''} - \bar{y}\ell_j + \bar{y}\ell_{j'} \\ &\geq (\ell_j - \ell_{j''}) \left(\frac{\ell_j - 1}{\bar{s}} \right) + \frac{1}{\bar{s}} (1 - \ell_j) \\ &= \frac{1}{\bar{s}} \left[(\ell_j - \ell_{j''})\,(\ell_j - 1) + 1 - \ell_j \right] \\ &= 0. \end{aligned}$$

We define $\beta := \frac{OPT(I) - OPT_\Delta}{OPT(I)}$ and $\gamma := \frac{SR(I) - OPT_\Delta}{SR(I)}$. A simple calculation shows that $\gamma \geq \beta$. We further obtain that

$$\begin{aligned} \frac{SR(I')}{OPT(I')} &\geq \frac{SR(I) - OPT_\Delta}{OPT(I) - OPT_\Delta} \\ &= \frac{\gamma \cdot SR(I)}{\beta \cdot OPT(I)} \\ &\geq \frac{SR(I)}{OPT(I)}. \end{aligned}$$

We repeat the operation described above until we obtain an instance in which there are only jobs of demand 1 in the respective set J_{small}. □

Now we are interested in the first job which does not have demand 1. Let j' denote the index of this job. Let s' and t' denote its start and finish times in $OPT(I)$ and let $\bar{s}'$ and $\bar{t}'$ denote its start and finish times in $SR(I)$. The next lemma shows that w. l. o. g. we can assume that $\bar{s}' \leq s'$.

Lemma 5.25. *Let I be an instance such that all jobs in J have a Smith's ratio of 1, J_{small} contains only jobs of demand 1, and the demands of all jobs are integral. Then there is an instance $I' = (J', M)$ such that*

- *J'_{small} consists only of jobs of demand 1,*
- *$\bar{s}' \leq s'$,*

- $\frac{SR(I')}{OPT(I')} \geq \frac{SR(I)}{OPT(I)}$, *and*
- *all jobs in* J' *have integral demand and a Smith's ratio of 1.*

Proof. If $\bar{s}' > s'$ then we apply to j' the procedure described in the proof of Lemma 5.24. This is possible since the start times of j' guarantee that $\bar{s}' \geq s'$ (with the notation in the proof). □

Now we show how we can change our instance I to an instance I' in which there is at most one (long) job which is not contained in J_{small}. This affects the ratio $\frac{SR(I)}{OPT(I)}$ only by a factor of $(1-\varepsilon)$ for an arbitrarily small ε. The ε will turn out to be negligible later.

Lemma 5.26. *Let* $\varepsilon > 0$. *Let* I *be an instance such that all jobs in* J *have a Smith's ratio of 1,* J_{small} *contains only jobs of demand 1, and* $\bar{s}' \leq s'$. *Then there is an instance* $I' = (J', M')$ *such that*

- J'_{small} *consists only of jobs of demand 1,*
- $|J' \setminus J'_{small}| = 1$ *(i. e., there is exactly one job which is larger than 1),*
- M' *has at most one speed change which occurs when the large job is finished in* $SR(I')$, *and*
- $\frac{SR(I')}{OPT(I')} \geq (1-\varepsilon)\frac{SR(I)}{OPT(I)}$.

Proof. First, we do the following two changes on I in order to obtain an instance I'': we combine all jobs which are not contained in J_{small} to one long job $j_{m''}$. Then we dramatically increase the speed of M after a timestep t^* to be defined later.

Denote by s_j and t_j the start and finish time of each job j in $OPT(I)$ and by $\bar{s}_j$ and $\bar{t}_j$ the start and finish time of each job j in $SR(I)$. Assume that $\ell_1 \leq \ell_2 \leq ... \leq \ell_{|J|}$. We define $I'' = (J'', M'')$ as follows: $J'' := J_{small} \cup \{j''\}$ where j'' is a new job with weight $w_{j''} := \ell_{j''} := \sum_{i=j'}^{|J|} \ell_i$. We obtain M'' with the following operations: We start with M. Let t^* denote the timestep with $\int_0^{t^*} s(x)dx = \ell_{m''}$. Let $\delta > 0$ be a constant to be defined later. In the time interval $[t^*, t^* + \delta]$ we define M'' to have speed $|J''_{small}|/\delta$. We allow only values for δ such that M'' never slows down (i. e., there are not timesteps x and x' with $x < x'$ but $s''(x) > s''(x')$ with s'' being the speed function of M''). Note that M'' completes all jobs within the interval $[0, t^* + \delta]$.

We claim that for any given $\varepsilon > 0$ there is a $\delta > 0$ such that $\frac{SR(I'')}{OPT(I'')} \geq (1-\varepsilon)\frac{SR(I)}{OPT(I)}$. In order to analyze $\frac{SR(I'')}{OPT(I'')}$ we split the jobs into *bricks* of demand 1. Let j be a job. For j we introduce ℓ_j bricks $B_{j,1}, ..., B_{j,\ell_j}$. Let $pay_{SR(I)}(j)$ be the amount that j contributed towards the objective function in $SR(I)$. We define $pay_{SR(I)}(B_{j,k}) := pay_{SR(I)}(j)/\ell_j$ for all k with $1 \leq k \leq \ell_i$. We define $pay_{OPT(I)}(j)$ and $pay_{OPT(I)}(B_{j,k})$ similarly. Let $\mathcal{B}$ denote the set of all bricks. Note that some of the bricks correspond to j'' in I'' *and* to some

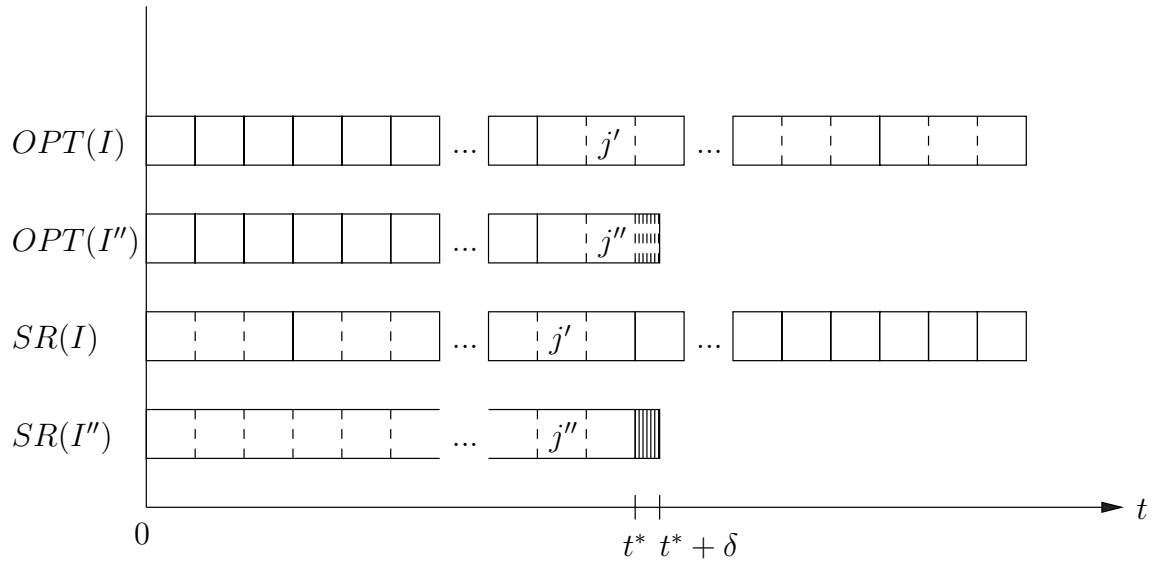


Figure 5.5: The time intervals in which the jobs are processed according to the respective schedules. In order to simplify the sketch we assume that M runs constantly with unit speed. The machine M'' runs with constant speed until time t^* and becomes extremely fast directly after t^*.

other jobs from $J \setminus J_{small}$ in I. We say a brick $B_{j,k}$ is *processed* in $OPT(I)$ during a time interval $[s,t]$ if until time s a fraction of $\frac{k-1}{\ell_j}$ of j is processed in $OPT(I)$ and until time t a fraction of $\frac{k}{\ell_j}$ of j is processed in $OPT(I)$.

For a schedule S we define $\mathcal{B}_{early}(S)$ to be all bricks which are processed up to time t^* in S. We define $\mathcal{B}_{late}(S) := \mathcal{B} \setminus \mathcal{B}_{early}(S)$. We observe that for all bricks $B_{j,k} \in \mathcal{B}_{early}(OPT(I))$ we have that $pay_{OPT(I)}(B_{j,k}) \geq pay_{OPT(I'')}(B_{j,k})$. Moreover, for all bricks $B_{j,k} \in \mathcal{B}_{late}(OPT(I))$ we have that $pay_{OPT(I)}(B_{j,k}) \leq t^* + \delta$. We observe that for all bricks $B_{j,k} \in \mathcal{B}_{early}(SR(I))$ we have that $pay_{SR(I'')}(B_{j,k}) \geq pay_{SR(I)}(B_{j,k})$. Moreover, for all bricks $B_{j,k} \in \mathcal{B}_{late}(SR(I))$ we have that $pay_{SR(I'')}(B_{j,k}) \geq t^*$. Figure 5.5 shows a sketch of the above reasoning.

To simplify notation, for a set of bricks $\mathcal{B}$ and a schedule S we define $pay_S(\mathcal{B}) := \sum_{B \in \mathcal{B}} pay_S(B)$. We calculate that

$$\begin{aligned}\frac{SR(I'')}{OPT(I'')} &= \frac{pay_{SR(I'')}(\mathcal{B}_{early}(SR(I''))) + pay_{SR(I'')}(\mathcal{B}_{late}(SR(I'')))}{pay_{OPT(I'')}(\mathcal{B}_{early}(OPT(I''))) + pay_{OPT(I'')}(\mathcal{B}_{late}(OPT(I'')))} \\ &\geq \frac{pay_{SR(I)}(\mathcal{B}_{early}(SR(I))) + |\mathcal{B}_{late}(SR(I))| \cdot t^*}{pay_{OPT(I)}(\mathcal{B}_{early}(OPT(I))) + |\mathcal{B}_{late}(OPT(I))| \cdot (t^* + \delta)}.\end{aligned}$$

Observe that

$$\begin{aligned}pay_{OPT(I)}(\mathcal{B}_{late}(OPT(I))) &\geq pay_{SR(I)}(\mathcal{B}_{late}(SR(I))) \\ &\geq |\mathcal{B}_{late}(SR(I))| \cdot t^*.\end{aligned}$$

This implies

$$\frac{pay_{SR(I)}\left(\mathcal{B}_{early}(SR(I))\right) + |\mathcal{B}_{late}(SR(I))| \cdot t^*}{pay_{OPT(I)}\left(\mathcal{B}_{early}(OPT(I))\right) + |\mathcal{B}_{late}(OPT(I))| \cdot t^*} \geq \frac{SR(I)}{OPT(I)}.$$

In other words: if we increased the speed of M to infinity after time t^* then the optimal schedule saves not less than the Smith's rule schedule. Thus, for every given $\varepsilon > 0$ there is a $\delta > 0$ such that

$$\frac{pay_{SR(I)}\left(\mathcal{B}_{early}(SR(I))\right) + |\mathcal{B}_{late}(SR(I))| \cdot t^*}{pay_{OPT(I)}\left(\mathcal{B}_{early}(OPT(I))\right) + |\mathcal{B}_{late}(OPT(I))| \cdot (t^* + \delta)} \geq (1-\varepsilon)\frac{SR(I)}{OPT(I)}.$$

It remains to modify I' such that the machine has at most two different speeds (without reducing the factor between the two schedules). Recall that in J'' there are some jobs of demand 1 and one large job. Moreover, in $OPT(I'')$ the jobs of demand 1 are all finished before t^*. We define $I' := (J'', M')$ and specify M' as follows: we start with M''. Let $z := \int_0^{t^*} s'(x)dx/t^*$ (i. e., z is the average speed of the machine during the time interval $[0, t^*]$). Then we define

- $s''(x) := z$ for all $x \in [0, t^*)$ and
- $s''(x) := s'(x)$ for all $x \in [t^*, t^* + \delta]$.

We have that $SR(I') = SR(I'')$ and $OPT(I') \leq OPT(I'')$. The former holds since the Smith's rule schedule cannot benefit from our adjustments. The latter holds since at each point in time M' has processed at least as much as M'' (i. e., the finishing times of the jobs cannot increase). This completes the adjustments of this lemma. □

Now we are ready to prove the approximation ratio of $\frac{\sqrt{3}+1}{2}$ for the Smith's rule algorithm.

Lemma 5.27. *Every algorithm that always respects Smith's rule is a* $\left(\frac{\sqrt{3}+1}{2}\right)$*-approximation algorithm.*

Proof. We prove that for any instance I it holds that $\frac{SR(I)}{OPT(I)} \leq \frac{\sqrt{3}+1}{2}$. Let $\varepsilon > 0$. Using the lemmas above we derive an instance $I' = (J', M')$ with the following properties:

- for all jobs $j \in J'$ it holds that $w_j/\ell_j = 1$,
- the demands (and thus the weights) of all jobs are integral,
- in $SR(I')$ the jobs are sorted non-ascendingly by their demand,
- in $OPT(I')$ the jobs are sorted non-descendingly by their demand,
- J'_{small} consists only of jobs with demand 1,
- there is exactly one job $j' \in J'$ which does not have demand 1 (denote by $\ell_{j'}$ its demand),

- M' has at most one speed change which occurs when the large job is finished in $SR(I')$ (denote by t^* the time of the speed change), and
- $\frac{SR(I')}{OPT(I')} \geq (1-\varepsilon)\,\frac{SR(I)}{OPT(I)}$.

In the remainder of this proof, we show that $\frac{SR(I')}{OPT(I')} \leq \frac{\sqrt{3}+1}{2}$ which implies the desired claim since we can choose ε arbitrarily small.

We define $L := \sum_{j \in J'} \ell_j$. Let s_1 and s_2 denote the two speed values of M' with $s_1 \leq s_2$. Denote by $t_{\max}$ the time when the last job finishes. We set $k := L - \ell_{j'}$ (i. e., we have k small jobs of demand 1) and calculate that

$$SR(I') = t^* \cdot \ell_{j'} + \sum_{i=1}^{k} \left(t^* + \frac{i}{s_2} \right) = L \cdot t^* + \frac{k\,(k+1)}{2 \cdot s_2}$$

and

$$OPT(I') = \ell_{j'} \cdot t_{\max} + \sum_{i=1}^{k} \frac{i}{s_1} = \ell_{j'} \cdot t_{\max} + \frac{k\,(k+1)}{2 \cdot s_1}.$$

We substitute $t_{\max} = \frac{\ell_{j'}}{s_1} + \frac{k}{s_2}$ and obtain

$$\begin{aligned} \frac{SR(I')}{OPT(I')} &= \frac{\frac{\ell_{j'}}{s_1} \cdot k + \frac{\left(\ell_{j'}\right)^2}{s_1} + \frac{k(k+1)}{2s_2}}{\frac{\left(\ell_{j'}\right)^2}{s_1} + \frac{k \cdot \ell_{j'}}{s_2} + \frac{k(k+1)}{2s_1}} \\ &\leq \frac{\ell_{j'} \cdot k + \left(\ell_{j'}\right)^2}{\left(\ell_{j'}\right)^2 + \frac{k^2+k}{2}} \\ &\leq \frac{\ell_{j'} \cdot k + \left(\ell_{j'}\right)^2}{\left(\ell_{j'}\right)^2 + \frac{k^2}{2}} \end{aligned}$$

using that $k \leq \ell_{j'}$. For fixed $\ell_{j'}$ we define $f(k) := \frac{\ell_{j'} \cdot k + \left(\ell_{j'}\right)^2}{\left(\ell_{j'}\right)^2 + \frac{k^2}{2}}$. We calculate that

$$\begin{aligned} f'(k) = 0 \quad &\Leftrightarrow \quad \left(\ell_{j'}\right)^2 + \frac{k^2}{2} - k^2 - k \cdot \ell_{j'} = 0 \\ &\Leftrightarrow \quad k = \left(-1 \pm \sqrt{3}\right) \cdot \ell_{j'}. \end{aligned}$$

Basic calculus shows that the function $f(k)$ attains its maximum in $[0, \ell_{j'}]$ at $k = \left(-1 + \sqrt{3}\right) \cdot \ell_{j'}$. We further calculate that

$$\begin{aligned}\frac{SR(I')}{OPT(I')} &\leq f\left(\left(\sqrt{3}-1\right)\cdot \ell_{j'}\right)\\ &= \frac{\ell_{j'}\cdot\left(\sqrt{3}-1\right)\cdot\ell_{j'}+\left(\ell_{j'}\right)^2}{\left(\ell_{j'}\right)^2+\frac{\left(4-2\sqrt{3}\right)\cdot\left(\ell_{j'}\right)^2}{2}}\\ &= \frac{\sqrt{3}+1}{2}.\end{aligned}$$

So for any $\varepsilon > 0$ we can show that $\frac{SR(I)}{OPT(I)} \leq \frac{1}{1-\varepsilon}\cdot\frac{SR(I')}{OPT(I')} \leq \frac{1}{1-\varepsilon}\cdot\frac{\sqrt{3}+1}{2}$. This implies that $\frac{SR(I)}{OPT(I)} \leq \frac{\sqrt{3}+1}{2}$. □

The proof of Lemma 5.27 bounds the approximation ratio of Smith's rule for instances with the derived properties by $\frac{\sqrt{3}+1}{2}$. Due to the previous lemmas we know that this implies the same bound for all instances. The following Proposition shows that on instances with the mentioned properties the approximation factor of Smith's rule can indeed be arbitrarily close to $\frac{\sqrt{3}+1}{2}$.

Proposition 5.28. *For any $\varepsilon > 0$ there is an instance I_ε with the property that $\frac{SR(I_\varepsilon)}{OPT(I_\varepsilon)} \geq \frac{\sqrt{3}+1}{2}(1-\varepsilon)$.*

Proof. Let $\varepsilon > 0$. Let $\ell_{j'}$ be an integer to be defined later. We introduce $k := \left\lfloor\left(\sqrt{3}-1\right)\ell_{j'}\right\rfloor$ jobs which have demand and weight 1 and one job j' which has demand and weight $\ell_{j'}$. We define our machine M_ε to have speed 1 in the time interval $[0, \ell_{j'}]$ and speed s in the time interval $\left[\ell_{j'}, \ell_{j'}+\frac{k}{s}\right]$ (we will define s later). We calculate that

$$OPT\left(I_\varepsilon\right) \leq \frac{k(k+1)}{2} + \ell_{j'}\cdot\left(\ell_{j'}+\frac{k}{s}\right)$$

and

$$SR\left(I_\varepsilon\right) = \left(\ell_{j'}\right)^2 + k\cdot\ell_{j'} + \frac{k(k+1)}{2s}$$

We further calculate that

$$\begin{aligned}\frac{SR\left(I_\varepsilon\right)}{OPT\left(I_\varepsilon\right)} &\geq \frac{\left(\ell_{j'}\right)^2 + k\cdot\ell_{j'} + \frac{k(k+1)}{2s}}{\frac{k(k+1)}{2}+\ell_{j'}\cdot\left(\ell_{j'}+\frac{k}{s}\right)}\\ &= \frac{\left(\ell_{j'}\right)^2 + \left\lfloor\left(\sqrt{3}-1\right)\ell_{j'}\right\rfloor\cdot\ell_{j'} + \frac{\left\lfloor\left(\sqrt{3}-1\right)\ell_{j'}\right\rfloor\left(\left\lfloor\left(\sqrt{3}-1\right)\ell_{j'}\right\rfloor+1\right)}{2s}}{\frac{\left\lfloor\left(\sqrt{3}-1\right)\ell_{j'}\right\rfloor\left(\left\lfloor\left(\sqrt{3}-1\right)\ell_{j'}\right\rfloor+1\right)}{2}+\ell_{j'}\cdot\left(\ell_{j'}+\frac{\left\lfloor\left(\sqrt{3}-1\right)\ell_{j'}\right\rfloor}{s}\right)}\end{aligned}$$

Now we choose s and $\ell_{j'}$ large enough such that

$$\begin{aligned}\frac{SR(I_\varepsilon)}{OPT(I_\varepsilon)} &\geq (1-\varepsilon)\,\frac{(\ell_{j'})^2 + \left(\left(\sqrt{3}-1\right)\ell_{j'}\right)\cdot \ell_{j'}}{\frac{\left((\sqrt{3}-1)\ell_{j'}\right)^2}{2} + (\ell_{j'})^2} \\ &= (1-\varepsilon)\,\frac{\sqrt{3}+1}{2}\end{aligned}$$

□

So far we compared $SR(I)$ and $OPT(I)$ where $SR(I)$ denotes the Smith's rule schedule with the *worst* possible tie-breaking. In the following theorem we state the main result of this section and in particular we show that even with the *best* possible tie-breaking the approximation ratio of the Smith's rule algorithm does not improve.

Theorem 5.29. *Any algorithm respecting Smith's rule is a $\frac{\sqrt{3}+1}{2}$-approximation, and none of these algorithms can achieve a better approximation factor for all instances.*

Proof. The upper bound for the approximation factor follows from Lemma 5.27. So far we always considered the worst possible tie-breaking for Smith's rule. Now consider a Smith's rule algorithm with the best possible tie-breaking for each instance. Let I be an instance. For any $\varepsilon > 0$ we can find a perturbation of the weights of the jobs in I yielding an instance I_ε such that

- there is only one possible ordering for the jobs in I_ε which obeys Smith's rule and still
- $\frac{SR(I_\varepsilon)}{OPT(I_\varepsilon)} \geq (1-\varepsilon)\,\frac{SR(I)}{OPT(I)}$.

Applying this reasoning to the worst-case instances presented in Proposition 5.28 shows that no Smith's rule algorithm can achieve a better approximation factor than $\frac{\sqrt{3}+1}{2}$. □

5.6 Blind Algorithms

The Smith's rule algorithm orders the jobs without knowledge of the machine. We call algorithms with this property *blind* algorithms. Assume that some blind algorithm has computed an ordering for a set of jobs. Given a dynamic network we can use Theorem 5.7 to obtain an optimal flow schedule for this given ordering. This yields the following theorem.

Theorem 5.30. *Assume there is a blind α-approximation algorithm for the ISS problem. Then there is also an α-approximation algorithm for the flow scheduling problem.*

In particular, since Smith's rule is a blind $\frac{\sqrt{3}+1}{2}$-approximation algorithm, we obtain the following corollary.

Corollary 5.31. *There is an approximation algorithm for the flow scheduling problem with approximation factor* $\frac{\sqrt{3}+1}{2}$.

Now we establish a lower bound for the possible performance ratio of any blind algorithm. Our original bound of 1.1215 [103] was improved to 1.1328 by an anonymous referee. Since the latter construction is gives a better bound and it is also simpler we present only that one.

Theorem 5.32. *No blind algorithm can have a better performance ratio than* $(19 + 3\sqrt{65})(22 + 2\sqrt{65}) \approx 1.1328$.

Proof. Let $\varepsilon > 0$. We consider the following instance I: there are two jobs with $\ell_1 = 1$, $w_1 = 1$, $\ell_2 = 2$, and $w_2 = \alpha$ (for a value α to be defined later). A blind algorithm has to order them without knowing the speed function of the machine. We consider two possible machines: machine M_1 runs constantly with unit speed. Machine M_2 runs with unit speed until time $t = 2$ and then accelerates to speed $1/\varepsilon$. Hence, the last job finishes at time $t = 2 + \varepsilon$.

We denote by $S(i, j)$ the schedules where job i is executed before job j, with $i, j \in \{1, 2\}$. On machine M_1 the schedule $S(1, 2)$ has an objective value of $w_1 + 3w_2$ and $S(2, 1)$ has an objective value of $2w_2 + 3w_1$. However, on machine M_2 the schedule $S(1, 2)$ yields an objective value of $w_1 + (2+\varepsilon)w_2$ and $S(2, 1)$ has an objective value of $2w_2 + (2 + \varepsilon)w_1$.

If the blind algorithm chooses the schedule $S(1, 2)$ then on machine M_1 it achieves an approximation ratio of $\frac{1+3\alpha}{3+2\alpha}$. If it chooses the schedule $S(2, 1)$ then on machine M_2 it has an approximation ratio of $\frac{2+2\alpha+2\varepsilon}{1+2\alpha+2\alpha\varepsilon}$. The worst case performance ratio of any blind algorithm is hence bounded from below by $f(\alpha) := \min\left\{\frac{1+3\alpha}{3+2\alpha}, \frac{2+2\alpha+2\varepsilon}{1+2\alpha+2\alpha\varepsilon}\right\}$. Choosing $\alpha := (5 + \sqrt{65})/4$ yields $f(\alpha) = (19 + 3\sqrt{65} + 2\varepsilon)(22 + 2\sqrt{65} + 2\alpha\varepsilon)$. Since we can choose ε arbitrarily small, our claim follows. □

5.7 Online Algorithms

In this section, we consider the increasing speed scheduling problem in an online setting. We show that the Smith's rule algorithm has still a competitive ratio of 2. Also, the lower bound of the blind algorithms carries over to a bound for online algorithms. Hence, there can be no online algorithm with a better competitive ratio than ≈ 1.1328. Finally, we show that if all jobs have unit weight then the intuitive shortest remaining processing time algorithm (SRPT) is optimal.

We assume the following online model:

- each job j has a release time r_j before which it cannot be processed,
- the existence and all data of a job become known at its release time, and
- at time t the speed of the machine up to time t is known, the speed of the machine *after* time t is not known.

The Smith's rule algorithm in the online-setting works as follows: We always process a job which has the largest Smith's factor among all available jobs. Denote by $SR_{online}(I)$ the resulting schedule (and its objective function value) for an instance I.

Theorem 5.33. In the online setting, the Smith's rule algorithm has a competitive ratio of 2.

Before we can prove the theorem we need some preparation. Let I be an instance of our problem. With the same reasoning as in Lemma 5.23 we can show that w. l. o. g. we can assume that all $\frac{w_j}{\ell_j} = 1$ for all jobs in I. However, note that here we need a different analysis since a job j might be preempted by a job j' with $r_j < r_{j'}$.

We choose ε such that it divides the demand of each job and each release time. We create a lower bound instance I' by replacing each job j by ℓ_j/ε jobs, each with demand ε and weight $\frac{w_j}{\ell_j}\varepsilon$, and release time r_j.

Lemma 5.34. *It holds that $OPT(I') \leq OPT(I)$.*

Proof. We start with $OPT(I)$. We show that splitting the jobs into pieces of demand ε does not increase the value of the objective function. Denote by S the schedule obtained by taking $OPT(I)$ and splitting each job j into ℓ_j/ε equal jobs with demand ε. Let $j \in J$ denote a job which is executed in one time interval $[s_j, t_j)$ in $OPT(I)$. Then j contributes $w_j \cdot t_j$ towards the sum of weighted completion times. Denote by J_j the jobs resulting from splitting j. For a job $j' \in J_j$ denote by $t(j')$ its completion time in S. In S the jobs resulting from splitting j contribute

$$\begin{aligned}\sum_{j' \in J_i} t(j') \cdot \frac{w_j}{\ell_j}\varepsilon &\leq \sum_{j' \in J_i} t_j \cdot \frac{w_j}{\ell_j}\varepsilon \\ &= \frac{\ell_j}{\varepsilon} \cdot t_j \cdot \frac{w_j}{\ell_j}\varepsilon \\ &= t_j \cdot w_j\end{aligned}$$

towards the objective function. A similar reasoning can be applied to jobs j which are executed in more than one time interval. Applying the above reasoning to all jobs shows that $S \leq OPT(I)$. This implies that $OPT(I') \leq OPT(I)$. □

The important property of the lower bound instance I' is that on I' the online Smith's rule algorithm is optimal, as the following lemma shows.

Lemma 5.35. *If in $SR_{online}(I')$ no job is preempted by another job with the same Smith's factor, then the schedule $SR_{online}(I')$ is optimal.*

Proof. Due to the choice of ε the condition of the lemma ensures that no job is ever preempted. Then the claim follows from Lemma 5.21. □

Now we are ready to prove Theorem 5.33. First we give some intuition for the proof. We define a function f as follows: Assume that a job starts its execution at time $t = 0$ and is never interrupted. We define $f(\ell)$ to be its completion time depending on its demand ℓ. If the machine runs constantly with unit speed then f is the identity function. If the machine might accelerate but does not slow down then we can still guarantee that f is concave. If the machine has no idle time then in the lower bound instance I' the Smith's rule algorithm pays (roughly) the area underneath the curve f, see Figure 5.6 for a sketch. In I the Smith's rule algorithm pays at most $f(T) \cdot T$ where T denotes the finishing time of the last job. Since f is concave we can prove that the two values differ by at most a factor of two.

Now we give a detailed proof.

Proof of Theorem 5.33. We observe that the time until the last job finishes can be partitioned into time intervals in which the machine has no idle time. Denote by $\mathcal{L}$ the set of these intervals. We now show the competitive factor of 2 for the online Smith's rule algorithm. We consider each interval $L = [x, y] \in \mathcal{L}$ separately. Let $J_L \subseteq J$ denote the jobs which finish within L in $SR_{online}(I)$ and let $J'_L \subseteq J'$ be the jobs which finish within L in $SR_{online}(I')$. We denote by $cost(J_L)$ and $cost(J'_L)$ the value that the jobs in J_L and J'_L contribute towards the objective function in $SR_{online}(I)$ and $SR_{online}(I')$, respectively. We want to show that $cost(J_L) \leq 2 \cdot cost(J'_L)$.

We define a function f as described above. Note that since the machine never slows down f is concave. We define $k := \frac{1}{\varepsilon} \sum_{j \in J_L} \ell_j$, i. e., the schedule I' executes k jobs the interval L. We want to show that $cost(J_L) \leq 2 \cdot cost(J'_L)$. We calculate that

$$\begin{aligned} cost(J'_L) &= \sum_{i=1}^{k} \varepsilon f(i \cdot \varepsilon) \\ &\geq \int_0^{k\varepsilon} f(z)\, dz \\ &\geq \frac{f(k\varepsilon)}{2} \cdot k\varepsilon \end{aligned}$$

(see Figure 5.6 for a sketch) and thus

$$\begin{aligned} cost(J_L) &\leq \sum_{j \in J_L} f(k\varepsilon) \cdot w_j \\ &= f(k\varepsilon) \cdot \sum_{j \in J_L} \ell_j \\ &= f(k\varepsilon) \cdot k\varepsilon \\ &\leq 2 \cdot cost(J_{L'}) \end{aligned}$$

Doing this reasoning for each interval $L \in \mathcal{L}$ proves the claim. □

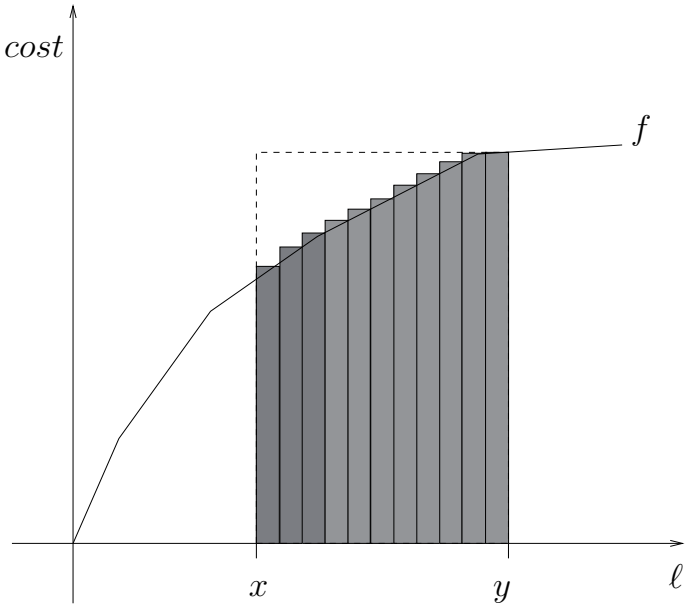


Figure 5.6: The function f together with the schedule of the jobs in J'_L. The y-axis denotes the cost that jobs contribute towards the objective function. The narrow bars represent the value of our lower bound. The dotted line represents our upper bound on the value for the true objective function (for the jobs in J_L).

Note that our analysis is tight since there are examples (even for the case that the speed of the machine does not change) where the online Smith's rule algorithm does not perform better than factor 2, see [94].

5.7.1 Lower Bound for Online Algorithms

The lower bound construction for blind algorithms presented in Section 5.6 carries over to a bound for online algorithms. It actually also holds if all jobs are released at time $t = 0$ and only the information about the machine becomes available online.

Theorem 5.36. *No online algorithm for the increasing speed scheduling problem can achieve a better competitive ratio than* $(19 + 3\sqrt{65})(22 + 2\sqrt{65}) \approx 1.1328$*, even if all jobs have the same release time.*

Proof. Let $\varepsilon > 0$. We use the same two jobs that were employed in the proof of Theorem 5.32 with $\ell_1 = 1$, $w_1 = 1$, $\ell_2 = 2$, and $w_2 = \alpha := (5 + \sqrt{65})/4$. Both jobs are released at time $t = 0$. The adversary runs the machine with unit speed until job 2 has finished. Then it accelerates the machine to speed $1/\varepsilon$. For the remainder of the calculations we pretend that $\varepsilon = 0$, knowing that we can choose ε arbitrarily small.

Assume that when job 2 finishes the algorithm has already processed a fraction of $x < 1$ of job 1. This yields an objective value of $(2+x)(\alpha+1)$. However, the optimum would have been $1 + \alpha(2 + x)$. The expression

$$\frac{(\alpha+1)(2+x)}{\alpha(2+x)+1}$$

is minimized for $x = 0$ (since $(\alpha+1)/\alpha \geq (2\alpha+2)/(2\alpha+1)$). Hence, the online algorithm cannot gain anything by processing job 1 partly before finishing job 2. Also, the algorithm cannot gain anything by processing job 2 partly before finishing job 1 (it might as well finish job 1 before start processing job 2). The remainder of the reasoning is identical to the proof of Theorem 5.32. □

5.7.2 Unit Weight Case

In this section we prove that for the unit weight case the shortest remaining processing time algorithm (SRPT) is optimal. The SRPT-algorithm works as follows: at each point in time, we process the available job which has the shortest remaining demand. Ties are broken arbitrarily. For an instance I denote by $SRPT(I)$ the resulting schedule.

This extends the fact that SRPT is optimal for the problem $1|r_i, pmtn|\sum C_j$.

Theorem 5.37. *If all jobs in an instance I have the same weight then $SRPT(I)$ is optimal.*

Proof. Assume on the contrary that $SRPT(I)$ is not optimal. Let $OPT(I)$ be the optimal schedule which differs from $SRPT(I)$ for the first time as late as possible. Let time t be the first timestep where $OPT(I)$ and $SRPT(I)$ differ. Let j_{long} denote the job which is processed by $OPT(I)$ at time t and let j_{short} be the job which is processed by $SRPT(I)$ at time t. Then at time t the remaining demand of j_{short} is strictly shorter than the remaining demand of j_{long}. (If both remaining demands equal then by an exchange argument we can show that there must be an optimal schedule which also schedules j_{short} at time t.) Let t_{short} and t_{long} denote the finishing times of j_{short} and j_{long} in $OPT(I)$. Let I denote the union of the time intervals after t in which $OPT(I)$ processes either j_{short} or j_{long}. We define a new schedule $OPT'(I)$ as follows: outside I the schedules $OPT'(I)$ and $OPT(I)$ are identical. Within I the schedule $OPT'(I)$ processes j_{short} and j_{long} according to the SRPT-rule. Denote by t'_{short} and t'_{long} the finishing times of j_{short} and j_{long} in $OPT'(I)$, respectively. If $t_{\text{short}} < t_{\text{long}}$ then $t'_{\text{short}} < t_{\text{short}}$ and $t'_{\text{long}} = t_{\text{long}}$. If $t_{\text{short}} > t_{\text{long}}$ then $t'_{\text{short}} < t_{\text{long}}$ and $t'_{\text{long}} = t_{\text{short}}$. In both cases we derive that $t_{\text{short}} + t_{\text{long}} > t'_{\text{short}} + t'_{\text{long}}$. The finishing times of all other jobs are the same in both schedules. This contradicts that $OPT(I)$ is an optimal schedule. □

5.8 Conclusion

In this chapter, we studied the flow scheduling problem which to the best of our knowledge has not been considered before. It motivates the increasing speed scheduling problem (ISS) which was also not studied for itself before. From an

algorithmic point of view, ISS with release dates is fully understood since it is NP-hard and we provided a PTAS. However, the NP-hardness proof due to the contained $1|r_i, pmtn|\sum w_j C_j$-problem requires release dates for the jobs. An intriguing question that we have to leave open is, whether the ISS problem without release dates is also NP-hard. Also, it remains open whether there is an FPTAS for ISS without release dates.

For blind algorithms, the best possible approximation factor is still unclear. We know that the Smith's rule algorithm yields a $\frac{\sqrt{3}+1}{2} \approx 1.366$ approximation. On the other hand, the best known lower bound is 1.1328. It remains open to fill this gap. A randomized algorithm might be an option. Recall that if the speed of the machine might increase and decrease, the best possible deterministic blind algorithm gives a 4-approximation, but a randomized algorithm with approximation ratio $e \approx 2.718$ is known [31]. Also, for the online setting it remains open to improve the 2-approximation or to strengthen the lower bound of 1.1328.

For the flow scheduling problem we considered only the setting with a single source s and a single sink t. Various generalizations are possible, like multiple sources and/or multiple sinks. In the setting of multiple sources and a single sink an earliest arrival flow still exists. However, note that in flow scheduling we can no longer guarantee that every optimal solution completes a job before particles of the next job arrive in the sink. For example, consider an instance with two sources that share an edge on their way to the sink t. It might be that in any optimal solution particles from two jobs from the two sources arrive at t simultaneously. For the setting of multiple sources and multiple sinks not even an EAF is guaranteed to exist [15, 36] which makes the flow scheduling problem even harder. Nevertheless, this yields a lot of space for further research.

Chapter 6

Periodic Maintenance Problem

6.1 Introduction

In former times, airplanes used to operate completely without any electronic equipment. Devices like an on-board computer or a fly-by-wire system were unimaginable. Those times have passed a long time ago. Nowadays, every modern aircraft uses a lot of computer-based systems to steer and control the plane, communicate with the air-traffic controller etc. In particular, systems like an autopilot would not be possible without the help of sophisticated on-board computers. In fact, in a modern airplane the input of the pilot is translated by the system to appropriate actions of the actuators, e. g., the ailerons or the rudders. Certain actions are even disallowed by the system, like bringing the plane into an angle which is too steep. Also, the computer controls the actuators without an explicit command of the pilot, e. g., in order to stabilize the plane or to run the autopilot.

Such on-board computers have great advantages. They take over work of the pilot who can then concentrate on other tasks. In particular, this reduces the risk of accidents due to human errors during long flights. However, in order to guarantee flight safety the computer must operate according to the specifications at all times. Everybody with a computer at home knows that they sometimes behave unexpectedly. For a home computer a little discrepancy between expected and actual behavior is totally acceptable. However, for a computer that controls an aircraft even a slight delay in the execution of a program might result in serious problems. For instance, consider a program which controls providing the engines with fuel. Abnormal termination of a program or even the crash of the entire computer could seriously endanger the flight and the lives of the crew and the passengers. Hence, the computer and its programs must be designed such that they are always guaranteed to operate as expected. Such a guarantee requires sophisticated mathematical methods.

The mathematical area that provides such guarantees is *real-time scheduling*. The on-board computer is modeled by a machine which runs some *tasks*. Each task represents a program that is executed by the processor of the computer. Usually, such a program does not need the processor continuously but only during some time windows. In those time windows, the program computes some information which is needed until a certain deadline (e. g., how much fuel needs to be transported to the engines). Depending on the program, the length of these time windows might vary. This is modeled as follows. Each task τ_i (computer program) is characterized by a processing time c_i (length of the mentioned time window), a period length p_i and a deadline d_i. The task τ_i continuously creates jobs with processing time c_i. A job which is created at a time t by the task τ_i must be fully processed by the machine until time $t + d_i$. As an additional information, we know that if τ_i emits a job at some time t, the next job of τ_i will be emitted at time $t + p_i$ at the earliest. The goal is to define a scheduling policy that at each point in time decides what job is executed by the processor. Usually, one allows *preemption*, i. e., it is allowed to interrupt a job and continue its execution later (without any additional delay for the interruption). Given a set of tasks, the question is whether there is a scheduling policy that guarantees that all deadlines are met. We call such a policy a *feasible policy.*

One very natural (and indeed also very strong) scheduling policy is *earliest deadline first* (EDF). This policy always runs the jobs which has the earliest deadline among all released but unfinished jobs. It is known that whenever there is a feasible policy for a set of tasks then EDF is feasible [24]. However, testing *whether* EDF is feasible is strongly *coNP*-hard, even for special cases [13, 29]. Sometimes one is interested in a simpler scheduling policy. One well-studied class of policies are *static-priority* policies. Such policies order the tasks in a fixed priority list. The priority of a job is then given by the priority of the task that created it. A natural static-priority schedule is the *rate monotonic schedule* (RM). This policy orders the tasks non-increasingly by the fractions $\frac{1}{p_i}$. This implies that a task with small period length has higher priority than a task with large period length. It is known for the special case of implicit deadlines (i. e., $d_i = p_i$ for all tasks τ_i) that if there is a feasible static-priority schedule then RM is feasible [74].

Our industrial partner, a major avionics company, approached us with the following real-time scheduling problem. They want to assign the tasks of their on-board computer to its different processors. The scheduling rules on each processor are much more conservative than in the general setting. In their model, they assume that each tasks τ_i creates a job *exactly* every p_i timesteps (strict periodic setting). Each job has to be processed *immediately* after it has been emitted (i. e., $d_i = c_i$). In particular, given feasible offsets for the tasks on a machine, no further scheduling decisions are necessary. The only degree of freedom on each machine is to specify the offset for each task, i. e., the first timestep when the task creates a job. This already defines the creation times of all jobs of this task. Note that in most other real-time scheduling settings one cannot define the times when the tasks emit jobs while here they are a core

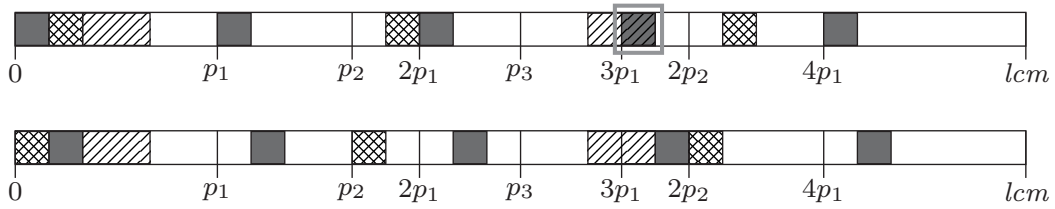


Figure 6.1: The picture above shows the three tasks $\tau_1 = (1, 6)$ (solid), $\tau_2 = (1, 10)$ (checkered), and $\tau_3 = (2, 15)$ (striped) on one machine. The upper part shows an infeasible assignment of offsets ($a_1 = 0,\ a_2 = 1,\ a_3 = 2$) whereas the lower part shows a feasible assignment of offsets ($a_1 = 1$, $a_2 = 0$, $a_3 = 2$). Notice that the schedule repeats after the least common multiple (lcm) of the periods.

part of the computed solution. The aim is to minimize the number of machines such that each task is assigned to one processor and for each processor there is a feasible schedule (assignment of offsets).

In this chapter we present theoretical results for the problem of our industrial partner. All results are joint work with Friedrich Eisenbrand, Nicolai Hähnle, Martin Niemeier, Martin Skutella, and José Verschae [27].

6.1.1 Problem Definition

The model that is used for our problem is as follows. One is given *tasks* $\tau_1, \ldots, \tau_n$ where each task $\tau_i = (c_i, p_i)$ is characterized by its *execution time* $c_i \in \mathbb{N}$ and *period length* $p_i \in \mathbb{N}$ (or short: its *period*). The goal is to assign the tasks to identical machines and to compute offsets $a_i \in \mathbb{N}_0$ such that no collision occurs: A task τ_i generates one *job* with execution time c_i at every timestep $a_i + p_i \cdot \ell$ for all $\ell \in \mathbb{N}_0$. Each job needs to be processed immediately and non-preemptively after its generation on the task's machine. A collision occurs if two jobs are simultaneously active on one machine. The overall aim is to minimize the total number of used machines. We refer to this problem as the *periodic maintenance problem (PMP).* See Figure 6.1 for an example.

An important special case for practitioners is the *harmonic PMP:* Here, we assume that the period lengths of the tasks all divide each other, i. e., $p_j | p_{j'}$ or $p_{j'} | p_j$ for each pair of tasks $\tau_j, \tau_{j'}$.

In the sequel, we will use the following notation: we define the *utilization* of a task $\tau = (c, p)$ by $\text{util}(\tau) := c/p$. Intuitively, the utilization of a task τ determines the fraction of the total processing power of a machine that is used by τ. For a set of tasks I we define $\text{util}(I) := \sum_{\tau \in I} \text{util}(\tau)$. Note that the utilization $\text{util}(I)$ is a lower bound for $OPT(I)$, the number of machines needed in an optimal solution. For a machine M that has some tasks assigned to it we define $\text{util}(M) := \sum_{\tau \text{ on } M} \text{util}(\tau)$. We denote by $q_1, q_2, \ldots, q_k$ the period lengths arising in a given instance, always assuming that $q_1 \leq q_2 \leq \ldots \leq q_k$.

6.1.2 Related Work

There is a wide amount of literature on real-time scheduling, for an overview see [13, 18, 70]. In many cases one is interested in verifying whether a set of tasks can be scheduled on one machine with a certain scheduling policy. Very common policies are rate monotonic (RM) and earliest deadline first (EDF). In particular, Liu and Layland [74] show that the rate monotonic schedule (i. e., priorities are assigned to the tasks according to their period lengths) is best possible among all scheduling policies with fixed priorities in the setting of implicit deadlines. Also, it is proven that the EDF algorithm produces a feasible schedule assuming that a feasible schedule exists [24].

The periodic maintenance problem is a generalization of the Bin-Packing problem (e. g., see [51, 105]). In fact, if the period lengths of all tasks are identical, the problem is equivalent to Bin-Packing. However, the problem is more general and the approximation ratios known for Bin-Packing do not carry over. In particular, it is possible to approximate Bin-Packing with an approximation ratio of 1.5 [98] but we show that it is impossible to approximate our problem with a ratio better than $n^{1-\varepsilon}$ for any $\varepsilon > 0$ if $P \neq NP$. For the special case of harmonic periods we can prove better approximation guarantees. But even then, the best possible polynomial time algorithm achieves a factor of 2 compared to 1.5 for Bin-Packing [98] (assuming that $P \neq NP$). Even asymptotically, one cannot prove a better bound than 2 for our problem (comparing to the APTAS [35] and the asymptotically optimal algorithm for Bin-Packing, see [51]).

In non-periodic scheduling there are results for minimizing the needed number of machines for a set of jobs where each job has a release time and a deadline. This problem is known as the SRDM-problem [21]. If the jobs have equal release times there is a 2-approximation algorithm [110]. For the general non-preemptive case there is a constant factor approximation [20]. If additionally all jobs have identical execution times there is a 6-approximation algorithm [110].

6.1.3 Outline of the Chapter

In this chapter, we present approximation algorithms and inapproximability results for the periodic maintenance problem. The approximation guarantees of our algorithms depend on the number of different period lengths k that arise in the given set of tasks. First, we study the general PMP. For arbitrary k, we present an algorithm which finds a solution using at most $2 \cdot OPT + k - 1$ machines. If k is bounded by a constant, we give an algorithm which uses only $(\frac{3}{2} + \varepsilon)OPT + k$ machines, for any positive constant ε. Both algorithms are presented in Section 6.2. There is not much room for improvement in this setting since we show that it is NP-hard to find solutions which use at most $(\frac{3}{2} - \varepsilon)OPT + k - 1$ machines.

There is little hope for a nontrivial approximation algorithm that has an approximation guarantee independent of the parameter k: Even for the PMP restricted to unit execution times, if n is the number of tasks, we show that there is no $n^{1-\varepsilon}$ approximation for any positive constant ε, unless $P = NP$.

General periodic maintenance problem		
Period lengths k	Algorithms	Hardness results
k arbitrary	$2OPT + k - 1$	$n^{1-\varepsilon}$
k constant	$\left(\frac{3}{2} + \varepsilon\right) OPT + k$	$\left(\frac{3}{2} - \varepsilon\right) OPT + k - 1$

Harmonic periodic maintenance problem		
Period lengths k	Algorithms	Hardness results
k arbitrary	$2OPT$	$(2 - \varepsilon)\, OPT + o(OPT)$
k constant	$(1 + \varepsilon)\, OPT + k$	$\left(\frac{3}{2} - \varepsilon\right) OPT + k - 1$
q_k/q_1 constant	$(1 + \varepsilon)\, OPT + 1$	$(2 - \varepsilon)\, OPT$

Table 6.1: The approximability landscape of the periodic maintenance problem. The value k denotes the number of period lengths arising in an instance; q_k and q_1 denote the largest and smallest period lengths, respectively.

The inapproximability of the general problem justifies considering the case of harmonic periods. Our results to this end are presented in Section 6.3. It turns out that it is much more tractable in terms of achievable approximation factors. Our main result here is a 2-approximation algorithm. We also show that this is the best we can hope for, since it is NP-hard to approximate within a factor of $2 - \varepsilon$ (for any $\varepsilon > 0$). This holds even asymptotically. If we restrict the problem even further we can improve the approximation guarantee: If k is bounded by a constant there is an asymptotic PTAS computing solutions with at most $(1 + \varepsilon)OPT + k$ machines. If even q_k/q_1 is bounded by a constant we can find a solution which needs at most $(1+\varepsilon)OPT+1$ machines in polynomial time.

Finally, in Section 6.4 we conclude and address open problems. Table 6.1 shows an overview of our approximation algorithms and complexity results.

6.2 General Periodic Maintenance Problem

In this section we study the periodic maintenance problem in the setting of arbitrary, i. e., not necessarily harmonic, period lengths. We present algorithms using at most $2 \cdot OPT + k - 1$ and $\left(\frac{3}{2} + \varepsilon\right) OPT + k$ machines for arbitrary and fixed k, respectively. (Recall that k denotes the number of different period lengths in an instance.) The former is a pure First-Fit algorithm, the latter uses enumeration and additionally First-Fit. Then we show that it is NP-hard to approximate the problem with factor of $n^{1-\varepsilon}$ for any $\varepsilon > 0$. In particular, this shows that the additive k in the performance guarantees of our algorithms is necessary.

6.2.1 First-Fit Algorithm

In this section we present our First-Fit algorithm for the general PMP that uses at most $2 \cdot OPT + k - 1$ machines in the setting that the number of period lengths k is part of the input.

Let I be an instance of the PMP. Our algorithm works as follows. We partition the tasks according to their period lengths. For each period length separately we do First-Fit: We iterate over the tasks and add each task to the machine with smallest index where it fits. If there is no such machine, we open a new machine. Note that we do not mix tasks with different period lengths on the same machine. Hence, it is easy to check whether we can add a task to a machine or not: Let I' be a set of tasks such that all tasks in I' have the same period length p. There is a schedule for all tasks in I' on one machine if and only if $\sum_{\tau_j \in I'} c_j \leq p$. Denote by $FF(I)$ the number of machines in the resulting schedule.

Theorem 6.1. *For any instance I it holds that $FF(I) \leq 2 \cdot OPT + k - 1$.*

Proof. For each arising period length q_r denote by I^r all tasks with this period length. Let $\mathcal{M}_r$ denote the machines used by $FF(I)$ for the tasks in I^r. Since we added the tasks by First-Fit, we have that $\frac{1}{2}(|\mathcal{M}_r| - 1) < \text{util}(I^r)$. For the overall bound, we calculate that

$$\begin{aligned} FF(I) &= \sum_{r=1}^{k} |\mathcal{M}_r| \\ &< \sum_{r=1}^{k} (2 \cdot \text{util}(I^r) + 1) \\ &\leq 2 \cdot OPT + k \end{aligned}$$

Since on both sides of the inequality have integer values, we conclude that $FF(I) \leq 2 \cdot OPT + k - 1$. □

6.2.2 Enumeration and First-Fit

For this section we assume that k is bounded by a constant. We present an algorithm that needs at most $(3/2 + \varepsilon)\,OPT + k$ machines, for any $\varepsilon > 0$. It performs some enumeration and subsequently uses First-Fit. We borrow ideas from the APTAS for Bin-Packing, see [105].

Let I be an instance of the PMP and let $\varepsilon > 0$. By slight abuse of notation whenever we write $O(\varepsilon)$ we mean $\ell \cdot \varepsilon$ for some positive constant ℓ. We call a task τ *small* if $\text{util}(\tau) \leq \varepsilon$, otherwise we call τ *big*. We define the sets I_{small} and I_{big} respectively. Our strategy is the following: We use a rounding method for computing an $(1 + \varepsilon)$-approximate solution for the tasks in I_{big}. Call this solution $ENUM(I_{\text{big}})$. For the tasks in I_{small} we use the First-Fit algorithm described above (on new machines). We obtain a solution using at

most $ENUM(I_{\text{big}}) + FF(I_{\text{small}})$ machines. We will show in Theorem 6.5 that either this solution or $FF(I)$ uses at most $(3/2 + O(\varepsilon))\,OPT + k$ machines.

Now we describe how to compute $ENUM(I_{\text{big}})$. For this we need some preparation in which we derive some interesting properties of the periodic maintenance problem.

Lemma 6.2. *Let K be a constant. Let $I = \{\tau_1, ..., \tau_K\}$ be a set of tasks. There is a polynomial time algorithm which determines whether the tasks in I can be scheduled on one machine.*

Proof. We explain how one can formulate the problem as an integer program (IP). Since here the number of tasks is bounded by a constant the number of variables of the IP will be bounded by a constant as well. Thus, we can use Lenstra's algorithm [68] to solve the problem in polynomial time.

For each task τ_j we introduce a variable $a_j \in \mathbb{Z}$ representing its start offset. For each variable a_j we introduce the constraints

$$0 \leq a_j \leq p_j - 1.$$

Then, for each pair of tasks $\tau_j, \tau_{j'}$ we need to ensure that none of their jobs collide. Since the jobs of τ_j and $\tau_{j'}$ are created at times in $a_j + p_j\mathbb{Z}$ and $a_{j'} + p_{j'}\mathbb{Z}$, respectively, elementary number theory [80] allow us to state the condition that no jobs collide as

$$a_j + f_j \not\equiv a_{j'} + f_{j'} \bmod \gcd(p_j, p_{j'})$$

for all $f_j, f_{j'} \in \mathbb{Z}$ with $0 \leq f_j \leq c_j - 1$ and $0 \leq f_{j'} \leq c_{j'} - 1$. The above statement is equivalent to

$$a_j - a_{j'} + f_j - f_{j'} \not\equiv 0 \bmod \gcd(p_j, p_{j'}).$$

We observe that the expression $(f_j - f_{j'})$ can attain all integral values in the interval $[-c_{j'} + 1, c_j - 1]$. Thus, the above condition is equivalent to the two conditions

$$a_j - a_{j'} \bmod \gcd(p_j, p_{j'}) \geq c_{j'} \tag{6.1}$$
$$a_j - a_{j'} \bmod \gcd(p_j, p_{j'}) \leq \gcd(p_j, p_{j'}) - c_j. \tag{6.2}$$

Since we are interested in an integer program (IP) we reformulate the mod-operator by introducing a variable $z_{j,j'} \in \mathbb{Z}$ and the conditions

$$0 \leq a_j - a_{j'} + z_{j,j'} \cdot \gcd(p_j, p_{j'}) \leq \gcd(p_j, p_{j'}) - 1. \tag{6.3}$$

Then, Inequalities (6.1) and (6.2) can be written as

$$a_j - a_{j'} + z_{j,j'} \cdot \gcd(p_j, p_{j'}) \geq c_{j'} \tag{6.4}$$
$$a_j - a_{j'} + z_{j,j'} \cdot \gcd(p_j, p_{j'}) \leq \gcd(p_j, p_{j'}) - c_j \tag{6.5}$$

Introducing the respective Inequalities (6.4) and (6.5) for each pair of tasks ensures that no two tasks collide. (Note that they imply Inequality 6.3). If the number of tasks is bounded by a constant we introduce only a constant number of variables, allowing Lenstra's algorithm [68] to run in polynomial time. □

Let a pair (c, p) consisting of an execution time c and a period length p to be a *type* of a task. In the following lemma we show that if the number of types of tasks is bounded by a constant and all tasks are big then the PMP is still polynomial time solvable.

Lemma 6.3. *Let K and $\varepsilon > 0$ be fixed constants. If there are at most K different types of tasks and for each task τ_j we have that* $\text{util}(\tau_j) \geq \varepsilon$ *then we can solve the periodic maintenance problem optimally in polynomial time.*

Proof. Since $\text{util}(\tau_j) \geq \varepsilon$ for all tasks τ_j we conclude that in any solution on each machine there can be at most $\lfloor 1/\varepsilon \rfloor$ tasks. Thus, there can be at most $R := \lfloor 1/\varepsilon \rfloor^K$ possible assignments of tasks to a machine. We refer to the latter as *configurations*. For each configuration we need to check whether it is feasible, i. e., whether there exist start offsets for the respective tasks which allow them to be processed on the same machine. Since we have only a constant number of tasks this can be done in polynomial time (see Lemma 6.2). Since we need at most $n = |I|$ machines in total there are at most n^R configurations. (For each configuration there are at most n machines with this configuration.) We can enumerate them in polynomial time. □

Using the above Lemma we can derive a PTAS for the tasks in I_{big}.

Lemma 6.4. *Let $\varepsilon > 0$. Let I be a set of tasks such that* $\text{util}(\tau_j) \geq \varepsilon$ *for each tasks $\tau_j \in I$. Assume that the number of different period lengths in I is bounded by a constant. Then there is a polynomial time algorithm which finds a solution which needs at most $(1 + \varepsilon)\, OPT(I)$ machines.*

Proof. For each period length q_r we define I^r to be the set of all tasks τ_j with $p_j = q_r$. For each period length q_r we order the tasks non-decreasingly according to their utilization. Then we partition the tasks from left to right into at most $K = \left\lceil \frac{2}{\varepsilon^2} \right\rceil$ groups. We do this such that all groups but the first one contain exactly $Q_r := \left\lfloor \varepsilon^2 \cdot |I^r| \right\rfloor$ tasks and the first group contains at most Q_r tasks (if $Q_r = 0$ we assign at most one task to each group). Then, we increase the execution time of each task such that its execution time (and thus its utilization) equals the execution time of the task with the largest execution time in its group. Denote by $\lceil I \rceil$ the resulting instance. In $\lceil I \rceil$ we have at most $k \cdot K$ types of tasks. Since we assumed k to be constant we have only a constant number of types, all with a utilization of at least ε. Using the algorithm described in Lemma 6.3 we can find an optimal solution for $\lceil I \rceil$. This solution yields a valid solution for I. In order to prove the claimed approximation guarantee we construct an instance $\lfloor I \rfloor$ in which we *decrease* the execution time of each task to the smallest execution time of a task in its group.

We know that $OPT(\lfloor I \rfloor) \leq OPT(I) \leq OPT(\lceil I \rceil)$. Also, the execution times of any task of a group is upper-bounded by the execution time of any task of the next higher group. Hence, if we remove all tasks from the largest group of $\lceil I \rceil$ for each period length, the remaining tasks of $\lceil I \rceil$ can be assigned to $OPT(\lfloor I \rfloor)$ machines using the solution $OPT(\lfloor I \rfloor)$ as a template. Hence, there

is a solution for $\lceil I \rceil$ using at most $OPT(\lfloor I \rfloor) + \sum_r Q_r$ machines. We conclude that

$$OPT(\lceil I \rceil) \leq OPT(\lfloor I \rfloor) + \sum_r Q_r \leq OPT(I) + n\varepsilon^2 \leq (1+\varepsilon)\, OPT(I).$$

Note that since $\text{util}(\tau_j) \geq \varepsilon$ we can use that $n\varepsilon \leq OPT(I)$. □

Now we are ready to describe the main algorithm. First, for the tasks in I_{big} we find a solution which needs at most $(1+\varepsilon)\, OPT(I_{\text{big}})$ machines, denoted by $ENUM(I_{\text{big}})$, using the algorithm described in Lemma 6.4. Observe that $OPT(I_{\text{big}}) \leq OPT(I)$. We put the machines aside which were used so far and we assign the tasks in I_{small} to new machines using the First-Fit algorithm described in Section 6.2.1. Then, we run the First-Fit algorithm directly with the set I of all tasks. We output the best among the two solutions $ENUM(I_{\text{big}}) + FF(I_{\text{small}})$ and $FF(I)$. Denote by $EFF(I)$ (enumeration first-fit) the number of needed machines.

Theorem 6.5. *Let I be a set of tasks and let $\varepsilon > 0$ as defined in the algorithm. Assume that the number of period lengths k is bounded by a constant. Then, $EFF(I)$ can be computed in polynomial time and it holds that $EFF(I) \leq (3/2 + O(\varepsilon))\, OPT(I) + k$.*

Proof. The computation of $FF(I)$ and $FF(I_{\text{small}})$ can clearly be done in polynomial time. Due to Lemma 6.4 the computation of $ENUM(I_{\text{big}})$ can also be done in polynomial time since k is bounded by a constant.

Now we want to prove the approximation ratio. First, we derive some bounds for the number of machines used by the two algorithms. For each period length q_r denote by I^r_{small} and I^r_{big} the small and big tasks with this period length. Using utilization bound arguments we have that

$$FF(I) < \sum_{\ell=1}^{k} \left(2 \cdot \text{util}(I^r_{\text{big}}) + \frac{\text{util}(I^r_{\text{small}})}{1-\varepsilon} + 1 \right) = 2 \cdot \text{util}(I_{\text{big}}) + \frac{\text{util}(I_{\text{small}})}{1-\varepsilon} + k.$$

Thus, if $\text{util}(I_{\text{small}}) \geq \text{util}(I_{\text{big}})$ then $FF(I) \leq (3/2 + O(\varepsilon))\, OPT(I) + k$ using $\text{util}(I)$ as a lower bound for $OPT(I)$. So now assume that

$$\text{util}(I_{\text{small}}) < \text{util}(I_{\text{big}}). \tag{6.6}$$

The total number of used machines is bounded by

$$ENUM(I_{\text{big}}) + FF(I_{\text{small}}) \leq ENUM(I_{\text{big}}) + \text{util}(I_{\text{small}})\,(1 + O(\varepsilon)) + k.$$

Now we distinguish two cases: If $\text{util}(I_{\text{small}}) \leq \frac{1}{2} ENUM(I_{\text{big}})$ this gives a bound of $(3/2 + O(\varepsilon))\, OPT(I) + k$ (since $ENUM(I_{\text{big}}) \leq OPT$). On the other hand, if $\text{util}(I_{\text{small}}) > \frac{1}{2} ENUM(I_{\text{big}})$ then

$$\begin{aligned} ENUM(I_{\text{big}}) + \text{util}(I_{\text{small}})\,(1 + O(\varepsilon)) &< \frac{3 \cdot \text{util}(I_{\text{small}})\,(1 + O(\varepsilon))}{\text{util}(I_{\text{small}}) + \text{util}(I_{\text{big}})} \cdot OPT(I) \\ &\leq \left(\frac{3}{2} + O(\varepsilon) \right) OPT(I) \end{aligned}$$

where the last inequality follows by Inequality 6.6. This concludes the proof. □

6.2.3 Complexity

In the previous sections we presented two approximation algorithms for the periodic maintenance problem. Both algorithms have an additive value k in the bound of the number of used machines. This raises the question whether one can design an approximation algorithm without this additive term, e. g., an algorithm which needs at most $\ell \cdot OPT$ machines for some constant ℓ. However, in this section we show that this is not possible. We prove that the PMP is NP-hard to approximate with a factor of $|I|^{1-\varepsilon}$ for any $\varepsilon > 0$ (if the number of period lengths k is part of the input). In particular, this rules out any constant factor approximation algorithm.

Our reduction heavily uses that the number of different period lengths in an instance is not bounded by any constant. In fact, in the reduction no two tasks have the same period length. Hence, for instances where the number of arising period lengths k is bounded by some constant there could still be better approximation algorithms. Recall that in Section 6.2.2 we presented an algorithm for this setting which needs at most $(\frac{3}{2} + \varepsilon)OPT + k$ machines. However, we show that there is no algorithm with an approximation guarantee of $(\frac{3}{2} - \varepsilon)OPT + k - 1$ for any $\varepsilon > 0$, unless $P = NP$, even if k is constant. Hence, our algorithm with the bound of $(\frac{3}{2} + \varepsilon)OPT + k$ is almost best possible.

First, we give our hardness result for the general PMP where the number of period lengths k is part of the input.

Theorem 6.6. *The periodic maintenance problem cannot be approximated within a factor of $n^{1-\varepsilon}$, for any constant $\varepsilon > 0$, unless $P = NP$.*

Proof. We show a reduction from Coloring. For unit execution times, a set of offsets is feasible if and only if $a_j + k_j \cdot p_j \neq a_{j'} + k_{j'} \cdot p_{j'}$ for each pair of tasks $\tau_j, \tau_{j'}$ and all $k_j, k_{j'} \in \mathbb{N}_0$. With elementary number theory [80] one can show that this is equivalent to $a_j \not\equiv a_{j'} \bmod \gcd(p_j, p_{j'})$. The reduction works as follows. Let the graph $G = (V, E)$ be an instance of Coloring. Let $\bar{E}$ be the complement of E, i. e., $\bar{E} := \{\{u, v\} \colon u, v \in V, \{u, v\} \notin E\}$. We choose pairwise different primes q_e for all $e \in \bar{E}$. This can be done in polynomial time with the sieve of Eratosthenes since the Prime Number Theorem guarantees $\Theta(x/\ln(x))$ primes among the first x natural numbers (see, e. g., [80]). For each node $v \in V$, we define a task $\tau_v = (c_v, p_v)$ with $c_v := 1$ and $p_v := \prod_{e \in \bar{E} \colon v \in e} q_e$. We denote by $\mathrm{red}(G) := \{\tau_v \colon v \in V\}$ the PMP instance obtained from the graph G using this construction.

The reduction has the following properties:

- For each edge $\{u, v\} \in E$ the tasks τ_u and τ_v cannot be assigned to the same machine.
- For an independent set $U \subseteq V$ the tasks $\{\tau_v \colon v \in U\}$ can be scheduled on one machine.

This implies that the tasks in red(G) can be scheduled on ℓ machines if and only if G can be colored with ℓ colors (for each $\ell \in \mathbb{N}$). Since for any $\varepsilon > 0$ the Coloring-problem cannot be approximated within a factor of $n^{1-\varepsilon}$ [112], the claim follows. □

Now we show that even for constant k there is no $(\frac{3}{2} - \varepsilon)OPT + k - 1$ approximation algorithm, unless $P = NP$. Note that this holds even if we restrict to harmonic instances.

Theorem 6.7. *Let k be a constant. Consider only instances of the harmonic periodic maintenance problem with at most k period lengths. In this setting, there is no $(\frac{3}{2} - \varepsilon)OPT + k - 1$ approximation algorithm for the problem for any $\varepsilon > 0$, unless $P = NP$.*

Proof. Assume that we have an $(\frac{3}{2} - \varepsilon)OPT + k - 1$ approximation algorithm. We show that then we can solve the Partition-problem (which is NP-hard, see [43]). Recall that Partition is the following problem: Given a set of integers $s_1, \ldots, s_n \in \mathbb{N}$, decide whether there is a subset $S \subseteq \{1, ..., n\}$ such that $\sum_{i \in S} s_i = \sum_{i \in [n] \setminus S} s_i$.

Consider an instance $s_1, \ldots, s_n$ of Partition. Let $B := \frac{1}{2} \sum_{i=1}^{n} s_i$. For each $i \in \{1, ..., n\}$, we define a task $\tau_i = (s_i, B)$. Observe that the instance $I = \{\tau_1, \ldots, \tau_n\}$ can be scheduled on 2 processors if the partition instance was a YES-instance. Otherwise we need at least 3 processors. Moreover, observe that $k = 1$ for instance I. Thus, the approximation algorithm can be used to decide Partition. □

6.3 Harmonic Periodic Maintenance Problem

The instances of the periodic maintenance problem arising at our industrial partner are harmonic, i. e., the period lengths divide each other pairwise. Therefore, this special case deserves particular investigation. Designing the tasks of a system in such a way allows much better usage of the computational power of the machines. For example, consider a non-harmonic instance with two tasks $\tau_1 = (1, p_1)$, $\tau_2 = (1, p_2)$ such that p_1 and p_2 are very large but $\gcd(p_1, p_2) = 1$. Then the two tasks cannot be processed on the same machine even though they both need only a very small share of the computational power of a machine. However, if p_1 and p_2 divide each other (and both are at least 2), we can safely assign both tasks on one machine.

It turns out that there are much better approximation algorithms possible for the harmonic PMP than for the general PMP. In particular, we present a 2-approximation algorithm (without the additive k-term as in the algorithm presented in Section 6.2.1). This is best possible: we show that not even asymptotically one can obtain an approximation algorithm with a better factor than 2, unless $P = NP$. Then we sketch how to design an asymptotic PTAS for the case that k is bounded by a constant. If even q_k/q_1 is bounded by a constant, we improve this even further.

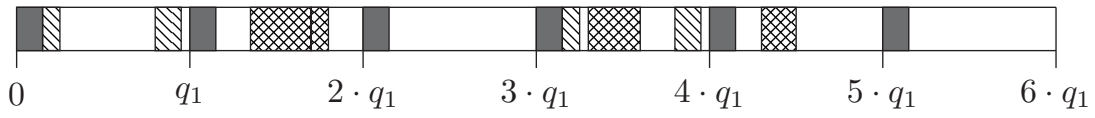


Figure 6.2: A schedule for a single machine. The solid jobs belong to tasks with period length q_1, the striped jobs to tasks with period length $q_2 = 3 \cdot q_1$, and the checkered jobs to tasks with period length $q_3 = 6 \cdot q_1$.

6.3.1 Bin-Trees

First, we introduce the structural concept of *bin-trees* which will be helpful when designing approximation algorithms. Also, the bin-trees are very useful for visualizing a schedule with its periodic tasks.

Assume that offsets a_j of tasks $\tau_j = (c_j, p_j)$, $j = 1, \ldots, n$, are given which form a feasible single-machine schedule. Then the resulting schedule repeats itself after the largest period q_k. W. l. o. g. we assume that the task τ_1 has the smallest arising period length, i. e., $p_1 = q_1$. By a simple shifting argument, we can assume w. l. o. g. that τ_1 has offset $a_1 = 0$. We observe that the jobs of the task τ_1 partitions the time-horizon $[0, q_k)$ into *bins* $[i \cdot q_1, (i+1) \cdot q_1)$ (see Figure 6.2). Consider two bins $B_i = [i \cdot q_1, (i+1) \cdot q_1)$ and $B_j = [j \cdot q_1, (j+1) \cdot q_1)$ such that $i \equiv j \bmod q_r/q_1$. As far as tasks with period length up to q_r are concerned, these bins look the same. As a shorthand, we write $B_i \equiv_r B_j$ when $i \equiv j \bmod q_r/q_1$.

We use this fact to describe schedules in a hierarchical structure we call *bin-tree* which we now define. The root of a *full bin-tree* is a node representing a bin containing all tasks of period length q_1. It has q_2/q_1 children, each of which represents a bin that contains all its parent's tasks and may contain additional tasks of period length q_2. We say that the root is of period q_1, and its children are of period q_2.

In general, a node B of period q_r contains only tasks of period length up to q_r. If $q_r < q_k$, it has q_{r+1}/q_r children, each of which is a node of period q_{r+1}. Each child of B represents a bin that contains all tasks of B and may contain additional tasks of period length q_{r+1}. Each scheduled task of period length q_r appears in a unique node of period q_r and in all children of that node.

As a consequence of this definition, there is a one-to-one correspondence between nodes of period q_r in the full bin-tree and equivalence classes of bins modulo the equivalence relation $\equiv_r$. Furthermore, the hierarchy of equivalence relations $\equiv_r$ $(r = 1, \ldots, k)$ corresponds to the hierarchy of the tree in the following way: If two nodes of period $\geq q_r$ have the same ancestor of period q_r, then their corresponding bins are equivalent modulo $\equiv_r$. In particular, the leafs of the bin-tree correspond to the bins of the schedule. Thus, we can freely convert between a feasible schedule in terms of task offsets and the corresponding bin-tree representation; see Figures 6.2 and 6.3.

The number of nodes in a full bin-tree is dominated by its leaves, of which there are q_k/q_1 many, so we cannot operate efficiently on full bin-trees. However, if a node of period q_r does not contain a task of period q_r, it is completely

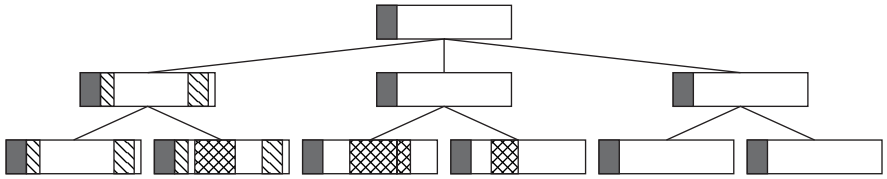


Figure 6.3: The full bin-tree corresponding to the schedule in Figure 6.2.

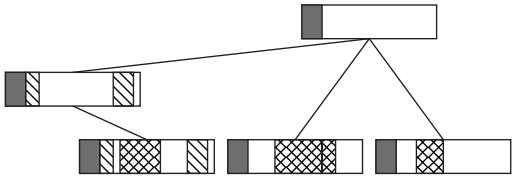


Figure 6.4: The compact form of the bin-tree in Figure 6.3.

determined by its parent. Therefore, we only need to store those nodes of the tree that introduce a new task to the schedule, see Figure 6.4 for an example. In this way we have a *compressed bin-tree* whose number of nodes is bounded by the number of tasks and that can be constructed in polynomial time.

6.3.2 First-Fit Algorithm

After having seen the concept of bin-trees it is clear that there is a very strong relation between the Bin-Packing problem and the harmonic PMP. For Bin-Packing one can show with a simple volume argument that the First-Fit algorithm is a 2-approximation. In particular, in Bin-Packing the total size of all items and OPT differ by at most a factor of two. For the harmonic PMP the respective counterpart of the sizes of the items would be the utilization of the tasks. However, here there are instances I where $\text{util}(I) \leq \varepsilon$ but $OPT(I) = n$. For example, consider an instance defined by $\tau_1 = (\varepsilon/n, 1)$ and $\tau_i = (p_{i-1}, p_{i-1} \cdot n/\varepsilon)$. In this instance, the total utilization is only ε and still no two tasks can be assigned to the same machine. Hence, here we need a more sophisticated analysis in order to show a bound on the approximation factor of First-Fit. To this end, we introduce the concept of *witnesses* that we will describe below. Intuitively, a witness task guarantees us in the analysis that a certain set of machines has a high average utilization. Using this concept, we show that for the harmonic PMP First-Fit is a 2-approximation algorithm

Now we describe and analyze the First-Fit algorithm for the harmonic PMP. We assume that the tasks are ordered non-descendingly by period length: for two tasks $\tau_j, \tau_{j'}$ with $j \leq j'$ we have that $p_j \leq p_{j'}$. In the sequel we use the following terminology: If a set of tasks is assigned to a machine, then the *type* of this machine is the smallest period length of these tasks. The First-Fit algorithm maintains a list $M_1, \ldots, M_\ell$ of open machines where M_i was opened before M_j if $i < j$. The algorithm is initialized with the empty list. Then, the algorithm proceeds as follows. For $j = 1, \ldots, n$:

1. Find the first machine on which τ_j fits and insert it into an arbitrary leaf of that machine's bin-tree which has enough space left to fit τ_j. Define the start offset of τ_j such that all idle time of the leaf is at its end.

2. If τ_j does not fit on any open machine, we open a new machine of type p_j and add τ_j to the root node of its bin-tree. Furthermore, to simplify the analysis later, we open a *second* new machine of type p_j. On this machine we schedule a dummy task with execution time 0 and period p_j.

Note that, because tasks are added in non-decreasing order of period lengths, one can easily determine whether τ_j can be inserted into the compact bin-tree: check all the leaves and all nodes to which a new leaf with period length p_j can be added.

The main result of this section is the following theorem.

Theorem 6.8. *The First-Fit algorithm is a 2-approximation algorithm for the harmonic PMP.*

Before we present the proof of Theorem 6.8 we will discuss some differences to the analysis of the First-Fit algorithm for Bin-Packing and motivate the concept of a *witness* that turns out to be very useful. With the following observation it can be shown that First-Fit for Bin-Packing is a 2-approximation: If First-Fit opens a new bin, then let α be the minimum load of all previously opened bins. This implies that the current item has a size of at least $1 - \alpha$. Also, if there are more than one open bins, all but one open bins must have a load of at least $\max\{\alpha, 1-\alpha\}$. The average load of the bins is thus at least $1/2$ (after inserting the new item).

Now suppose that the First-Fit algorithm for the harmonic PMP opens a new machine for task $\tau_j = (c_j, p_j)$. If the *type* q of some machine M with low utilization is smaller than the running time c_j of τ_j, then τ_j cannot be run on this machine, even though $\operatorname{util}(M)+\operatorname{util}(\tau_j)$ might still be smaller than 1. Thus, it may happen that there are many open machines with a low utilization. In particular, it is not true that the average utilization of the open machines is at least $1/2$. However, in the following lemma we derive a lower bound on the average load of the machines whose type is *compatible* with τ_j, where the set of types compatible to τ_j is denoted by $Q(j) := \{q \in Q \colon c_j \leq q \leq p_j\}$.

Lemma 6.9. *Suppose that the First-Fit algorithm cannot schedule τ_j on any open machine (and opens two new machines instead). Let $q \in Q(j)$ be a type compatible to τ_j, and let $M_1, \ldots, M_\ell$, $\ell > 0$, be the machines of type q that were open before the algorithm tried to assign τ_j. Then $\frac{1}{\ell}\sum_{i=1}^{\ell} \operatorname{util}(M_i) > \frac{1}{2}$.*

Proof. First observe that $\ell \geq 2$ because the First-Fit algorithm always opens two machines of the same type at a time. Consider the bin-trees corresponding to machines $M_1, \ldots, M_\ell$ when τ_j should be added. Note that the leaves of the trees are of period at most p_j. Let $\alpha > 0$ be the minimum fill ratio over all leaf-bins of the trees. If $\alpha > \frac{1}{2}$, then every bin is more than half filled and the claim follows.

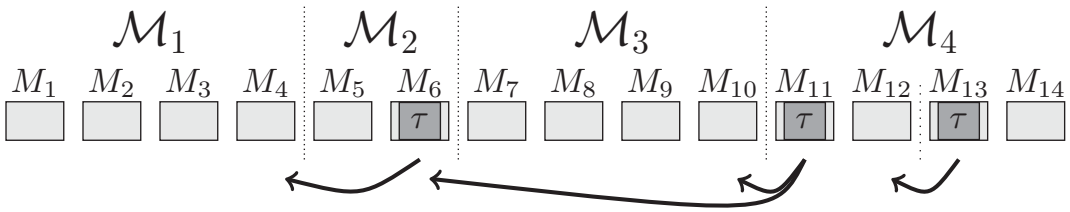


Figure 6.5: The situation in Lemma 6.10: Every group of machines but the last has a witness. The last two machines of the last group provide a witness for the remainder of the last group.

Thus, we can assume that $\alpha \leq \frac{1}{2}$. Let $\bar{B}$ be a leaf bin of fill ratio α, and let $\bar{M}$ be the machine $\bar{B}$ belongs to. Thus, the utilization of $\bar{M}$ is at least α. We will show that all leaf bins of the machines $\{M_1, \ldots, M_\ell\} \setminus \{\bar{M}\}$ have a fill ratio greater than $1 - \alpha$. This implies, in particular, that all machines other than $\bar{M}$ have a utilization greater than $1 - \alpha$. Since $\ell \geq 2$, the claim then follows by an averaging argument.

Let B be a leaf bin on a machine $M_i \neq \bar{M}$. Our goal is to show that the fill ratio of B is greater than $1 - \alpha$. There are two cases to consider: The First-Fit algorithm considers M_i *before* or *after* $\bar{M}$. If M_i is considered before $\bar{M}$, let τ be any task assigned to $\bar{M}$. Now consider the time when τ was assigned by First-Fit. At that time, either B or an ancestor of B was a leaf-bin. First-Fit tried to assign τ to B (or its ancestor) but failed. Now τ fills at most an α-fraction of a bin, which implies that the fill ratio of B (or its ancestor) must have been more than $1 - \alpha$, otherwise τ would have been packed there instead. Thus, B had fill ratio greater than $1 - \alpha$ at the time τ was assigned. For the case where M_i is considered after $\bar{M}$, we can analogously argue that a task in B could have been assigned to $\bar{B}$ (or an ancestor). □

Let τ_j be a task and let $\mathcal{M}^j$ be the set of machines that are open when First-Fit tries to assign τ_j. Let $Q' \subseteq Q(j)$ be a subset of the machine types compatible to τ_j and let $\mathcal{M}' \subseteq \mathcal{M}^j$ be the subset of machines whose type is in Q'. The above lemma implies that, if First-Fit opens two new machines when it tries to assign τ_j, then the average load of the machines in $\mathcal{M}'$ is at least $1/2$. We say that τ_j is a *witness* of $\mathcal{M}'$. In particular, if a period length $q < p_j$ is compatible with τ_j and if $\mathcal{M}_q$ denotes the machines of type q that are created by First-Fit, then τ_j is a witness of $\mathcal{M}_q$. We make use of the concept of witnesses in the following lemma.

Lemma 6.10. *Let $FF(I)$ denote the number of machines opened by the First-Fit algorithm. If for all $q \in Q$, $q < q_k$, the set $\mathcal{M}_q$ has a witness, then it holds that $FF(I) \leq 2 \cdot OPT(I)$.*

Proof. An illustration of the essential proof idea is given in Figure 6.5. Let $q \in Q$ with $q < q_k$, and let τ be a witness for $\mathcal{M}_q$. Then we can apply Lemma 6.9 to show that $\sum_{M \in \mathcal{M}_q} \text{util}(M) \geq \frac{1}{2}|\mathcal{M}_q|$. Now let $\mathcal{M}'$ be the set $\mathcal{M}_{q_k}$ without the two last machines that we call $\tilde{M}_1$ and $\tilde{M}_2$. Let τ be a task assigned to $\tilde{M}_1$.

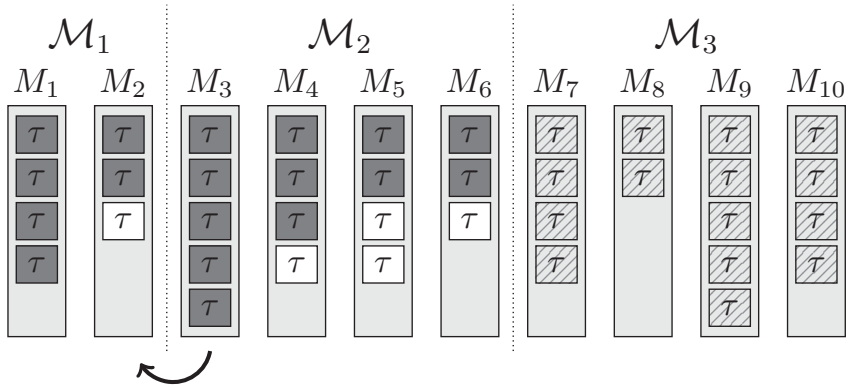


Figure 6.6: Possible situation in the proof of Theorem 6.8: The group $\mathcal{M}_2$ does not have a witness. The dark tasks form the set $\bar{I}$. Together with the white tasks, they form the set I'. The striped tasks form the set I''. One task on machine M_3 is a witness for $\mathcal{M}_1$.

Observe that τ is a witness for $\mathcal{M}'$. Thus, $\sum_{M\in\mathcal{M}'} \text{util}(M) \geq \frac{1}{2}|\mathcal{M}'|$. Hence we have $FF(I) - 2 \leq 2 \cdot \text{util}(I \setminus \{\tau\})$ which implies $FF(I) - 2 < 2 \cdot \text{util}(I) \leq 2 \cdot OPT(I)$. Since both $FF(I)$ and $2 \cdot OPT(I)$ are even, we conclude that $FF(I) \leq 2 \cdot OPT(I)$. □

If the special case of Lemma 6.10 does not apply, we can identify sub-instances that are pairwise independent of each other, yet cover all machines opened by First-Fit. We use this observation to prove Theorem 6.8 by induction; see also Figure 6.6.

Proof of Theorem 6.8. We prove the theorem by induction over k, the number of different period lengths. If $k = 1$, then the claim is obvious. Now assume that First-Fit is a 2-approximation for all instances with less than k period lengths. If for all $q \in Q$, $q < q_k$, the set $\mathcal{M}_q$ has a witness, again the claim follows directly with Lemma 6.10.

Thus, let $q \in Q$, $q < q_k$ be a period length such that $\mathcal{M}_q$ does not have a witness. We now partition the tasks into the set I' which contains all tasks which were assigned by First-Fit to a machine of type $q' \leq q$ and $I'' := I \setminus I'$. Moreover, let $\bar{I} := \{\tau_i \in I' : p_i \leq q\}$. Let τ_j be an arbitrary task in I''. Then q is not compatible with τ_j since otherwise τ_j would be a witness for $\mathcal{M}_q$. Thus, each task in I'' has a running time strictly larger than q. As the period lengths of all tasks in $\bar{I}$ are at most q, no task in $\bar{I}$ can be scheduled together with a task in I''. This shows that $\bar{I}$ and I'' are independent in the sense that $OPT(\bar{I}) + OPT(I'') = OPT(\bar{I} \cup I'') \leq OPT(I)$.

On the other hand, since First-Fit assigns each task of I' to a machine of type at most q and these must have been opened and assigned a type by a job in $\bar{I}$, we have $FF(\bar{I}) = FF(I')$ and $FF(I) = FF(\bar{I}) + FF(I'')$. Using the induction hypothesis, we get

$$\begin{aligned} FF(I) &= FF(\bar{I}) + FF(I'') \\ &\leq 2 \cdot OPT(\bar{I}) + 2 \cdot OPT(I'') \\ &\leq 2 \cdot OPT(I). \end{aligned}$$

This concludes the proof. □

6.3.3 Complexity

We have seen in the previous secion that for the harmonic PMP the First-Fit algorithm is a 2-approximation. It is known that First-Fit for the strongly related Bin-Packing problem is a 1.5-approximation algorithm if one first orders the items non-ascendingly by size [98]. However, for the harmonic PMP we prove that such an improvement is impossible unless $P = NP$. We prove that it is NP-hard to approximate the harmonic PMP with a factor of $2 - \varepsilon$, for any $\varepsilon > 0$. This holds even asymptotically. In particular, this shows that our 2-approximation algorithm for the harmonic PMP is best possible.

Theorem 6.11. *Unless $P = NP$, for the harmonic PMP there is no approximation algorithm which uses at most $(2 - \varepsilon)OPT + o(OPT)$ machines, for any $\varepsilon > 0$.*

Proof. The hardness result is established via a reduction from Bin-Packing and boosting. Assume we are given a Bin-Packing instance specified by a bin size $B \in \mathbb{N}$, items i with sizes $a_i \in \mathbb{N}$ and $a_i \leq B$, and an integer k. The question is whether there is a solution which uses at most k bins of size B. W. l. o. g. we assume that more than k items are given since otherwise the instance is obviously a YES-instance.

We reduce this problem to the following PMP instance I. The instance has has $n + 1$ tasks where $\tau_i := (a_i, kB + k)$, $i = 1, \ldots, n$ and $\tau_{n+1} := (1, B + 1)$. Observe that I is harmonic. The task τ_{n+1} creates k bins of "capacity" B each. Thus, we can schedule this set of tasks on one machine if and only if the Bin-Packing instance has a solution using at most k bins. This shows that, unless $P = NP$, there is no $(2 - \varepsilon)$-approximation algorithm for the harmonic PMP.

Now we want to boost this reduction to rule out any asymptotic approximation algorithm with a better approximation factor than 2. Assume we are given an algorithm which guarantees to use at most $\alpha OPT + f(OPT)$ machines, where $f(OPT) \in o(OPT)$. Let $\varepsilon > 0$. Then there is a constant M such that $f(OPT) \leq \frac{\varepsilon}{2} OPT$ for all $OPT \geq M$. Thus, for instances where the optimal solution is at least M, the algorithm provides an $(\alpha + \frac{\varepsilon}{2})$-approximation.

We show now how to boost the reduction provided above. This is done by *duplicating* the instance. For the instance I and any $\ell \in \mathbb{N}$, let $\ell \cdot I$ be a copy of I where each execution time and each period length is multiplied by the factor ℓ. Note that there is a correspondence between solutions to I and to $\ell \cdot I$. Moreover, since $q_k := 2B + 2$ is the largest period length of I and each task has an execution time of at least 1, no task of I can be scheduled together with a

task of $q_k \cdot I$. We conclude that I and $q_k \cdot I$ are independent in the sense that $OPT(I \cup q_k \cdot I) = OPT(I) + OPT(q_k \cdot I) = 2 \cdot OPT(I)$. Iterating this idea, we create M duplicates of I

$$I,\ q_k \cdot I,\ q_k^2 \cdot I, \ldots, q_k^M \cdot I$$

to obtain an instance I' with $OPT(I') = M \cdot OPT(I)$. In particular, we have that $OPT(I') = M$ if the Bin-Packing-instance is a YES-instance and it holds that $OPT(I') = 2M$ if the Bin-Packing-instance is a NO-instance. Applying the approximation algorithm to I' based on a YES-instance of Bin-Packing yields a solution with at most $(\alpha + \frac{\varepsilon}{2})M$ machines. Hence, $\alpha \geq 2 - \frac{\varepsilon}{2} > 2 - \varepsilon$ or $P = NP$. □

Finally, we would like to recall that Theorem 6.7 also holds for instance of the harmonic PMP. Hence, even if k is bounded by a constant, there is no $(\frac{3}{2} - \varepsilon)OPT + k - 1$ approximation algorithm for the harmonic PMP for any $\varepsilon > 0$, unless $P = NP$.

6.3.4 APTAS for Constant Number of Periods

As we have seen, for the harmonic PMP the 2-approximation algorithm cannot be improved, unless $P = NP$. This holds even asymptotically. In particular, there is no hope for an (asymptotic) PTAS. Our experience with real-world instances from our industrial partner, however, is that these instances often have only very few different period lengths. Indeed, for the case that the number of period lengths k is bounded by a constant there is an asymptotic PTAS. If even q_k/q_1 is bounded by a constant this can be improved even further. Since these results are not the main focus of this work we give only the respective theorems and sketch the key ideas of the algorithms. For more detailed descriptions we refer to [27].

Theorem 6.12. *Let $\varepsilon > 0$ and let k be bounded by a constant. There is a polynomial time algorithm that computes a solution using at most $(1 + \varepsilon)OPT(I) + k$ machines for the harmonic PMP.*

Proof. Although the high level idea is the same as for the APTAS for Bin-Packing [35], the more complex nature of the periodic maintenance problem imposes novel and interesting difficulties. In particular, we cannot simply classify tasks as *big* or *small*, since different bin sizes occur on different machines. Therefore, we employ a rounding procedure that does this classification relative to the size of the bin that executes the task, which makes the procedure significantly more involved. After the rounding, we determine the optimal solution for the big tasks by enumeration: we enumerate how many bins of what configuration arise in the solution. A configuration of a bin encodes the big tasks in this bin as well as some reserved space for small tasks. In order to determine whether these enumerated bins can be combined to an actual assignment of tasks to machines, we make non-trivial use of the concept of bin-trees. Finally,

we add the small tasks using First-Fit into the reserved spaces in the bins as well as on additional machines. All these steps have to be performed with extreme care not to introduce extra additive factors to the guarantee on the objective function. □

If not only k but also q_k/q_1 (which equals the maximum number of leaves of a bin-tree) is bounded by a constant, we can improve the APTAS above even further.

Theorem 6.13. *Let $\varepsilon > 0$ and assume that q_k/q_1 is bounded by a constant. There is a polynomial time algorithm that computes a solution using at most $(1+\varepsilon)OPT(I)+1$ machines for the harmonic PMP.*

Proof. The main difference to the algorithm above is that here we declare a task τ_j to be small if $c_j \leq \varepsilon q_1$ and big otherwise. Since q_k/q_1 is bounded by a constant we can still enumerate over the big tasks. Also, the rounding procedure is significantly simpler than in the APTAS above. Moreover, if we now assign the small jobs by First-Fit we can ensure a utilization of at least $1-\varepsilon$ in all but one machine (rather than in all but k machines). Combining these ingredients yields a polynomial time algorithm with the bounds stated in the theorem. □

6.4 Conclusion

In this chapter we studied the theoretical aspects of the periodic maintenance problem. Apart from that, we also investigated computational aspects of the problem [28]. There, it turns out that straightforward textbook approaches are not sufficient to solve the real-world instances of our industrial partner. However, using the theoretical insights presented in this chapter for the (practical relevant) harmonic case lead to a competitive IP-model. In particular, the bin-structure and the bin-trees are the key ingredients to design an IP-model that abstracts the actual schedules for the machines. Instead, we model the assignment of the tasks to bins without explicitly defining the exact start offsets of the tasks. The whole project is an example of a fruitful transfer of theoretical insights used in approximation algorithms to better computational methods for real-world problems.

From a theoretical perspective, for most of our algorithmic results we have a tightly matching NP-hardness lower bound. Hence, further improvements would require that $P = NP$. However, it might be possible to further generalize the harmonic case. Right now, we require that each pair of period lengths divides each other. This yields a chain of period lengths $q_1, q_2, ..., q_k$ such that $q_i | q_{i+1}$ for each i. A possible generalization would be to require only that the period lengths form a tree such that each period length q divides the period lengths of all its children. Also, it would be interesting what approximation factors can be achieved by pseudopolynomial algorithms. Even in the harmonic case the problem is still strongly NP-hard (due to the contained Bin-Packing problem) and hence optimal pseudopolynomial algorithms are unlikely to exist.

In Theorem 6.11 we showed that in the harmonic case not even a pseudopolynomial algorithm can have a better approximation factor than two (and the First-Fit algorithm achieves this factor). However, our reduction for the general case heavily uses numbers which are exponential in the size of the reduced Coloring instance. Hence, in that setting there could be pseudopolynomial approximation algorithms which perform much better than the factor $n^{1-\varepsilon}$ which we ruled out for polynomial time algorithms (assuming that $P \neq NP$).

Chapter 7

Scheduling on Unrelated Machines

7.1 Introduction

One of the most prominent open problems in machine scheduling is scheduling jobs on unrelated machines to minimize the makespan, denoted by $R||C_{\max}$ in the three-field notation. We are given n jobs, m machines, and processing times $p_{i,j}$ for each job j on each machine i. The goal is to assign the jobs to the machines to minimize the overall makespan, i. e., the time when the last machine finishes.

The complexity of the problem is due to the fact that in general, the $p_{i,j}$ can be arbitrarily, without any specific structure that one can exploit. For example, a job j may have a processing time of ε on some machine i, processing time 1 on some other machine i' and on yet another machine i'' job j cannot be processed at all. For another job j' the situation might be totally different, e. g., $p_{i,j'} = 1$ and $p_{i',j'} = p_{i'',j'} = \varepsilon$.

In a seminal work, a 2-approximation algorithm for the problem is presented by Lenstra, Shmoys, and Tardos [69]. The algorithm is based on a canonical linear program formulation. In the same paper, they proved that it is NP-hard to approximate the problem with a factor better than 3/2. The gap between 3/2 and 2 has persisted for more than 20 years, even though the problem is considered to be very important in the scheduling community. For instance, Schuurman and Woeginger [95] list it as one of ten most "vexing" open questions in the area of approximation algorithms for NP-hard scheduling problems. Also, it is listed as an open problem in the book by Williamson and Shmoys [109].

The best known approximation algorithms for this problem and its special cases are derived by linear programming techniques [26, 69, 104]. A special role plays the *configuration-LP*. It is the strongest linear program for the problem considered in the literature and it implicitly contains a vast class of inequalities. In fact, for the most relevant cases of $R||C_{\max}$ the best known approxima-

tion factors match the best known upper bounds on the integrality gap of the configuration-LP. For the restricted assignment case, the LP is even the only known linear program which yields the respective bound [104].

There are two interesting special cases of the problem: the restricted assignment case and the unrelated graph balancing case. In the *restricted assignment* case, for each job j there is a value p_j such that for all machines i we have that $p_{i,j} \in \{p_j, \infty\}$. In the *unrelated graph balancing* case each job can be assigned to at most two machines (but with possibly different processing times). These two cases are sort of perpendicular to each other. One main result of this chapter is the analysis of the configuration-LP for the general case of $R||C_{\max}$ and for the unrelated graph balancing case. For both cases, we show that the configuration-LP has an integrality gap of two and hence it cannot help to obtain a better approximation factor than 2.

A related problem which we also study in this chapter is the problem of scheduling jobs on unrelated machines with a different objective function. For each machine, we denote by its *load* the sum of the processing times of its jobs. The objective is to maximize the load of the machine with the minimum load. This can be understood as assigning items (jobs) to agents (machines). Each item j has a certain utility $p_{i,j}$ for each agent i. The goal is to assign the items as fair as possible by maximizing the minimum total utility of an agent. The problem can be subdivided into the same special cases as $R||C_{\max}$. For the most relevant cases, the best known approximation algorithms are derived via the configuration-LP [8, 9, 10, 19].

The results presented in this chapter are joint work with José Verschae [106].

7.1.1 The Minimum Makespan Problem

The problem to minimize the makespan on unrelated machines is considered to be an important problem in machine scheduling. In the sequel we revise the literature for the general problem and the already mentioned special cases.

General Setting. As mentioned above, in a seminal paper Lenstra et al. [69] present a 2-approximation algorithm and prove that the problem is NP-hard to approximate within a factor of $3/2 - \varepsilon$ for all $\varepsilon > 0$. Besides this paper, there has not been much progress on the approximation ratio for $R||C_{\max}$. Shchepin and Vakhania [96] give a more sophisticated rounding for the linear program by Lenstra et al. and improve the approximation guarantee to $2 - 1/m$, which is best possible among all rounding algorithms for this LP. On the other hand, Gairing, Monien, and Woclaw [41] propose a more efficient combinatorial 2-approximation algorithm based on unsplittable flow techniques. If the number of machines is bounded by a constant, Horowitz et al. [57] give a $(1 + \varepsilon)$-approximation algorithm. This is best possible since still in this setting the problem is NP-hard (follows from a straightforward reduction from Partition).

In the preemptive version of this problem, we are allowed to stop processing a job at an arbitrary time and resume it later, possibly on a different machine. In

contrast to the non-preemptive problem, Lawler and Labetoulle [63] introduce a polynomial time algorithm to compute an optimal preemptive schedule. Thus, it is possible to design an approximation algorithm for $R||C_{\max}$ by using the value of an optimal preemptive schedule as a lower bound. Shmoys and Tardos (cited as a personal communication in [72]), show that it is possible to obtain a 4-approximation algorithm using this method. Moreover, Correa, Skutella and Verschae [23] prove that this is best possible by proving that the *power of preemption*, i. e., the worst case ratio between the makespan of an optimal preemptive and non-preemptive schedule, equals 4.

Restricted Assignment. The best approximation algorithm for the restricted assignment problem known so far is the $(2-1/m)$-approximation algorithm that follows from the general setting of $R||C_{\max}$. As mentioned above, Svensson [104] shows how to estimate the optimal makespan within a factor $33/17+\varepsilon$ in polynomial time. In particular, he proves that in this setting the configuration-LP has an integrality gap of at most 33/17. However, no polynomial time rounding procedure is known.

There are further results for various special cases in the restricted assignment setting, depending on the structure of the jobs and the machines, see [71] for a survey. If all processing times are equal, Lin and Li [73] prove that the restricted assignment problem is solvable in polynomial time.

Restricted Graph Balancing. The restricted graph balancing case can be interpreted as a problem on an undirected graph. The nodes of the graph correspond to machines and the edges correspond to jobs. The endpoints of an edge associated to a job j are the machines on which j has finite processing time $p_j \in \mathbb{N}^+$. The objective is to find an orientation of the edges so as to minimize the maximum load of all nodes, where the load of a node is defined as the sum of processing time of its incoming edges (jobs). Notice that the graph may have loops and in that case the corresponding job must be assigned to one particular machine.

Ebenlendr et al. [26] give a 1.75-approximation algorithm based on a tighter version of the LP-relaxation by Lenstra et al. [69]. They strengthen this LP by adding inequalities that prohibit two large jobs to be simultaneously assigned to the same machine. Additionally to the 1.75-approximation algorithm for graph balancing, Ebenlendr et al. [26] also show that it is NP-hard to approximate this problem with a better factor than 3/2. This matches the lower bound for the more general problem $R||C_{\max}$. Furthermore, some special cases are studied. For example, it is known that if the underlying graph is a tree, the problem admits a PTAS. If the processing times are either 1 or 2, there is a (3/2)-approximation algorithm, which is best possible, unless $P = NP$. For these and more related results see [64] and the references therein.

There is not much known for the *unrelated* graph balancing problem, where the processing time of a job can be different on its two available machines. To the best of our knowledge, everything that is known about this problem

follows from results for the general case of $R||C_{\max}$. In this chapter we show that even for this special case the configuration-LP has an integrality gap of two. Hence, already for this case methods are needed which go beyond the pure configuration-LP.

7.1.2 The MaxMin-Allocation Problem

So far we considered the objective to minimize the maximum machine load. In the MaxMin-allocation problem the objective is somehow the opposite: We want to maximize the minimum machine load. The intuition for this objective function is that the machines correspond to agents and the jobs correspond to items that have to be assigned to the agents. Each item j has a certain utility $p_{i,j}$ for each agent i. The aim is to maximize the total utility of the agent with the least total utility.

Unrelated Machines. The MaxMin-allocation problem has drawn a lot of attention recently. For the general setting of unrelated machines Bansal and Sviridenko [12] show that the configuration-LP has an integrality gap of $\Omega(\sqrt{m})$. On the other hand, Asadpour and Saberi [10] show constructively that this is tight up to logarithmic factors yielding an algorithm with approximation ratio $O\left(\sqrt{m}\log^3 m\right)$. Relaxing the bound on the running time, Chakrabarty, Chuzhoy, and Khanna [19] present a poly-logarithmic approximation algorithm that runs in quasi-polynomial time. In terms of complexity, the best known result is that it is NP-hard to approximate the problem within a factor of $2-\varepsilon$ for any $\varepsilon > 0$ [14, 19]. For the special case that there are only two processing times arising in an instance (apart from zero), Golovin [47] gives an $O(\sqrt{n})$-approximation algorithm. He also provides an algorithm that assigns to at least a $(1-1/k)$ fraction of the machines a load of at least OPT/k. If the number of machines is bounded by a constant, the PTAS by Lenstra et al. [69] for a constant number of machines for $R||C_{\max}$ can easily be adapted to a PTAS for MaxMin-allocation. This is best possible since even for two machines MaxMin-allocation is NP-hard (straightforward reduction from Partition).

Restricted Assignment. Bansal et al. [12] study the case where every job has the same processing time on every machine that it can be assigned to. They show that the configuration-LP has an integrality gap of $O(\log\log m/\log\log\log m)$ in this setting. Based on this they present an algorithm with the same approximation ratio. The bound on the integrality gap was improved to $O(1)$ by Feige [34] and to 5 and subsequently to 4 by Asadpour, Feige, and Saberi [8, 9]. The former proof is non-constructive using the Lovász Local Lemma, the latter two are given by an (possibly exponential time) local search algorithm. However, Haeupler et al. [48] give a constructive version of the Lovász Local Lemma which – together with the the proof by Feige [34] – yields a polynomial time constant factor approximation algorithm.

	General	Unrelated graph balancing
General assignment	2	2
Restricted assignment	$\left[1.5, \frac{33}{17}\right]$ [26, 104]	[1.5, 1.75] [26]

Table 7.1: The integrality gap of the configuration-LP for $R||C_{\max}$ in the various settings.

Unrelated Graph Balancing. For the special case that every job can be assigned to at most two machines (but still with possibly different execution times on them) Bateni et al. [14] give a 4-approximation algorithm. This is improved by Chakrabarty et al. [19] who show that the configuration-LP has an integrality gap of 2 which yields a $(2+\varepsilon)$-approximation algorithm. Moreover, it is NP-hard to approximate even this special case with a better ratio than 2 [14, 19]. In fact, the proofs for this result use only jobs which have the same processing time on their two respective machines. Interestingly, the case that every job can be assigned to at most three machines is essentially equivalent to the general case [14].

7.1.3 Outline of the Chapter

In the following section, we first revise the algorithm by Lenstra et al. [69]. Also, we state our result that the configuration-LP has an integrality gap of 2 – even for unrelated graph balancing – and we discuss the implications of it. In particular, it implies that any set of cuts that involves only one machine per inequality cannot help to improve the integrality gap of the LP-relaxation of Lenstra et al. [69]. Recall that for the restricted assignment case the configuration-LP has an integrality gap of at most $33/17 < 2$ [104]. Hence, our result indicates that the core complexity of $R||C_{\max}$ lies in the unrelated graph balancing case rather than in the restricted assignment case. Even more, our instances use only processing times from the set $\{\varepsilon, 1, \infty\}$ for a small value ε.

We give our proof that the integrality gap of the configuration-LP is at least 2 in Section 7.3. Together with the algorithm by Lenstra et al. [69] this implies that the gap is exactly two. First, we give a family of instances of $R||C_{\max}$ where the integrality gap gets arbitrarily close to two. Then, we give a more involved construction that shows this result even for unrelated graph balancing. Table 7.1 shows an overview of the known values/ranges for the integrality gap of the configuration-LP for the various cases.

In Section 7.4 we study special cases for which we obtain better approximation factors than 2. In particular, we obtain a $(1 + 5/6)$-approximation algorithm for the special case of $R||C_{\max}$ where the processing times belong to the set $[\gamma, 10\gamma/3] \cup \{\infty\}$ for some $\gamma > 0$. In other words: the processing times of the jobs differ by at most a factor of $10/3$. Note that the strongest known NP-hardness reductions create instances with this property. Moreover, we show that there exists a $(2 - g/p_{\max})$-approximation algorithm, where g denotes the

greatest common divisor of the processing times, and $p_{\max}$ the largest finite processing time. This generalizes the result by Lin et al. [73], that states that the case where the processing times are either 1 or infinity is polynomially solvable.

We also give a 5/3-approximation algorithm for the case that an optimal solution assigns only a constant number of jobs to machines where they need more processing time than a 2/3 fraction of the makespan. We achieve the same approximation guarantee for the case that for all but $O(\log n)$ machines it is known a priori whether they execute such big jobs. These results yield necessary properties for an NP-hardness reduction which shows a non-approximability of 2 for $R||C_{\max}$.

In Section 7.5 we study certain cases of the MaxMin-allocation problem. Our main result is in the unrelated graph balancing setting, for which we present a simple purely combinatorial algorithm with quadratic running time which has a performance guarantee of 2. This improves on the LP-based $(2+\varepsilon)$-approximation algorithm by Chakrabarty et al. [19]. Their algorithm resorts to the ellipsoid method to approximately solve the configuration-LP. Note that this linear program has exponentially many variables and the separation problem of the dual is the KNAPSACK problem which can only be solved approximately. Our algorithm is significantly simpler to implement and moreover best possible, unless $P = NP$. Finally, we study what is achievable by allowing *half-integral* solutions, that is, solutions where we allow each job to be split into two halves. We give a polynomial time algorithm that computes a half integral solution whose objective value is within a factor of 2 of the optimal integral solution. Moreover, by loosing an extra factor of at most 2 in the objective we can transform this solution to one with at most $m/2$ fractional jobs. This result contrasts the integral version of the problem for which only an $O(\sqrt{m}\log^3 m)$-approximation algorithm is known.

Finally, in Section 7.6 we conclude and discuss further research directions.

7.2 LP-Based Approaches

In this section we revise the classical rounding procedure by Lenstra et al. [69] and elaborate on the implications of our results. The way we describe the techniques goes along the lines of [97] where Shmoys and Tardos generalize the problem to a setting with costs. In the sequel, we denote by J the set of jobs and M the set of machines of a given instance.

The LP-Relaxation. The IP-formulation used by Lenstra et al. [69] employs assignment variables $x_{i,j} \in \{0,1\}$ that denote whether job j is assigned to machine i. This formulation, which we denote by LST-IP, takes a target value for the makespan T (which will be determined later by a binary search) and does not use any objective function:

$$\text{(LST-IP)} \qquad \sum_{i \in M} x_{i,j} = 1 \qquad \forall j \in J \tag{7.1}$$

$$\sum_{j \in J} p_{i,j} \cdot x_{i,j} \leq T \qquad \forall i \in M$$

$$x_{i,j} = 0 \qquad \forall i, j : p_{i,j} > T$$

$$x_{i,j} \in \{0, 1\} \qquad \forall i \in M, j \in J.$$

The corresponding LP-relaxation of this IP, which we denote by LST-LP, can be obtained by replacing the integrality condition by $x_{i,j} \geq 0$. Let C_{LP} be the smallest integer value for T so that LST-LP is feasible, and let C^* be the optimal makespan of our instance (or equivalently, C^* is the smallest target makespan for which LST-IP is feasible). Thus, since the LP is feasible for $T = C^*$ we have that C_{LP} is a lower bound on C^*. Moreover, we can easily find C_{LP} in polynomial time with a binary search procedure.

Lenstra et al. [69] give a rounding procedure that takes a feasible solution of LST-LP with target makespan T and returns an integral solution with makespan at most $2T$. By taking $T = C_{LP} \leq C^*$ this yields a 2-approximation algorithm. The rounding, which we call *LST-rounding*, consists in interpreting the $x_{i,j}$ variables as a fractional matching in a bipartite graph, and then rounding this fractional matching to find an integral solution.

Now we describe the LST-rounding procedure in detail. Let the values $x_{i,j}$ denote a fractional solution of LST-LP. We interpret the solution as a fractional matching in the following bipartite graph $(A \cup B, E)$. The nodes in A contain one node for each job. We call these nodes the *job nodes*. For each machine i, we construct $k_i = \left\lceil \sum_{j \in J} x_{i,j} \right\rceil$ nodes in the set B, and call them $\{v_1^i, \ldots, v_{k_i}^i\}$. We refer to the nodes in B as the *machine nodes*.

For the edges E we construct a new fractional assignment of job nodes to machine nodes based on the assignment given by the $x_{i,j}$ variables. After that, whenever a job j has a fraction of $y_{i,\ell}^j$ assigned to a machine node v_ℓ^i we introduce an edge (j, v_ℓ^i) with weight $y_{i,\ell}^j$. For the construction of the new fractional assignment, consider a machine i and relabel the jobs so that $\{1, \ldots, n_i\}$ is the set of jobs that has some fraction assigned to i, i. e., $1 \leq j \leq n_i$ if and only if $x_{i,j} > 0$. Moreover, we assume w. l. o. g. that the jobs are in non-increasing order of processing times, $p_{i,1} \geq \ldots \geq p_{i,n_i}$. The fractional assignment of jobs to the nodes of i is performed in a greedy fashion: We iterate over the jobs in the above order. While doing this we maintain the invariant that all but one machine node of i have one (fractional) unit of jobs assigned to it and at most one machine node v_ℓ^i of i has α units of jobs assigned to it for some value $\alpha \in (0, 1)$. Consider a job j. If $p_{i,j} \leq \alpha$ then we assign j completely to v_ℓ^i. If $p_{i,j} > \alpha$ then we assign α units of j to v_ℓ^i and the remaining $p_{i,j} - \alpha$ units to a new machine node $v_{\ell+1}^i$ for i. Note that this maintains the above invariant. We do this procedure for each machine i and construct the edges E based on this assignment as described above.

The rounding procedure simply constructs a maximum matching in the bipartite graph $(A \cup B, E)$, that matches each job node to some machine node. Such a matching exists since the existence of a fractional matching that matches all job nodes implies the existence of a matching satisfying the same property (see, e. g., [93, Vol A]). Each job j will be matched to some node v_ℓ^i, and in this case we define an assignment variable $\bar{y}_{i,\ell}^j$ to be one and otherwise zero. Moreover, we define an assignment of jobs to machines as $\bar{x}_{ij} = \sum_{\ell=1}^{k_i} \bar{y}_{i,\ell}^j$ for all j and i. The following theorem shows a bound on the makespan of each machine due to this rounding procedure.

Theorem 7.1 ([97])**.** *Let $(x_{i,j})_{j\in J, i\in M}$ be a feasible solution of LST-LP with a target makespan T. Then, there exists a polynomial time rounding procedure that computes a binary solution $\{\bar{x}_{i,j}\}_{j\in J, i\in M}$ satisfying Equation* (7.1) *and*

$$\sum_{j\in J} \bar{x}_{i,j} p_{i,j} \leq T + \max\{p_{i,j} : j \in J \text{ and } x_{i,j} > 0\} \qquad \forall i \in M.$$

Proof. We give here a proof of the theorem which goes along the lines of the proof given in [97]. We show that the previously described rounding satisfies the properties of the theorem. Consider a machine i and a particular node v_ℓ^i for $\ell \in \{2, \ldots, k_i\}$. In the rest of the proof we omit the super-index i to shorten notation. Note that the processing time of the job assigned to node v_ℓ by the matching $\bar{y}$ is not larger than the processing time of any job which is assigned to $v_{\ell-1}$ in y. Hence,

$$\sum_{j\in J} \bar{y}_{i,\ell}^j \cdot p_{i,j} \leq \sum_{j\in J} y_{i,\ell-1}^j \cdot p_{i,j}.$$

for all $\ell \geq 2$. Upper bounding $\sum_{j\in J} \bar{y}_{(v_1,j)} \cdot p_{i,j}$ by $\max\{p_{i,j} : j \in J, x_{i,j} > 0\}$ we that

$$\begin{aligned}
\sum_{j\in J} \bar{x}_{i,j} \cdot p_{i,j} &= \sum_{\ell=1}^{k_i} \sum_{j\in J} \bar{y}_{i,\ell}^j \cdot p_{i,j} \\
&\leq \sum_{\ell=1}^{k_i-1} \sum_{j\in J} y_{i,\ell}^j \cdot p_{i,j} + \max\{p_{i,j} : j \in J, x_{i,j} > 0\} \\
&\leq \sum_{j\in J} x_{i,j} p_{i,j} + \max\{p_{i,j} : j \in J, x_{i,j} > 0\} \\
&\leq T + \max\{p_{i,j} : j \in J, x_{i,j} > 0\}.
\end{aligned}$$

□

Since $\max\{p_{i,j} : j \in J \text{ and } x_{i,j} > 0\} \leq T$ the previous theorem yields that the rounding procedure embedded in a binary search framework is a 2-approximation algorithm for $R||C_{\max}$.

Integrality Gaps and the Configuration-LP. Lenstra et al. [69] implicitly show that the rounding just given is best possible by means of the *integrality gap* of LST-LP. For an instance I of $R||C_{\max}$, let $C_{LP}(I)$ be the smallest integer value of T so that LST-LP is feasible, and let $C^*(I)$ the minimum makespan of this instance. Then the integrality gap of this LP is defined as $\sup_I C^*(I)/C_{LP}(I)$. It is easy to see that if C_{LP} is used as a lower bound for deriving an approximation algorithm then the integrality gap is the best possible approximation guarantee that we can show. Lenstra et al. [69] give an example showing that the integrality gap of LST-LP is arbitrarily close to 2, and thus the rounding procedure is best possible. This together with Theorem 7.1 implies that the integrality gap of LST-LP equals 2.

It is natural to ask whether adding a family of cuts can help to obtain a formulation with smaller integrality gap. Indeed, for special cases of our problem it has been shown that adding certain inequalities reduces the integrality gap. In particular, Ebenlendr et al. [26] show that adding the following inequalities to LST-LP yields an integrality gap of at most 1.75 in the graph balancing setting if each job has the same processing time on each of its at most two machines:

$$\sum_{j \in J: p_{i,j} > T/2} x_{i,j} \leq 1 \qquad \forall i \in M. \tag{7.2}$$

In this chapter we study whether it is possible to add similar cuts to strengthen the LP for the *unrelated* graph balancing problem or even for the general case of $R||C_{\max}$. For this we consider the *configuration-LP*, defined as follows. Let T be a target makespan, and define $\mathcal{C}_i(T)$ as the collection of all subsets of jobs with total processing time at most T, i. e.,

$$\mathcal{C}_i(T) := \left\{ C \subseteq J : \sum_{j \in C} p_{i,j} \leq T \right\}.$$

We introduce a variable $y_{i,C}$ for all $i \in M$ and $C \in \mathcal{C}_i(T)$, representing whether the jobs assigned to machine i equal exactly the jobs in C. The configuration-LP is defined as follows:

$$\begin{aligned}
\sum_{C \in \mathcal{C}_i(T)} y_{i,C} &= 1 && \forall i \in M \\
\sum_{i \in M} \sum_{C \in \mathcal{C}_i(T): j \in C} y_{i,C} &= 1 && \forall j \in J \\
y_{i,C} &\geq 0 && \forall i \in M, C \in \mathcal{C}_i(T).
\end{aligned}$$

It is not hard to see that an integral version of this LP is a formulation for $R||C_{\max}$. Also notice that the configuration-LP suffers from an exponential number of variables, and thus it is not possible to solve it directly in polynomial time. However, it is easy to show that the separation problem of the dual corresponds to an instance of Knapsack and thus we can solve the LP approximately in polynomial time. More precisely, given a target makespan T there is

a polynomial time algorithm that either asserts that the configuration-LP is infeasible or computes a solution which uses only configurations whose makespan is at most $(1+\varepsilon)T$, for any constant $\varepsilon > 0$ [104]. The following result, which will be proven in the next section, shows that the integrality gap of this formulation is as large as the integrality gap of LST-LP, even for the unrelated graph balancing case.

Theorem 7.2. *The configuration-LP for the unrelated graph balancing problem has an integrality gap of 2.*

A solution $(y_{i,C})$ of the configuration-LP yields a feasible solution to LST-LP with the same target makespan by using the following formula

$$x_{i,j} = \sum_{C \in \mathcal{C}_i(T): C \ni j} y_{i,C} \qquad \forall i \in M, j \in J. \tag{7.3}$$

This implies that the integrality gap of the configuration-LP is not larger than the integrality gap of LST-LP, and thus it is at most 2. On the other hand, there are solutions to LST-LP that do not have corresponding feasible solutions to the configuration-LP. For example, consider an instance with three jobs and two machines, where $p_{i,j} = 1$ for all jobs j and machines i. If we have a target makespan $T = 3/2$, it is easy to see that LST-LP is feasible, but the solution space of the configuration-LP is empty for any $T < 2$.

In the sequel we elaborate on the relation of the two LPs by giving a formulation in the space with $x_{i,j}$ variables that is equivalent to the configuration-LP. For any set $S \in \mathbb{R}^n$ we define $\operatorname{conv}\{S\}$ to be its convex closure.

Proposition 7.3. *Let $x^C \in \{0,1\}^J$ be the characteristic vector of a configuration $C \in \mathcal{C}_i(T)$, i. e., x^C_j is one if $j \in C$ and zero otherwise. The feasibility of the configuration-LP is equivalent to the feasibility of the linear program defined by Equations* (7.1) *and*

$$(x_{i,j})_{j \in J} \in \operatorname{conv}\{x^C : C \in \mathcal{C}_i(T)\} \qquad \forall i \in M. \tag{7.4}$$

Proof. Let $(x_{i,j})_{i \in M, j \in J}$ be a solution satisfying (7.1) and (7.4) for a given T. We show that the configuration-LP is feasible for the same value of T. Indeed, $(x_{i,j})_{j \in J}$ is a convex combination of vectors in $\{x^C : C \in \mathcal{C}_i(T)\}$, and thus

$$(x_{i,j})_{j \in J} = \sum_{C \in \mathcal{C}_i(T)} y_{i,C} \cdot x^C,$$

for some values $y_{i,C} \geq 0$ such that $\sum_{C \in \mathcal{C}_i(T)} y_{i,C} = 1$. Moreover, for each $j \in J$,

$$1 = \sum_{i \in M} x_{i,j} = \sum_{i \in M} \sum_{C \in \mathcal{C}_i(T)} y_{i,C} \cdot x^C_j = \sum_{i \in M} \sum_{C \in \mathcal{C}_i(T): C \ni j} y_{i,C}.$$

This shows that the values $y_{i,C}$ give a solution to the configuration-LP. The converse implication follows from reversing the just given argument. □

The last proposition implies that adding any family of cuts to LST-LP that does not remove any vector of the form $(x^{C_1}, \ldots, x^{C_m}) \in \mathbb{R}^{n \cdot m}$, where $C_i \in \mathcal{C}_i(T)$, cannot help to reduce the integrality gap of the linear relaxation. As an example of the implications of this proposition, we note that that adding the cuts given by Inequality 7.2 does not help to diminish the integrality gap of LST-LP for unrelated graph balancing. To give another example, a generalization of these cuts given by $\sum_{j:p_{i,j}>T/k} x_{i,j} \leq k-1$ for each machine i and each $k \in \mathbb{N}$ does not help to diminish the gap either.

7.3 Integrality Gap of the Configuration-LP

We have seen in the previous section that the configuration-LP implicitly contains a vast class of linear cuts. Hence, it is at least as strong (in terms of its integrality gap) as any linear program that contains any subset of these cuts. However, in this section we prove that the configuration-LP has an integrality gap of 2. This implies that even all the cuts that are contained in the configuration-LP are not enough to construct an algorithm with a better approximation factor than 2.

Then we show that even for the special case of unrelated graph balancing the configuration-LP has an integrality gap of 2. This is somehow surprising: if one additionally requires that each job has the same processing time on its two machines then Ebenlendr et al. [26] implicitly proved that the configuration-LP has an integrality gap between 1.5 and 1.75. Hence, we demonstrate that the property that a job can have different processing times on different machines makes the problem significantly harder.

7.3.1 Integrality Gap of the Configuration-LP

We describe a family of instances I_k for the general $R||C_{max}$-problem such that the configuration-LP has an integrality gap of $2 - \frac{1}{k}$ for each instance I_k. Since we can choose k arbitrarily large this proves an integrality gap of 2 for the configuration-LP. The construction we present here is significantly simpler than the construction for unrelated graph balancing which we will present in Section 7.3.2.

Let $k \in \mathbb{N}$. The instance I_k has $2k$ machines $m_1, m_1', m_2, m_2', ..., m_k, m_k'$. For every pair of machines m_i, m_i' there are k jobs $j_i^1, j_i^2, ..., j_i^k$ which have processing time $\frac{1}{k}$ on m_i, processing time 1 on m_i', and processing time ∞ on any other machine. Finally, there is one job j_{big} which has processing time 1 on any machine m_i and ∞ on any machine m_i'.

To evaluate the integrality gap of the configuration-LP on I_k we first lower-bound the optimal makespan.

Lemma 7.4. *Every integral solution for I_k has a makespan of at least $2 - \frac{1}{k}$.*

Proof. Consider an integral solution for I_k and assume its makespan is less than 2. Let m_i be the machine that j_{big} is assigned to. At most one of the

jobs $j_i^1, j_i^2, ..., j_i^k$ is assigned to m_i' and the other $k-1$ jobs are assigned to m_i. Hence, machine m_i has a makespan of $1 + (k-1) \cdot \frac{1}{k} = 2 - \frac{1}{k}$. □

Now let us study the configurations for the different machines. Since we want to show an integrality gap of $2-\frac{1}{k}$ we consider only configurations with makespan at most 1 (we will see that these configurations suffice to find a feasible solution for the LP). Also, we consider only maximal configurations, i. e., configurations whose jobs are not contained in another configuration on the respective machine. For each machine m_i there are two maximal configurations: take all jobs j_i^ℓ or only j_{big}. We call the former configuration the *small configuration* and the latter the *big configuration*. For each machine m_i' there are k maximal configurations: take only job j_i^ℓ for $1 \leq \ell \leq k$.

Lemma 7.5. *There is a solution of the configuration-LP for I_k that uses only configurations with makespan 1.*

Proof. We assign every machine m_i a ratio of $\frac{1}{k}$ of the big configuration and a ratio of $1 - \frac{1}{k}$ of the small configuration. Note that this assigns the job j_{big} completely and every job j_i^ℓ is assigned to an extent of $1 - \frac{1}{k}$. We assign every machine m_i' a ratio of $\frac{1}{k}$ of each of its k configurations. Hence, also every job j_i^ℓ is now fully assigned. This assigns every job completely. □

Knowing that the optimal makespan for I_k is at least $2 - \frac{1}{k}$ and there is a solution for the configuration-LP using only configurations with makespan 1, we obtain the following theorem.

Theorem 7.6. *The configuration-LP for $R||C_{\max}$ has an integrality gap of at least $2 - \frac{1}{k}$ for instances such that $p_{i,j} \in \{\frac{1}{k}, 1, \infty\}$ for all machines i and all jobs j.*

The bound of $2 - \frac{1}{k}$ is actually tight for instance where all $p_{i,j} \in \{\frac{1}{k}, 1, \infty\}$ as the following proposition shows.

Proposition 7.7. *The integrality gap of the configuration-LP for the $R||C_{\max}$ for instances with $p_{i,j} \in \{\frac{1}{k}, 1, \infty\}$ is bounded by $2 - \frac{1}{k}$.*

Proof. Let I be an instances of $R||C_{\max}$ such that $p_{i,j} \in \{\frac{1}{k}, 1, \infty\}$ for all machines i and all jobs j. Assume that there is a feasible solution for the configuration-LP for I using only configurations with a makespan of at most T. If $T < 1$ then the LST-rounding procedure will yield a solution with makespan at most $T + \frac{1}{k}$. So now assume that $T = 1 + \frac{\ell}{k}$ for some integer ℓ (other values for the makespan cannot arise). We perform the LST-rounding and analyze the resulting makespan. Consider a fixed machine i. We call a job j *big* if $p_{i,j} = 1$ and *small* if $p_{i,j} = \frac{1}{k}$. We call a configuration *big* if it contains a big job and *small* otherwise. We can assume that the configuration-LP assigned y_{big} units of big configurations to i and y_{small} units of small configurations. W. l. o. g. we assume that each assigned big configuration contains one big job and ℓ small

jobs. In the rounding procedure $y_{\text{big}} \cdot (\ell+1) + y_{\text{small}} \cdot (\ell+k)$ vertices are introduced for i. After the rounding at most one of them can have a big job assigned to it. Hence, the total makespan of i is bounded by

$$\begin{aligned} 1 + \frac{1}{k}\left(y_{\text{big}}\,(\ell+1) + y_{\text{small}}(\ell+k) - 1\right) &= 1 + y_{\text{big}} \cdot \frac{\ell+1}{k} + y_{\text{small}} \cdot \frac{\ell+k}{k} - \frac{1}{k} \\ &\leq 2 + \frac{\ell-1}{k}. \end{aligned}$$

This yields an integrality gap of $\left(2 + \frac{\ell-1}{k}\right) / \left(1 + \frac{\ell}{k}\right)$. The integrality gap becomes maximal for $\ell = 0$. □

We will generalize the above proposition later in Theorem 7.12.

7.3.2 Integrality Gap for Unrelated Graph Balancing

Now we improve the result from the previous section and show that even for unrestricted graph balancing the integrality gap of the configuration-LP is 2. We construct a family of instances I_k such that $p_{i,j} \in \{\frac{1}{k}, 1, \infty\}$ for each machine i and each job j for some integer k. We will show that for I_k there is a solution of the configuration-LP which uses only configurations with makespan $1 + \frac{1}{k}$. However, every integral solution for I_k requires a makespan of at least $2 - \frac{1}{k}$.

Let $k \in \mathbb{N}$ and let N be the minimum integer such that $k^N/(k-1)^{N+1} \geq \frac{1}{2}$. Consider two k-ary trees of height $N-1$, i. e., two trees of height $N-1$ in which apart from the leaves every vertex has k children. For every leaf v, we introduce another vertex v' and k edges between v and v'. (Hence, v is no longer a leaf.) Hence, the resulting "tree" has height N.

Based on this, we describe our instance of unrelated graph balancing. For each vertex v we introduce a machine m_v. For each edge $e = \{u, v\}$ we introduce a job j_e. Assume that u is closer to the root than v. We define that j_e has processing time $\frac{1}{k}$ on machine m_u, processing time 1 on machine m_v, and that it cannot be scheduled on any other machine. Finally, let $m_r^{(1)}$ and $m_r^{(2)}$ denote the two machines corresponding to the two root vertices. We introduce a job j_{big} which has processing time 1 on $m_r^{(1)}$ and $m_r^{(2)}$. Denote by I_k the resulting instance. See Figure 7.1 for a sketch.

Similarly to Section 7.3.1 we evaluate the optimal makespan for I_k and then the smallest makespan for which the configuration-LP is feasible. As mentioned before, we claim that any integral solution for I_k has a makespan of at least $2 - \frac{1}{k}$. We prove this in the following lemma.

Lemma 7.8. *Any integral solution for I_k has a makespan of at least $2 - \frac{1}{k}$.*

Proof. Assume that we are given an integral solution for I_k which has a strictly smaller makespan than 2. W. l. o. g. assume that j_{big} is assigned to machine $m_r^{(1)}$. Since the makespan of our solution is strictly less than 2 at most $k-1$ jobs with processing time $\frac{1}{k}$ can be assigned to $m_r^{(1)}$. Hence, there is an edge e adjacent

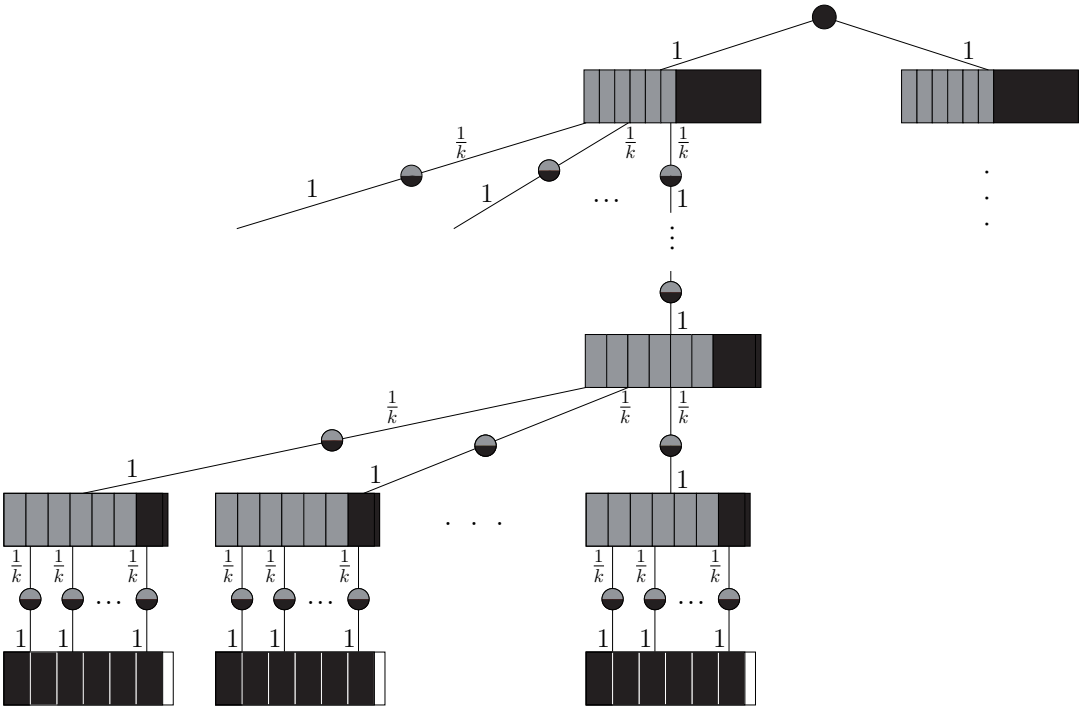


Figure 7.1: A sketch of the construction for the instance of unrelated graph balancing with an integrality gap of $2 - O(\frac{1}{k})$. The jobs on the machines correspond to the fractional solution of the configuration-LP for this instance with makespan $T = 1 + \frac{1}{k}$.

to the root r of the first tree such that j_e is *not* assigned to r. Thus, j_e must be assigned to the machine corresponding to the other vertex that e is adjacent to. We iterate the argument. Eventually, we have that there must be a vertex v of height 1 and a corresponding machine m_v which has a job j with processing time 1 assigned to it. Recall that our solution has a makespan strictly less than 2. Hence, at most one job can be assigned to machine $m_{v'}$ where v' is the child vertex of v. Thus, $k-1$ jobs with processing time $\frac{1}{k}$ are assigned to m_v. Together with j this gives a makespan of $1+(k-1)\frac{1}{k} = 2-\frac{1}{k}$ on machine m_v. □

Now we want to show that there is a feasible solution for the configuration-LP for I_k which uses only configurations with makespan $1 + \frac{1}{k}$. To this end, we introduce the concept of *j-α-solution* for the configuration-LP. We call j-α-solution a solution for the configuration-LP whose right hand side is modified as follows: the job j does not need to be fully assigned but only to an extent of $\alpha \leq 1$. I. e., instead of the equality

$$\sum_{i \in M} \sum_{C \in \mathcal{C}_i(T) : C \ni j} y_{i,C} = 1$$

we have the inequality

$$\sum_{i \in M} \sum_{C \in \mathcal{C}_i(T) : j \in C} y_{i,C} \geq \alpha.$$

For our purposes, we define the *height* of a vertex to be its distance to a leaf vertex. For any $h \in \mathbb{N}$ denote by $I_k^{(h)}$ a subinstance of I_k defined as follows: Take a vertex v of height h and consider the subtree $T(v)$ rooted at v. For the subinstance $I_k^{(h)}$ we take all machines and jobs which correspond to vertices and edges in $T(v)$. (Note that since our construction is symmetric it does not matter which vertex of height h we take.) Additionally, we take the job which has processing time 1 on m_v. We denote the latter by $j^{(h)}$.

We prove inductively that there are j-$\alpha^{(h)}$-solutions for the subinstances $I_k^{(h)}$ for values $\alpha^{(h)}$ which depend only on h. These values $\alpha^{(h)}$ increase for increasing h. The important point is that $\alpha^{(N)} \geq \frac{1}{2}$. Hence there are solutions for the configuration-LP which distribute j_{big} on the two machines $m_r^{(1)}$ and $m_r^{(2)}$ (which correspond to the two root vertices).

The following lemma gives the base case of the induction.

Lemma 7.9. *There is a $j^{(1)}$-$\frac{1}{k-1}$-solution of the configuration-LP for $I_k^{(1)}$ which uses only configurations with makespan at most $1 + \frac{1}{k}$.*

Proof. Let m_v be the machine in $I_k^{(1)}$ which corresponds to the root of $I_k^{(1)}$. Similarly, let $m_{v'}$ denote the machine which corresponds to the leaf v'. For $\ell \in \{1, ..., k\}$ let $j_\ell^{(0)}$ be the jobs which have processing time 1 on $m_{v'}$ and processing time $\frac{1}{k}$ on m_v.

For $m_{v'}$ the configurations with makespan at most $1+\frac{1}{k}$ are $C_\ell := \left\{j_\ell^{(0)}\right\}$ for each $\ell \in \{1, ..., k\}$. We define $y_{m_{v'}, C_\ell} := \frac{1}{k}$ for each ℓ. Hence, for each job $j_\ell^{(0)}$ a fraction of $\frac{k-1}{k}$ remains unassigned. For machine m_v there are the following (maximal) configurations: $C_{\text{small}} := \left\{j_1^{(0)}, ..., j_k^{(0)}\right\}$ and $C_{\text{big}}^\ell := \left\{j^{(1)}, j_\ell^{(0)}\right\}$ for each $\ell \in \{1, ..., k\}$. We define $y_{m_v, C_{\text{big}}^\ell} := \frac{1}{k(k-1)}$ for each ℓ and $y_{m_v, C_{\text{small}}} := 1 - \frac{1}{k-1}$. This assigns each job $j_\ell^{(0)}$ completely and the job $j^{(1)}$ to an extent of $k \cdot \frac{1}{k(k-1)} = \frac{1}{k-1}$. □

After having proven the base case, the following lemma yields the inductive step.

Lemma 7.10. *Assume that there exists a $j^{(n)}$-$\left(k^n/(k-1)^{n+1}\right)$-solution of the configuration-LP for $I_k^{(n)}$ which uses only configurations with makespan at most $1+\frac{1}{k}$. Then, there is a $j^{(n+1)}$-$\left(k^{n+1}/(k-1)^{n+2}\right)$-solution of the configuration-LP for $I_k^{(n+1)}$ which uses only configurations with makespan at most $1 + \frac{1}{k}$.*

Proof. Note that $I_k^{(n+1)}$ consists of k copies of $I_k^{(n)}$, one additional machine and one additional job. Denote by m_v the additional machine (which forms the "root" of $I_k^{(n+1)}$). Recall that $j^{(n+1)}$ is the (additional) job that can be assigned to m_v but to no other machine in $I_k^{(n+1)}$. For $\ell \in \{1, ..., k\}$ let $j_\ell^{(n)}$ be the jobs which have processing time $\frac{1}{k}$ on m_v.

Inside of the copies of $I_k^{(n)}$ we use the solution defined in the induction hypothesis. Hence, each job $j_\ell^{(n)}$ is already assigned to an extent of $\left(k^n/(k-1)^{n+1}\right)$. Like in Lemma 7.9 the (maximal) configurations for m_v are given by $C_{\text{small}} := \left\{j_1^{(n)}, ..., j_k^{(n)}\right\}$ and $C_{\text{big}}^\ell := \left\{j^{(n+1)}, j_\ell^{(n)}\right\}$ for each $\ell \in \{1, ..., k\}$. We define $y_{m_v, C_{\text{big}}^\ell} := k^n/(k-1)^{n+2}$ for each ℓ and $y_{m_v, C_{\text{small}}} := 1 - k^{n+1}/(k-1)^{n+2}$. This assigns each job $j_\ell^{(n)}$ completely and the job $j^{(n+1)}$ is assigned to an extent of $k \cdot k^n/(k-1)^{n+2} = k^{n+1}/(k-1)^{n+2}$. □

After this preparation we are ready to prove that there is a feasible solution for the configuration-LP for I_k.

Lemma 7.11. *There is a solution of the configuration-LP for I_k which uses only configurations with a makespan of at most $1 + \frac{1}{k}$.*

Proof. Recall that the two k-ary trees from the construction of I_k – together with the additional vertices – have height N such that $k^N/(k-1)^{N+1} \geq \frac{1}{2}$. Hence, there are j_{big}-$\frac{1}{2}$-solutions for each of the two subinstances $I_k^{(N)}$ which use only configurations with a makespan of at most $1 + \frac{1}{k}$. This proves the claim. □

Now our main theorem follows from the previous lemmas (theorem restated).

Theorem 7.2. *The configuration-LP for the unrelated graph balancing problem has an integrality gap of 2.*

Proof. Lemmas 7.8 and 7.11 imply that for the instance I_k the integrality gap of the configuration-LP is at least $(2 - \frac{1}{k})/(1 + \frac{1}{k})$. The claim follows since we can choose k arbitrarily large. The upper bound of 2 follows from [69]. □

7.4 Cases with Better Approximation Factors

It has been open for a long time whether the approximation ratio of 2 for $R||C_{\max}$ by Lenstra et al. [69] can be improved. Our results from Section 7.3 can be seen as an indicator that this is not possible, unless $P = NP$. In particular, recall that the best known approximation factors for the most relevant cases of $R||C_{\max}$ and MaxMin-allocation are implied by the configuration-LP (or can alternatively be derived from the integrality gap of the configuration-LP).

In this section we identify classes of instances of $R||C_{\max}$ for which a better approximation factor than 2 is possible. This can be understood as a guideline of properties that a NP-hardness reduction must fulfill to rule out a better approximation factor than 2.

7.4.1 Bounded GCD of Processing Times

The inapproximability results for $R||C_{\max}$ given in [26, 69] use only jobs j such that $p_{i,j} \in \{1, 2, 3, \infty\}$ for all machines i. We show now that for classes of

instances which use only a finite set of processing times, there exists an approximation algorithm with a performance guarantee which is strictly better than 2. This implies that NP-hardness reductions which rule out approximation algorithms with a ratio of $2-\varepsilon$ need an infinite set of processing times for the jobs.

Theorem 7.12. *There exists a $(2-\alpha)$-approximation algorithm for the problem of minimizing makespan on unrelated machines, where $\alpha = \gcd\{p_{i,j} | i \in M, j \in J, p_{i,j} < \infty\} / \max\{p_{i,j} | i \in M, j \in J, p_{i,j} < \infty\}$ for a given instance.*

Proof. We give a slighty strengthened analysis of the 2-approximation algorithm by Lenstra et al. [69]. Let $g := \gcd\{p_{i,j} | i \in M, j \in J, p_{i,j} < \infty\}$ and $P := \max\{p_{i,j} | i \in M, j \in J, p_{i,j} < \infty\}$. Note that the optimal makespan of our instance is a multiple of g, and therefore we can restrict our target makespan T to be of the form $k \cdot g$ with $k \in \mathbb{N}$. Let T^* be the target makespan defined as the smallest multiple of g that yields a feasible solution to LST-LP (can be computed by a binary search). Assume we have computed a fractional solution for LST-LP with target makespan T^*. We apply LST-rounding to this fractional solution. Being a little more careful in the proof of Theorem 7.1 we reason that

$$\begin{aligned}
\sum_{j \in J} \bar{x}_{i,j} \cdot p_{i,j} &= \sum_{\ell=1}^{k_i} \sum_{j \in J} \bar{y}_{i,\ell}^{j} \cdot p_{i,j} \\
&\leq \sum_{\ell=1}^{k_i - 1} \sum_{j \in J} y_{i,\ell}^{j} \cdot p_{i,j} + \max\{p_{i,j} : j \in J, x_{i,j} > 0\} \\
&< \sum_{j \in J} x_{i,j} p_{i,j} + \max\{p_{i,j} : j \in J, x_{i,j} > 0\} \\
&\leq T + \max\{p_{i,j} : j \in J, x_{i,j} > 0\}
\end{aligned}$$

and hence $\sum_{j \in J} \bar{x}_{i,j} \cdot p_{i,j} < T^* + P$. Since $\sum_{j \in J} \bar{x}_{i,j} \cdot p_{i,j}$, P and T^* are multiples of g, we conclude that $\sum_{j \in J} \bar{x}_{i,j} \cdot p_{i,j} \leq T^* + P - g$. The following calculation then shows the claimed approximation guarantee:

$$\begin{aligned}
T^* + P - g &\leq T^* \left(2 - \frac{(\beta+1)g}{T^*}\right) \\
&\leq T^* \left(2 - \frac{(\beta+1)g}{P + \beta \cdot g}\right) \\
&\leq T^* (2 - \alpha) .
\end{aligned}$$

□

In particular, the above theorem applies to families of instances which use only a finite set of processing times. Such families often arise in NP-hardness reductions. Hence, if one wants to prove that $R||C_{\max}$ cannot be approximated with a better factor than 2 then one has to construct reductions which use an infinite number of processing times. We formalize this observation in the following corollary.

Corollary 7.13. *Let $\mathcal{I}$ be a family of instances of $R||C_{\max}$. Let P be a finite set of integers. Assume that for each instance $I \in \mathcal{I}$ and each in I arising processing time $p_{i,j}$ it holds that $p_{i,j} \in P \cup \{\infty\}$. Then for the family of instances $\mathcal{I}$ there is an approximation algorithm with performance guarantee $2 - \alpha$ with $\alpha = \gcd\{p|p \in P\}/\max\{p|p \in P\}$.*

7.4.2 Bounded Range of Processing Times

Now we show that if the finite execution times of the jobs differ by at most a factor of 10/3 then the configuration-LP has an integrality gap of at most $1 + \frac{5}{6}$. Hence, using reductions of this type (which applies to the strongest known NP-hardness reductions for $R||C_{\max}$) one cannot rule out a $2 - \varepsilon$ approximation algorithm.

Theorem 7.14. *Consider instances of $R||C_{\max}$ with a value γ such that $p_{i,j} \in [\gamma, 10\gamma/3] \cup \{\infty\}$ for all machines i and all jobs j. For these instances there is a $1 + \frac{5}{6} \approx 1.83$-approximation algorithm.*

Proof. We use the LST-LP and – depending on the makespan T given by the binary search – we strengthen it with additional linear inequalities. Assume we are given a target makespan T. If $T \geq 4\gamma$ then we solve the (original) LST-LP. If it is feasible, due to Theorem 7.1 we know that we can round it to an integral solution whose makespan is bounded by $4\gamma + \frac{10}{3}\gamma = \frac{22}{3}\gamma \leq \left(1 + \frac{5}{6}\right) T$.

So now assume that $T < 4\gamma$. Here, we strengthen the LST-LP with the linear inequalities

$$\begin{aligned}
\sum_{j \in J : p_{i,j} > \frac{T}{2}} x_{i,j} &\leq 1 && \forall i \in M \\
\sum_{j \in J : p_{i,j} > \frac{T}{3}} x_{i,j} &\leq 2 && \forall i \in M \\
\sum_{j \in J} x_{i,j} &\leq 3 && \forall i \in M.
\end{aligned}$$

Note that any integral solution with a makespan $T' \leq T$ satisfies the additional cuts. Hence, the resulting LP is feasible if the optimal makespan is at most T. We perform LST-rounding. With the notation of the proof of Theorem 7.1 for each machine i there can be at most three machines nodes v_ℓ^i. The node v_2^i has only jobs j with $p_{i,j} \leq T/2$ assigned to it, the node v_3^i only jobs j with $p_{i,j} \leq T/3$. This yields a bound of $T + \frac{T}{2} + \frac{T}{3} = \left(1 + \frac{5}{6}\right) T$ for the makespan of i. Doing this reasoning for all machines yields the desired bound on the makespan.

Note that in contrast to the previous algorithms here we have two linear programs: the usual LST-LP if $T \geq 4\gamma$ for the guessed makespan T and the strengthened LP if $T < 4\gamma$. Still, there is a value T^* such that for all $T < T^*$ the respective LP is infeasible and for all $T \geq T^*$ the respective LP is feasible. We solve the respective LP for T^* and perform the above rounding procedure. This yields a solution with makespan $\left(1 + \frac{5}{6}\right) T^* \leq \left(1 + \frac{5}{6}\right) OPT$. □

Unfortunately, we do not gain anything by generalizing this method further to, e. g., the case that $p_{i,j} \in [\gamma, 4\gamma] \cup \{\infty\}$. The reason is that $T + \frac{T}{2} + \frac{T}{3} = T(1 + \frac{5}{6}) < 2T$ but $T + \frac{T}{2} + \frac{T}{3} + \frac{T}{4} \approx 2.08T > 2T$ and a 2-approximation algorithm is already known.

7.4.3 Big Machines/Small Machines

Finally, we show that for an improved NP-hardness reduction it is crucial that it is not clear what machines execute big jobs. Here, we call a job *big on machine i* if $p_{i,j} \geq \frac{2}{3}OPT$. Formally, assume we are given an instance of $R||C_{\max}$ and assume we know exactly what machines execute a big job. We call such machines *big machines* and all other machines *small machines*. For this setting we give a 5/3-approximation algorithm. Then we generalize this to the setting that for $m - O(\log n)$ machines we know whether they execute a big job. Another application is the setting where there are at most a constant number of jobs which are big on some machine. There, we also obtain a 5/3-approximation algorithm.

We call a job j *big on machine* i if $p_{i,j} \geq \frac{2}{3}OPT$. First, we present an algorithm that assumes that for each machine it is known in advance whether it executes a big job. Let I be an instance of $R||C_{\max}$. Assume that the set of machines is partitioned into two sets M_{big} and M_{small} such that we know that in some optimal solution each machine $i \in M_{\text{big}}$ executes a big job and each machine $i' \in M_{\text{small}}$ does not execute a big job. Like above, we use a binary search framework to "guess" the optimal makespan T.

For each machine $i \in M$ denote by $J^i_{\text{big}} \subseteq J$ all jobs j with $2T/3 \leq p_{i,j} \leq T$, by $J^i_{\text{med}} \subseteq J$ all jobs j with $T/3 \leq p_{i,j} < 2T/3$ and by $J^i_{\text{small}} \subseteq J$ all jobs j with $p_{i,j} < T/3$. We solve the following linear program, denoted by BS-LP:

$$
\begin{aligned}
\text{(BS-LP)} \qquad \sum_{j \in J^i_{\text{big}} \cup J^i_{\text{med}}} x_{i,j} &\leq 1 && \forall i \in M_{\text{big}} \\
\sum_{j \in J^i_{\text{small}}} x_{i,j} \cdot p_{i,j} &\leq T/3 && \forall i \in M_{\text{big}} \\
\sum_{j \in J^i_{\text{med}} \cup J^i_{\text{small}}} x_{i,j} \cdot p_{i,j} &\leq T && \forall i \in M_{\text{small}} \\
\sum_{j \in J^i_{\text{big}}} x_{i,j} &= 0 && \forall i \in M_{\text{small}} \\
\sum_{i \in M} x_{i,j} &= 1 && \forall j \in J \\
x_{i,j} &\geq 0 && \forall i \in M, j \in J.
\end{aligned}
$$

Note that despite the separation of the jobs into the three classes we have that for all $T \geq OPT$ the integral optimum satisfies BS-LP and hence BS-LP is feasible. Now assume that BS-LP is feasible for the guessed makespan T and let x be a solution. We perform LST-rounding.

Lemma 7.15. *Let I be an instance of $R||C_{\max}$, let T be an integer and assume we are given a partition of the machines into big and small machines. If BS-LP is feasible then the makespan after LST-rounding is bounded by $5T/3$.*

Proof. Let $i \in M_{\text{small}}$. Only jobs j with $p_{i,j} \leq 2T/3$ were (fractionally) assigned to i by BS-LP. Hence, during the LST-rounding the makespan of i can increase to at most $T + 2T/3 = 5T/3$. Now let $i' \in M_{\text{big}}$. The total processing time of small jobs on i' (bounded by $T/3$ in the LP-solution) can increase by at most $T/3$ (since $p_{i',j} \leq T/3$ for all $j \in J^{i'}_{\text{small}}$). There is at most one big job assigned to i'. Hence, the makespan of i' is bounded by $T/3 + T/3 + T = 5T/3$. Hence, the overall makespan of the solution is bounded by $5T/3$. □

Now assume that we know for $m - O(\log n)$ machines whether in an optimal solution they execute a big job. For the remaining $O(\log n)$ machines we can enumerate this information in polynomial time. For the remaining problem we apply the above algorithm. This yields the following theorem.

Theorem 7.16. *Let I be an instance of $R||C_{\max}$. Assume we know for at least $m - O(\log n)$ machines whether they execute a big job in an optimal solution. Then there is a polynomial time algorithm that computes a solution for I whose makespan is bounded by $\frac{5}{3}OPT$.*

For instances with at most c jobs which are big on some machine (for a fixed constant c) we can enumerate in polynomial time the at most $O(m^c)$ sets of machines which execute big jobs. For each of the sets we run the above algorithm together with a binary search. This yields the following theorem.

Theorem 7.17. *Let c be a fixed integer. There is a $5/3$-approximation algorithm for instances of $R||C_{\max}$ with at most c jobs which are big on some machine in an optimal solution.*

We would like to point out that Theorem 7.16 is particularly important if one wants prove that there can be no approximation algorithm for $R||C_{\max}$ with a better performance ratio than α for some $\alpha \in (\frac{5}{3}, 2]$. It implies that a reduction showing this must use instances for which no polynomial time algorithm can determine for almost all machines whether they execute a big job in an optimal solution.

7.5 MaxMin-Allocation Problem

In this section we study the MaxMin-allocation problem on unrelated machines. Recall that in contrast to $R||C_{\max}$ now the objective is to maximize the minimum load of a machine. First, we investigate the *MaxMin-balancing* problem, where every job can be assigned to at most two machines (with possibly different processing times on each machine). For this case it is known that the configuration-LP has an integrality gap of 2 [19]. However, when allowing only polynomial running time it can only be solved approximately which yields a

$(2+\varepsilon)$-approximation algorithm for MaxMin-balancing. Also, it requires to solve a linear program with a PTAS for Knapsack as a separation oracle. In particular, for small ε this algorithm needs much running time and it is highly non-trivial to implement. Instead, we present here a purely combinatorial 2-approximation algorithm with quadratic running time which is quite easy to implement.

After that we present approximation algorithms which compute 2- and 4-approximate half-integral solutions for the general MaxMin-allocation problem. Recall that for this setting the best known approximation algorithm (which computes integral solutions) has a performance guarantee of $O\left(\sqrt{m}\log^3 m\right)$. Finally, we discuss the setting where it is known a priori what machines execute big jobs (similar to Section 7.4.3). For this setting we give a 2-approximation algorithm.

7.5.1 2-Approximation for MaxMin-Balancing

We present our purely combinatorial 2-approximation algorithm for MaxMin-balancing. Let I be an instance of the problem and let T be a positive integer. Our algorithm either finds a solution with value $T/2$ or asserts that there is no solution with value T or larger. With an additional binary search this yields a 2-approximation algorithm. For each machine i denote by $J_i = \{j_{i,1}, j_{i,2}, ...\}$ the list of all jobs which can be assigned to i. We partition this set into the sets $A_i \dot\cup B_i$ where $A_i = \{a_{i,1}, a_{i,2}, ...\}$ denotes the jobs in J_i which can be assigned to two machines (machine i and some other machine) and B_i denotes the jobs in J_i which can only be assigned to i. We define A'_i to be the set A_i without the job with largest processing time (or one of those jobs in case there is a tie). For any set of jobs J'_i and a machine i we define $p(J'_i) := \sum_{j\in J'} p_{i,j}$.

Denote by $p_{i,\ell}$ the processing time of job $a_{i,\ell}$ on machine i. We assume that the elements of A_i are ordered non-increasingly by processing time, i. e., $p_{i,\ell} \geq p_{i,\ell+1}$ for all respective values of ℓ. If there is a machine i such that $p(A_i) + p(B_i) < T$ we output that there is no solution with value T or larger. So now assume that $p(A_i)+p(B_i) \geq T$ for all machines i. If there is a machine i such that $p(A'_i) + p(B_i) < T$ (but $p(A_i) + p(B_i) \geq T$) then any solution with value at least T has to assign $a_{i,1}$ to i. Hence, we assign $a_{i,1}$ to i. This can be understood as moving $a_{i,1}$ from A_i to B_i. We rename the remaining jobs in A_i accordingly and update the values $p(A_i)$, $p(A'_i)$, and $p(B_i)$. We do this procedure until either

- there is one machine i such that $p(A_i)+p(B_i) < T$, in this case we output that there is no solution with value T or larger, or
- for all machines i we have that $p(A'_i) + p(B_i) \geq T$.

We call this phase the *preassignment phase*.

Lemma 7.18. *If during the preassignment phase the algorithm outputs that that no solution with value T or larger exists, then there can be no such solution.*

Proof. If the algorithm moves a job $a_{i,\ell}$ from A_i to B_i then any solution with value T or larger has to assign $a_{i,\ell}$ to B_i. Hence, if at some point there is a machine i such that $p(A_i) + p(B_i) < T$ then there can be no solution with value at least T. □

Now we construct a graph G as follows: For each machine i and each job $a_{i,\ell} \in A_i$ we introduce a vertex $\langle a_{i,\ell} \rangle$. We connect two vertices $\langle a_{i,\ell} \rangle, \langle a_{i',\ell'} \rangle$ if $a_{i,\ell}$ and $a_{i',\ell'}$ represent the same job (but on different machines). Also, for each machine i we introduce an edge between the vertices $\langle a_{i,2k+1} \rangle$ and $\langle a_{i,2k+2} \rangle$ for each respective value $k \geq 0$. The reason for the latter edges is that later exactly one of the two jobs $j_{i,2k+1}$, $j_{i,2k+2}$ will be assigned to i.

Lemma 7.19. *The graph G is bipartite.*

Proof. Since every vertex in G has degree two or less the graph splits into cycles and paths. It remains to show that all cycles have even length. There are two types of edges: edges which connect two vertices $\langle a_{i,\ell} \rangle, \langle a_{i',\ell'} \rangle$ such that $i = i'$ and edges connecting two vertices which correspond to the same job on two different machines. On a cycle, the edges of these two types alternate and hence the graph is bipartite. □

Due to Lemma 7.19 we can color G with two colors, black and white. Let i be a machine. We assign each job $a_{i,\ell}$ to i if and only if $\langle a_{i,\ell} \rangle$ is black. Also, we assign each job in B_i to i.

Lemma 7.20. *The algorithm outputs a solution whose value is at least $T/2$.*

Proof. Let i be a machine. We show that the total weight of the jobs assigned to i is at least $p(A'_i)/2 + p(B_i)$. For each connected pair of vertices $\langle a_{i,2k+1} \rangle, \langle a_{i,2k+2} \rangle$ we have that either $a_{i,2k+1}$ or $a_{i,2k+2}$ is assigned to i. We calculate that $\sum_{k \in \mathbb{N}} p_{i,2k+2} \geq p(A'_i)/2$. Since $p_{i,2k+1} \geq p_{i,2k+2}$ (for all respective values k) we conclude that the total weight of the jobs assigned to i is at least $p(A'_i)/2 + p(B_i)$. Since $p(A'_i) + p(B_i) \geq T$ the claim follows. □

In order to turn the above algorithm into an algorithm for the entire problem an additional binary search is necessary to find the correct value of T. Now we discuss how to implement the overall algorithm efficiently.

First, we test whether $n < m$. If this is the case then any (optimal) solution has value 0. So now assume that $n \geq m$. In order to initialize the ordered sets A_i and B_i we need to sort the jobs by execution time (in the list that we sort we have two entries for every job, each corresponding to one of its possible execution times). We sort this list in $O(n \log n)$ steps. Note that the sorting needs to be done only once, no matter how many values T we try. Starting with an ordered list of the jobs, we can build the ordered lists A_i and the sets B_i in linear time. The preassignment phase can be implemented in linear time: For each machine i we need to check whether $p(A'_i) + p(B_i) < T$. We call this a *first-check*. If we move a job $a_{i,\ell}$ from A_i to B_i then the other machine on which one could possibly assign $a_{i,\ell}$ needs to be checked again. We call this a

second-check. There are m first-checks and at most n second-checks necessary. Hence, this procedure can be implemented in linear time. Coloring the graph G with two colors also requires only linear time.

For the binary search we need to try at most $\log D$ values, where D is defined by $D := \sum_{i,j} p_{i,j}$. We have that $\log D \leq |I|$ where $|I|$ denotes the length of the overall input in binary encoding. The sorting needs to be done only once and this takes in $O(|I| \log |I|)$ time. For every value T that we try $O(|I|)$ steps are necessary. This yields an overall running time of $O\big(|I|^2\big)$.

Theorem 7.21. *There is a 2-approximation algorithm for the Max-Min-balancing problem with running time $O\big(|I|^2\big)$.*

7.5.2 Half-Integral Solutions

For the general MaxMin-allocation problem the best known polynomial time approximation algorithm achieves an approximation factor of $O(\sqrt{m}\log^3 m)$ [10]. A constant factor approximation algorithm seems difficult to achieve, in particular since the configuration-LP has an integrality gap of $\Omega(\sqrt{m})$ [12]. However, we present a polynomial time algorithm that computes a half-integral solution whose value is at most by a factor 2 smaller than the best integral solution. Moreover, we show that at the cost of at most a factor of 2 this solution can be transformed to a half-integral solution in which at most $m/2$ jobs are fractionally assigned.

Let I be an instance of the MaxMin-allocation problem. Let T be a guessed optimal value. As a first step we redefine I to an instance I' by changing the execution times of the jobs to $p'_{i,j} := \min\{p_{i,j}, T\}$. Clearly, if there is an integral solution with value T for I then there is also an integral solution with value T for I'. With the instance I' we solve the following linear program MMA-LP:

$$\text{(MMA-LP)} \qquad \sum_{j \in J} x_{i,j} \cdot p'_{i,j} \geq T \qquad \forall i \in M$$

$$\sum_{i \in M} x_{i,j} = 1 \qquad \forall j \in J$$

$$x_{i,j} \geq 0 \qquad \forall j \in J, i \in M.$$

If MMA-LP is infeasible there can be no integral solution with value T or larger. Now assume that MMA-LP is feasible. We perform a slightly modified LST-rounding: instead of assigning one unit of jobs to each vertex for a machine i, we assign $1/2$ units of jobs to every vertex (and hence we need more vertices). After the rounding we obtain a half-integral solution $HALF(I)$.

Theorem 7.22. *Let I be an instance of the MaxMin-allocation problem. There is a polynomial time algorithm that computes a half-integral solution $HALF(I)$ such that $HALF(I) \geq \frac{1}{2}OPT(I)$.*

Proof. Let i be a machine. Similarly to the analysis of LST-rounding for $R||C_{\max}$, during the rounding we lose at most the load of the jobs (fractionally) assigned to the first vertex of i. Since at most $1/2$ units of jobs can be assigned to this vertex and all $p'_{i,j} \leq T$, the machine i keeps a makespan of at least $T/2$. Since MMA-LP was feasible for T we conclude that T is an upper bound for $OPT(I)$. □

Now we show how to modify $HALF(I)$ to get another half-integral solution in which at most $m/2$ jobs are fractionally assigned. This modification comes at a cost of at most a factor 2 and hence yields a 4-approximation.

Let I be an instance of the MaxMin-allocation problem and let $HALF(I)$ be any half-integral solution for I. We do not change the assignment of jobs which were assigned integrally in $HALF(I)$. For each machine i let $J_i = \{j_{i,1}, j_{i,2}, ...\}$ denote the jobs which are fractionally assigned to i. Let $p_{i,\ell}$ denote the processing time of $j_{i,\ell}$. We assume that the jobs are ordered non-increasingly by processing time, i. e., $p_{i,\ell} \geq p_{i,\ell+1}$ for all respective values of ℓ. We define a graph G as follows: For each job $j_{i,\ell}$ we introduce a vertex $\langle j_{i,\ell} \rangle$. We connect two vertices $\langle j_{i,\ell} \rangle$, $\langle j_{i',\ell'} \rangle$ by an edge if $j_{i,\ell}$ and $j_{i',\ell'}$ represent the same job (but on different machines). We call such edges the *outer edges*. Also, for each machine i we introduce an edge between the vertices $\langle j_{i,2k} \rangle$ and $\langle j_{i,2k+1} \rangle$ for each respective value $k \geq 1$ (*inner edges*).

Lemma 7.23. *The graph G is bipartite.*

Proof. Can be proven similarly as Lemma 7.19. □

Now we color G with two colors, black and white. From the coloring we compute a new solution as follows: we take each the black vertex $\langle j_{i,\ell} \rangle$ and assign the job $j_{i,\ell}$ completely to machine i. Note that this is well-defined since if $\langle j_{i,\ell} \rangle$ is black then the other vertex that represents the job is white.

Now there are two cases that we treat separately. For each machine i we call the vertex $\langle j_{i,1} \rangle$ the *head vertex*. Let P be a path in G. If one of the end vertices of P is a head vertex and the other one is not a head vertex, then we (re-)color P such that the head vertex is black and do the job assignment as above. If both end vertices of P are head vertices then we take the head vertex $\langle j_{i,1} \rangle$ that was colored in white and assign one half of the job $j_{i,1}$ to machine i and the other half to the respective other machine. Denote by $HALF'(I)$ the resulting solution.

Lemma 7.24. *The solution $HALF'(I)$ is half-integral and at most $m/2$ jobs are fractionally assigned. Moreover, $HALF'(I) \geq \frac{1}{2} \cdot HALF(I)$.*

Proof. From the definition of the algorithm we conclude that for each machine i the job $j_{i,1}$ corresponding to the head vertex $\langle j_{i,1} \rangle$ of i is at least half assigned to i. For each pair of jobs $j_{i,2k}, j_{i,2k+1}$ (for $k \geq 1$) one of them is at least half assigned to i. Denote by $B(i)$ the jobs which were integrally assigned to i in $HALF(I)$. In the solution $HALF'(I)$ the makespan of the machine i is at

least $\sum_{j\in B(i)} p_{i,j} + \frac{1}{2}\sum_{k\geq 0} p_{i,2k+1}$. Since $p_{i,2k} \geq p_{i,2k+1}$ for all machines i an all values k we conclude that

$$\begin{aligned}\sum_{j\in B(i)} p_{i,j} + \frac{1}{2}\sum_{k\geq 0} p_{i,2k+1} &\geq \sum_{j\in B(i)} p_{i,j} + \frac{1}{4}\sum_{k\geq 1} p_{i,k} \\ &\geq \frac{1}{2}\left(\sum_{j\in B(i)} p_{i,j} + \frac{1}{2}\sum_{k\geq 1} p_{i,k}\right) \\ &= \frac{1}{2}HALF_i(I)\end{aligned}$$

where $HALF_i(I)$ denotes the makespan of i in $HALF(I)$.

The only case in which a job is fractionally assigned is when both end vertices of a path P are head vertices. Since every machine has only one head vertex, there can be at most $m/2$ such paths. Hence, the number of fractionally assigned jobs is bounded by $m/2$. □

We would like to note that during the transformation from $HALF(I)$ to the solution $HALF'(I)$ a machine could indeed lose half of its load. For instance, consider a machine with two jobs with identical processing times which are half assigned to it. During the transformation, the machine could lose one of the two halves and keep only the other one. We summarize the algorithm in the following theorem.

Theorem 7.25. *Let I be an instance of the MaxMin-allocation problem. There is a polynomial time algorithm that computes a half-integral solution $HALF'(I)$ such that $HALF(I) \geq \frac{1}{4}OPT(I)$. Moreover, $HALF'(I)$ assigns at most $m/2$ jobs fractionally.*

7.5.3 Tractable Cases

Similarly as for $R||C_{\max}$ if we knew what machines execute a big job in an optimal solution or if the number of big jobs is bounded by a constant we can guarantee better approximation factors. Here, we define a job j to be *big on machine i* if $p_{i,j} \geq \frac{1}{2}OPT$. We call a machine *big* if we know that it executes a big job in an optimal solution, and *small* otherwise. The sets M_{big} and M_{small} denote the respective sets of machines. For a guessed objective value T we set up the following linear program:

$$\begin{aligned}\sum_{j\in J^i_{\text{big}}} x_{i,j} &= 1 && \forall i \in M_{\text{big}} \\ \sum_{j\in J^i_{\text{small}}} x_{i,j}\cdot p_{i,j} &\geq T && \forall i \in M_{\text{small}} \\ \sum_{i\in M} x_{i,j} &= 1 && \forall j \in J \\ x_{i,j} &\geq 0 && \forall i \in M, j \in J.\end{aligned}$$

Here, for each machine i we denote by J^i_{small} all jobs j such that $p_{i,j} < T/2$ and by J^i_{big} all jobs j with $p_{i,j} \geq T/2$. We perform LST-rounding with the obtained fractional solution. After the rounding procedure, each big machine will still have exactly one big job assigned to it. The load of each small machine will decrease by at most $T/2$. This yields the following theorems.

Theorem 7.26. *Let I be an instance of the MaxMin-allocation problem. Assume we know for at least $m - O(\log n)$ machines whether they execute a big job in an optimal solution. Then there is a polynomial time algorithm that computes a solution for I whose value is at least $\frac{1}{2}OPT(I)$.*

Proof. For $m - O(\log n)$ machines we know whether they are small or big. For the remaining $O(\log n)$ machines we can enumerate this information in polynomial time. For each enumeration we apply the above framework. □

Theorem 7.27. *Let c be a fixed integer. There is a 2-approximation algorithm for instances of the MaxMin-allocation problem where it is known that at most c jobs are big in an optimal solution.*

Proof. We enumerate the c machines which are big. For remainder we apply the above framework. □

7.6 Conclusion

As mentioned above, the problem of minimizing the makespan for scheduling of unrelated machines is one of the most prominent open problems in scheduling. Closing the gap between the 2-approximation algorithm by Lenstra et al. [69] and their 3/2-hardness result seems a very challenging task. Since the machines are unrelated, usual rounding and enumeration approaches like for identical machines cannot be used (if the number of machines is part of the input). However, our results show that most LP-based approaches are deemed to fail, even for the unrelated graph balancing case. Hence, when trying to find a better approximation algorithm it seems reasonable to study the latter setting. To the best of our knowledge, it has not been considered in its own right so far.

In the paper by Ebenlendr et al. [26] the setting of graph balancing *and* restricted assignment is studied. Our results and the recent result by Svensson [104] indicate that the restricted assignment feature is actually the reason why this improvement was possible, rather than the restriction to graph balancing. In the latter paper, Svensson proves an upper bound for the integrality gap of the configuration-LP of 33/17 in the general setting and $5/3 + \varepsilon$ if $p_{i,j} \in \{\varepsilon, 1, \infty\}$ for all machines i and all jobs j. To the best of our knowledge, for the restricted assignment case no instance is known for which the configuration-LP has an integrality gap larger than 3/2. It would be interesting to construct such an instance. In fact, in our constructions we used only the processing times $\{\varepsilon, 1, \infty\}$. It is not clear to us how more distinct processing times in an instance could help to show a larger integrality gap.

Another way of looking at the restricted assignment case is that for each job j there are two values $p_j \in \mathbb{N}^+$ and $p'_j = \infty$ such that for each machine i it holds that $p_{i,j} \in \{p_j, p'_j\}$. It would be interesting to see what one can prove if $p'_j \in \mathbb{N}^+ \cup \{\infty\}$ and hence each job has two different processing times on the machines. Knowing our results it seems likely to encounter a jump in the complexity if each job j is allowed three different processing times p_j, p'_j, p''_j. Note that for this setting we proved that the configuration-LP has an integrality gap of 2.

For the MaxMin-allocation problem, our algorithm presented in Section 7.5 achieves the best known approximation factor in its setting (and it is in fact best possible, unless $P = NP$). To the best of our knowledge, it is the only such algorithm for a complex case of the MaxMin-allocation problem which does not rely on solving a linear program, in particular not the computationally expensive configuration-LP. It would be interesting whether purely combinatorial algorithms are also possible for other settings of the problem. Also, it is worthwhile to note that for MaxMin-allocation, the unrelated graph balancing setting seems completely solved (with a combinatorial 2-approximation algorithm and a matching NP-hardness lower bound) whereas for $R||C_{\max}$ this setting is not better understood than the general case.

Bibliography

[1] M. Adler, S. Khanna, R. Rajaraman, and A. Rosén. Time-constrained scheduling of weighted packets on trees and meshes. *Algorithmica*, 36:123–152, 2003.

[2] M. Adler, R. Sitaraman, A. Rosenberg, and W. Unger. Scheduling time-constrained communication in linear networks. *Theory of Computing Systems*, 35:599–623, 2008.

[3] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1999)*, pages 32–44. IEEE, 1999.

[4] N. Alon and J. Spencer. *The Probabilistic Method*. John Wiley & Sons, 1992.

[5] M. Andrews, A. Fernández, M. Harchol-Balter, F. Leighton, and L. Zhang. General dynamic routing with per-packet delay guarantees of O(distance + 1/session rate). *SIAM Journal on Computing*, 30:1594–1623, 2000.

[6] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45:501–555, 1998.

[7] S. Arora and S. Safra. Probabilistic checking of proofs: a new characterization of NP. *Journal of the ACM*, 45:70–122, 1998.

[8] A. Asadpour, U. Feige, and A. Saberi. Santa Claus meets hypergraph matchings. In *Proceedings of the 11th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2008)*, volume 5171 of *Lecture Notes in Computer Science*, pages 10–20. Springer, 2008.

[9] A. Asadpour, U. Feige, and A. Saberi. Santa Claus meets hypergraph matchings. Technical report, Standford University, 2009. Available for download at `http://www.stanford.edu/~asadpour/publication.htm`.

[10] A. Asadpour and A. Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. *SIAM Journal on Computing*, 39:2970–2989, 2010.

[11] F. Meyer auf der Heide and B. Vöcking. Shortest paths routing in arbitrary networks. *Journal of Algorithms*, 31:105–131, 1999.

[12] N. Bansal and M. Sviridenko. The Santa Claus problem. In *Proceedings of the 38th ACM Symposium on Theory of computing (STOC 2006)*, pages 31–40. ACM, 2006.

[13] S. K. Baruah, R. R. Howell, and L. E. Rosier. Feasibility problems for recurring tasks on one processor. *Theoretical Computer Science*, 118:3–20, 1993.

[14] M. Bateni, M. Charikar, and V. Guruswami. Maxmin allocation via degree lower-bounded arborescences. In *Proceedings of the 41st ACM Symposium on Theory of Computing (STOC 2009)*, pages 543–552. ACM, 2009.

[15] N. Baumann and M. Skutella. Earliest arrival flows with multiple sources. *Mathematics of Operations Research*, 34:499–512, 2009.

[16] R. E. Burkard, K. Dlaska, and B. Klinz. The quickest flow problem. *ZOR — Methods and Models of Operations Research*, 37:31–58, 1993.

[17] C. Busch, M. Magdon-Ismail, M. Mavronicolas, and P. Spirakis. Direct routing: Algorithms and complexity. *Algorithmica*, 45:45–68, 2006.

[18] G. C. Buttazzo. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series)*. Springer-Verlag TELOS, 2004.

[19] D. Chakrabarty, J. Chuzhoy, and S. Khanna. On allocating goods to maximize fairness. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2009)*, pages 107–116. IEEE, 2009.

[20] J. Chuzhoy and P. Codenotti. Resource minimization job scheduling. In *Proceedings of the 12th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2009)*, volume 5687 of *Lecture Notes in Computer Science*, pages 70–83. Springer, 2009.

[21] M. Cieliebak, T. Erlebach, F. Hennecke, B. Weber, and P. Widmayer. Scheduling with release times and deadlines on a minimum number of machines. In *Exploring New Frontiers of Theoretical Informatics*, volume 155 of *International Federation for Information Processing (IFIP)*, pages 209–222. Springer, 2004.

[22] R. Cole, K. Ost, and S. Schirra. Edge-coloring bipartite multigraphs in $O(E \log D)$ time. *Combinatorica*, 21:5–12, 2001.

[23] J. R. Correa, M. Skutella, and J. Verschae. The power of preemption on unrelated machines and applications to scheduling orders. In *Proceedings of the 12th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2009)*, volume 5687 of *Lecture Notes in Computer Science*, pages 84–97. Springer, 2009.

[24] M. Dertouzos. Control robotics: The procedural control of physical processes. In *Proceedings of the IFIP Congress (IFIP'74)*, pages 807–813, 1974.

[25] M. di Ianni. Efficient delay routing. *Theoretical Computer Science*, 196:131–151, 1998.

[26] T. Ebenlendr, M. Krčál, and J. Sgall. Graph balancing: a special case of scheduling unrelated parallel machines. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete algorithms (SODA 2008)*, pages 483–490. SIAM, 2008.

[27] F. Eisenbrand, N. Hähnle, M. Niemeier, M. Skutella, J. Verschae, and A. Wiese. Scheduling periodic tasks in a hard real-time environment. In *Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP 2010)*, volume 6198 of *Lecture Notes in Computer Science*, pages 299–311. Springer, 2010.

[28] F. Eisenbrand, K. Kesavan, R. Mattikalli, M. Niemeier, A. Nordsieck, M. Skutella, J. Verschae, and A. Wiese. Solving an avionics real-time scheduling problem by advanced IP-methods. In *Proceedings of the 18th Annual European Symposium on Algorithms (ESA 2010)*, volume 6346 of *Lecture Notes in Computer Science*, pages 11–22. Springer, 2010.

[29] F. Eisenbrand and T. Rothvoß. EDF-schedulability of synchronous periodic task systems is coNP-hard. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010)*, pages 1029–1034. SIAM, 2010.

[30] D. Engels, D. Karger, S. Kolliopoulos, S. Sengupta, R. Uma, and J. Wein. Techniques for scheduling with rejection. *Journal of Algorithms*, 49:175–191, 2003.

[31] L. Epstein, A. Levin, A. Marchetti-Spaccamela, N. Megow, J. Mestre, M. Skutella, and L. Stougie. Universal sequencing on a single machine. In *Proceedings of the 14th Conference on Integer Programming and Combinatorial Optimization (IPCO 2010)*, volume 6080 of *Lecture Notes in Computer Science*, pages 230–243. Springer, 2010.

[32] T. Erlebach and K. Jansen. An optimal greedy algorithm for wavelength allocation in directed tree networks. In *Proceedings of the DIMACS Workshop on Network Design: Connectivity and Facilities Location*, volume 40, pages 117–129. AMS, 1997.

[33] T. Erlebach and K. Jansen. The complexity of path coloring and call scheduling. *Theoretical Computer Science*, 255:33–50, 2001.

[34] U. Feige. On allocations that maximize fairness. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete algorithms (SODA 2008)*, pages 287–293. SIAM, 2008.

[35] W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within $1 + \varepsilon$ in linear time. *Combinatorica*, 1:349–355, 1981.

[36] L. Fleischer. Faster algorithms for the quickest transshipment problem. *SIAM Journal on Optimization*, 12:18–35, 2001.

[37] L. Fleischer and M. Skutella. Quickest flows over time. *SIAM Journal on Computing*, 36:1600–1630, 2007.

[38] L. K. Fleischer and É. Tardos. Efficient continuous-time dynamic network flow algorithms. *Operations Research Letters*, 23:71–80, 1998.

[39] L. R. Ford and D. R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6:419–433, 1958.

[40] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[41] M. Gairing, B. Monien, and A. Woclaw. A faster combinatorial approximation algorithm for scheduling unrelated parallel machines. *Theoretical Computer Science*, 380:87–99, 2007.

[42] D. Gale. Transient flows in networks. *Michigan Mathematical Journal*, 6:59–63, 1959.

[43] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the theory of NP-completeness*. Freeman, 1979.

[44] L. Gargano, P. Hell, and S. Perennes. Coloring all directed paths in a symmetric tree, with an application to optical networks. *Journal of Graph Theory*, 38:183–196, 2001.

[45] E. Gawrilow, E. Köhler, R. H. Möhring, and B. Stenzel. Dynamic routing of automated guided vehicles in real-time. In *Mathematics — Key Technology for the Future*, pages 165–178. Springer, 2008.

[46] M. X. Goemans, J. M. Wein, and D. P. Williamson. A 1.47-approximation algorithm for a preemptive single-machine scheduling problem. *Operations Research Letters*, 26:149 – 154, 2000.

[47] D. Golovin. Max-min fair allocation of indivisible goods. Technical Report CMU-CS-05-144, School of Computer Science, Carnegie Mellon University, June 2005.

[48] B. Haeupler, B. Saha, and A. Srinivasan. New Constructive Aspects of the Lovász Local Lemma. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010)*, pages 397–406. IEEE, 2010.

[49] T. Hagerup and C. Rüb. A guided tour of chernoff bounds. *Information Processing Letters*, 33:305–308, 1990.

[50] A. Hall, S. Hippler, and M. Skutella. Multicommodity flows over time: Efficient algorithms and complexity. *Theoretical Computer Science*, 379:387–404, 2007.

[51] D. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems.* Thomson, 1996.

[52] D. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM*, 34:144–162, January 1987.

[53] D. Hochbaum and D. B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM Journal on Computing*, 17:539–551, 1988.

[54] H. Hoogeveen, M. Skutella, and G. J. Woeginger. Preemptive scheduling with rejection. *Mathematical Programming*, 94:361–374, 2003.

[55] B. Hoppe and É. Tardos. Polynomial time algorithms for some evacuation problems. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1994)*, pages 433–441. SIAM, 1994.

[56] B. Hoppe and É. Tardos. The quickest transshipment problem. *Mathematics of Operations Research*, 25:36–62, 2000.

[57] E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM*, 23:317–327, April 1976.

[58] K. Jansen. Approximation results for wavelength routing in directed trees. In *Proceedings of the 2nd Workshop on Optics and Computer Science (WOCS 1997)*, 1997.

[59] S. Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the 34th ACM Symposium on Theory of computing (STOC 2002)*, pages 767–775. ACM, 2002.

[60] R. Koch, B. Peis, M. Skutella, and A. Wiese. Real-time message routing and scheduling. In *Proceedings of the 12th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2009)*, volume 5687 of *Lecture Notes in Computer Science*, pages 217–230. Springer, 2009.

[61] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 2007.

[62] J. Labetoulle, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Preemptive scheduling of uniform machines subject to release dates. In *Progress in Combinatorial Optimization*, pages 245–261. Academic Press, 1984.

[63] E. L. Lawler and J. Labetoulle. On preemptive scheduling of unrelated parallel processors by linear programming. *Journal of the ACM*, 25:612–619, 1978.

[64] K. Lee, J. Y.-T. Leung, and M. L. Pinedo. A note on graph balancing problems with restrictions. *Information Processing Letters*, 110:24–29, 2009.

[65] F. T. Leighton, B. M. Maggs, and S. B. Rao. Packet routing and job-scheduling in $O(congestion + dilation)$ steps. *Combinatorica*, 14:167–186, 1994.

[66] F. T. Leighton, B. M. Maggs, and A. W. Richa. Fast algorithms for finding $O(congestion+dilation)$ packet routing schedules. *Combinatorica*, 19:375–401, 1999.

[67] F. T. Leighton, F. Makedon, and I. G. Tollis. A $2n - 2$ step algorithm for routing in an $n \times n$ array with constant-size queues. *Algorithmica*, 14:291–304, 1995.

[68] H. W. jun. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.

[69] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.

[70] J. Y.-T. Leung. *Handbook of Scheduling: Algorithms, Models and Performance Analysis*. Chapman & Hall/CRC, 2004.

[71] J. Y.-T. Leung and C. Li. Scheduling with processing set restrictions: A survey. *International Journal of Production Economics*, 116:251–262, 2008.

[72] J.-H. Lin and J. S. Vitter. Epsilon-Approximations with minimum packing constraint violation. In *Proceedings of the 24th ACM Symposium on Theory of Computing (STOC 1992)*, pages 771–782. ACM, 1992.

[73] Y. Lin and W. Li. Parallel machine scheduling of machine-dependent jobs with unit-length. *European Journal of Operational Research*, 156:261–266, 2004.

[74] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20:46–61, 1973.

[75] Y. Mansour and B. Patt-Shamir. Greedy packet scheduling on shortest paths. *Journal of Algorithms*, 14, 1993.

[76] Y. Mansour and B. Patt-Shamir. Many-to-one packet routing on grids. In *Proceedings of the 27th ACM Symposium on Theory of Computing (STOC 1995)*, pages 258–267. ACM, 1995.

[77] N. Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4:414–424, 1979.

[78] E. Minieka. Maximal, lexicographic, and dynamic network flows. *Operations Research*, 21:517–527, 1973.

[79] R. Moser and G. Tardos. A constructive proof of the general Lovász Local Lemma. *Journal of the ACM*, 57:1–15, 2010.

[80] I. Niven, H. S. Zuckerman, and H. L. Montgomery. *An Introduction to the Theory of Numbers, 5th edition.* John Wiley & Sons, 1991.

[81] R. Ostrovsky and Y. Rabani. Universal $O(congestion + dilation + \log^{1+\varepsilon} N)$ local control packet switching algorithms. In *Proceedings of the 29th ACM Symposium on Theory of Computing (STOC 1997)*, pages 644–653. ACM, 1997.

[82] P. Erdős and L. Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In *Infinite and Finite Sets*, volume 11 of *Colloq. Math. Soc. Janos Bolyai*, pages 609–627. North-Holland, 1975.

[83] C. Papadimitriou. *Computational complexity.* John Wiley & Sons, 2003.

[84] B. Peis, M. Skutella, and A. Wiese. Packet routing: Complexity and algorithms. In *Proceedings of the 7th Workshop on Approximation and Online Algorithms (WAOA 2009)*, volume 5893 of *Lecture Notes in Computer Science*, pages 217–228. Springer, 2010.

[85] B. Peis, M. Skutella, and A. Wiese. Packet routing on the grid. In *Proceedings of the 9th Latin American Theoretical Informatics Symposium (LATIN 2010)*, volume 6034 of *Lecture Notes in Computer Science*, pages 120–130. Springer, 2010.

[86] B. Peis, S. Stiller, and A. Wiese. Policies for periodic packet routing. In *Proceedings of the 21st International Symposium on Algorithms and Computation (ISAAC 2010)*, volume 6507 of *Lecture Notes in Computer Science*, pages 266–278. Springer, 2010.

[87] B. Peis and A. Wiese. Throughput maximization for periodic packet routing on trees and grids. In *Proceedings of the 8th Workshop on Approximation and Online Algorithms (WAOA 2010)*, volume 6534 of *Lecture Notes in Computer Science*, pages 213–224. Springer, 2011.

[88] B. Peis and A. Wiese. Universal packet routing with arbitrary bandwidths and transit times. In *Proceedings of the 15th Conference on Integer Programming and Combinatorial Optimization (IPCO 2011)*. Springer, 2011. to appear.

[89] Y. Rabani and É. Tardos. Distributed packet switching in arbitrary networks. In *Proceedings of the 28th ACM Symposium on Theory of Computing (STOC 1996)*, pages 366–375. ACM, 1996.

[90] P. Raghavan and E. Upfal. Efficient routing in all-optical networks. In *Proceedings of the 26th ACM Symposium on Theory of Computing (STOC 1994)*, pages 134–143. ACM, 1994.

[91] S. Rajasekaran. Randomized algorithms for packet routing on the mesh. Technical Report MS-CIS-91-92, Deptartment of Computer and Information Sciences, University of Pennsylvania, 1991.

[92] C. Scheideler. Offline routing protocols. In *Universal Routing Strategies for Interconnection Networks*, volume 1390 of *Lecture Notes in Computer Science*, pages 57–71. Springer, 1998.

[93] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003.

[94] A. S. Schulz and M. Skutella. The power of α-points in preemptive single machine scheduling. *Journal of Scheduling*, 5:121–133, 2002.

[95] P. Schuurman and G. J. Woeginger. Polynomial time approximation algorithms for machine scheduling: Ten open problems. *Journal of Scheduling*, 2:203–213, 1999.

[96] E. V. Shchepin and N. Vakhania. An optimal rounding gives a better approximation for scheduling unrelated machines. *Operations Research Letters*, 33:127–133, 2005.

[97] D. B. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62:461–474, 1993.

[98] D. Simchi-Levi. New worst-case results for the bin-packing problem. *Naval Research Logistics*, 41:579–585, 1994.

[99] M. Skutella. An introduction to network flows over time. In *Research Trends in Combinatorial Optimization*, pages 451–482. Springer, Berlin, 2009.

[100] W. E. Smith. Various optimizers for single-stage production. *Naval Research and Logistics Quarterly*, 3:59–66, 1956.

[101] A. Srinivasan and C.-P. Teo. A constant-factor approximation algorithm for packet routing and balancing local vs. global criteria. *SIAM Journal on Computing*, 30:2051–2068, 2001.

[102] S. Stiller and A. Wiese. Increasing speed scheduling and flow scheduling. In *Proceedings of the 21st International Symposium on Algorithms and Computation (ISAAC 2010)*, Lecture Notes in Computer Science, pages 279–290. Springer, 2010.

[103] S. Stiller and A. Wiese. Increasing speed scheduling and flow scheduling. Technical Report 007-2010, Technische Universität Berlin, February 2010.

[104] O. Svensson. Santa Claus Schedules Jobs on Unrelated Machines. In *Proceedings of the 43nd ACM Symposium on Theory of Computing (STOC 2011)*. ACM, 2011. to appear.

[105] V. V. Vazirani. *Approximation Algorithms*. Springer, Berlin, 2001.

[106] J. Verschae and A. Wiese. On the configuration-LP for scheduling on unrelated machines. In *Prceedings of the 19th European Symposium on Algorithms (ESA 2011)*, Lecture Notes in Computer Science. Springer, 2011. to appear.

[107] W. L. Wilkinson. An algorithm for universal maximal dynamic flows in a network. *Operations Research*, 19:1602–1612, 1971.

[108] D. P. Williamson, L. A. Hall, J. A. Hoogeveen, C. A. J. Hurkens, J. K. Lenstra, S. V. Sevast'janov, and D. B. Shmoys. Short shop schedules. *Operations Research*, 45:288–294, 1997.

[109] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. to appear.

[110] G. Yu and G. Zhang. Scheduling with a minimum number of machines. *Operations Research Letters*, 37:97–101, 2009.

[111] N. Zadeh. A bad network problem for the simplex method and other minimum cost flow algorithms. *Mathematical Programming*, 20:255–266, 1973.

[112] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3:103–128, 2007.