Fast Algorithms for

Two Scheduling Problems

Annamária Kovács

Dissertation
zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
Universität des Saarlandes

Saarbrücken, 2006

Datum des Kolloquiums:	14. 05. 2007
Dekan:	Prof. Dr. Thorsten Herfet
Prüfungsausschuss:	
Vorsitzender:	Prof. Dr. Raimund Seidel
Gutachter:	Prof. Dr. Kurt Mehlhorn
	Prof. Dr. Katalin Friedl
	Prof. Dr. Markus Bläser
Akademischer Mitarbeiter:	Dr. Holger Bast

Abstract. The thesis deals with problems from two distint areas of scheduling theory. In the first part we consider the *preemptive Sum Multicoloring* (pSMC) problem. In an instance of pSMC, pairwise conflicting jobs are represented by a conflict graph, and the time demands of jobs are given by integer weights on the nodes. The goal is to schedule the jobs in such a way that the *sum* of their finish times is minimized. We give the first polynomial algorithm for pSMC on paths and cycles, running in time $\mathcal{O}(\min(n^2, n \log p))$, where n is the number of nodes and p is the largest time demand. This answers a question raised by Halldórsson et al. [51] about the hardness of this problem. Our result identifies a gap between binary-tree conflict graphs – where the question is NP-hard – and paths.

In the second part of the thesis we consider the problem of scheduling n jobs on m machines of different speeds s.t. the makespan is minimized ($Q||C_{\max}$). We provide a fast and simple, deterministic *monotone* 2.8-approximation algorithm for $Q||C_{\max}$. Monotonicity is relevant in the context of *truthful mechanisms*: when each machine speed is only known to the machine itself, we need to motivate that machines 'declare' their true speeds to the scheduling mechanism. So far the best deterministic truthful mechanism that is polynomial in n and m, was a 5-approximation by Andelman et al. [3]. A *randomized* 2-approximation method, satisfying a weaker definition of truthfulness, was given by Archer and Tardos [4, 5]. As a core result, we prove the conjecture of Auletta et al. [8], that the greedy list scheduling algorithm Lpt is monotone if machine speeds are all integer powers of two (2-divisible machines).

Proving the worst case bound of 2.8 involves studying the approximation ratio of Lpt on 2-divisible machines. As a side result, we obtain a tight bound of $(\sqrt{3}+1)/2 \approx 1.3660$ for the 'one fast machine' case, i.e., when $m-1$ machine speeds are equal, and there is only one faster machine. In this special case the best previous lower and upper bounds were $4/3 - \epsilon < Lpt/Opt \leq 3/2 - 1/(2m)$, shown in a classic paper by Gonzalez et al. [42]. Moreover, the authors of [42] conjectured the bound 4/3 to be tight.

Thus, the results of the thesis answer three open questions in scheduling theory.

Kurzzusammenfassung. In dieser Arbeit befassen wir uns mit Problemen aus zwei verschiedenen Teilgebieten der Scheduling-Theorie. Im ersten Teil betrachten wir das sog. *preemptive Sum Multicoloring* (pSMC) Problem. In einer Eingabe für pSMC werden paarweise Konflikte zwischen Jobs durch einen Konfliktgraphen repräsentiert; der Zeitbedarf eines Jobs ist durch ein ganzzahliges, positives Gewicht in seinem jeweiligen Knoten gegeben. Die Aufgabe besteht darin, die Jobs so den Maschinen zuzuweisen, daß die *Summe* ihrer Maschinenlaufzeiten minimiert wird.

Wir liefern den ersten Algorithmus für pSMC auf Pfaden und Kreisen mit polynomieller Laufzeit; er benötigt $\mathcal{O}(\min(n^2, n \log p))$ Zeit, wobei n die Anzahl der Jobs und p die maximale Zeitanforderung darstellen.

Dies liefert eine Antwort auf die von Halldórsson et al. [51] aufgeworfene Frage der Komplexitätsklasse von pSMC. Unser Resultat identifiziert eine Diskrepanz zwischen der Komplexität auf binären Bäumen – für diese ist das Problem NP-schwer – und Pfaden.

Im zweiten Teil dieser Arbeit betrachten wir das Problem, n Jobs auf m Maschinen mit unterschiedlichen Geschwindigkeiten so zu verteilen, daß der *Makespan* minimiert wird ($Q||C_{\max}$). Wir präsentieren einen einfachen deterministischen *monotonen* Algorithmus mit Approximationsgüte 2.8 für $Q||C_{\max}$. Monotonie ist relevant im Zusammenhang mit *truthful* Mechanismen: wenn die Geschwindigkeiten der Maschinen nur diesen selbst bekannt sind, müssen sie motiviert werden, dem Scheduling Mechanismus ihre tatsächlichen Geschwindigkeiten offenzulegen.

Der beste bisherige deterministische truthful Mechanismus mit polynomieller Laufzeit in n und m von Andelman et al. [3] erreicht Approximationsgüte fünf. Eine *randomisierte* Methode mit Approximationsgüte zwei, die jedoch nur eine schwächere Definition von truthful Mechanismen unterstützt, wurde von Archer und Tardos [4, 5] entwickelt. Als ein zentrales Ergebnis beweisen wir die Vermutung von Auletta et al. [8], daß der *greedy list-scheduling* Algorithmus LPT monoton ist, falls alle Maschinengeschwindigkeiten ganze Potenzen von zwei sind (2-*divisible* Maschinen).

Der Beweis der obigen Approximationsschranke von 2.8 benutzt die Approximationsgüte von LPT auf 2-divisible Maschinen. Als Nebenresultat erhalten wir eine scharfe Schranke von $(\sqrt{3}+1)/2 \approx 1.3660$ für den Fall 'einer schnellen Maschine', d.h. $m-1$ Maschinen haben identische Geschwindigkeiten und es gibt nur eine schnellere Maschine. Die bisherigen besten unteren und oberen Schranken für diesen Spezialfall waren $\frac{4}{3} - \epsilon < Lpt/Opt \leq \frac{3}{2} - \frac{1}{2m}$. Letztere wurden 1977 von Gonzalez, Ibara und Sahni [42] bewiesen, die mutmaßten, daß die tatächliche obere Schranke bei 4/3 läge.

Alles in allem, liefert diese Arbeit Antworten auf drei offene Fragen im Bereich der Scheduling-Theorie.

Acknowledgements. A considerable part of this work was supported by the Marie Curie Fellowship Programme of the EU. On the personal side, I am indebted to two people in the first place: To Katalin Friedl, my advisor at the Technical University of Budapest, for taking up the tedious task of proof-reading this, and many previous write-ups of my results, and consistently urging me on clearer and more accurate formulation; and to Kurt Mehlhorn at the Max Planck Institute for Informatics in Saarbrücken, who was open to give me a chance at the beginning, and generous to let me carry out this work among the invaluable circumstances of the institute, over the last couple of years.

Further, I am thankful to Lajos Rónyai from the Computation and Automation Research Institute in Hungary, for his advice and encouragement at the start. I have good remembrance of the first common steps with Dániel Marx, whose PhD thesis has proved to be the perfect reference on graph-coloring problems. I would like to thank Martin Skutella for turning my attention to the monotone scheduling problem, and Vincenzo Auletta from Salerno, Jiří Sgall from Prague, and Tamás Kis from Budapest for their interest and for the fruitful personal and e-mail discussions on this problem. I am also grateful to those anonymous referees along the way, who were good-willing, or maybe sometimes enthusiastic enough, not to be scared away by the inevitable technicalities in the presentation.

Many thanks to Arno and Eric, for the jolly atmosphere I enjoyed whenever I worked in our office. Last but not least, I thank, with love, for the patience and help of my family, especially my husband Ulrich Meyer, without whose steady support and belief I would not have managed.

I dedicate this work to my father.

Contents

“ ... Különben is minden olyanná formálódik, amilyenné formálni tudjuk.”

(Eörsi István)

“ ... Anyway, everything gets formed into what we are able to form it.”

(István Eörsi)

Chapter 1

Introduction

Scheduling is a research area concerned with the optimal allocation of scarce resources over time to a set of tasks or activities – most surveys about scheduling start with some variant of this sentence [68, 85, 86].

This thesis deals with two completely different problems. Both of them adhere to the above characterization of scheduling. Still, we start this introduction by making it clear that our first problem – *sum multicoloring* – is not a classic one in scheduling theory, and thus is not listed among the traditional types of problems in most books or surveys in this field. Graph coloring models are typically applied for those scheduling problems, in which resources are limited by dependencies among subsets – or pairs – of tasks, and not by some global cardinality constraint (i.e., by the number of machines).

Although sum multicoloring models a special problem in *resource-constrained scheduling* [18, 68], in particular in *multiprocessor task scheduling* [17, 20, 21, 34], the theory of these nonstandard scheduling problems usually considers more general variants. Here we will concentrate on the specific multicoloring model – while keeping much of the scheduling terminology. We admit to have often regarded our problem simply as a nice mind-teaser in graph coloring.

The second topic concerns *scheduling on related machines*, which – in contrast to the previous one – is among the oldest problems in the history of scheduling. It is closely related to the classic optimization problem of bin-packing with variable bin sizes. Following the three-field classification of Graham, Lawler, Lenstra and Rinnooy Kan [43], this problem is denoted by $Q||C_{\max}$.

We consider the *largest processing time* (LPT) heuristic, which (on identical machines) was analyzed as one of the very first approximation algorithms [45]. The study of $Q||C_{\max}$ and LPT started nearly 40 years ago, at the same time when scheduling arose as an independent discipline due to the seminal survey of Conway, Maxwell, and Miller [31].

In our main result, we examine the *monotonicity* property of LPT, which gained relevance recently in the context of algorithmic mechanism design [5]. This in turn, moves also our second theme more into the scope of nonstandard scheduling problems, in the sense that originally we are motivated by strategical questions – like the acquisition and paying of machines.

There was no reason for trying to integrate these two topics into a common framework in any aspect. In particular, it was natural to stay with the more or less traditional terminology settled in the literature concerning the respective problems. So it happens for instance, that we denote the *size* or *demand* of the ith job by $x(i)$ in Chapter 2, and by t_i in Chapter 3. As another technical difference, $x(i)$ takes integer values, whereas it simplifies the discussion of the monotone scheduling problem, if we regard the t_i as real numbers.

Finally, let us also mention the similarities in the two cases. First, in both we deal with *deterministic problems* and *deterministic algorithms*, where by deterministic problems we mean in the strict sense that they are also *offline*, that is all information defining a problem instance is known in advance. Second, what certainly makes the two problems resemble, is the fact that both of them could be solved by way of some most elementary mathematical reasoning. This circumstance can be best illustrated with the words of Parker [85]: "... the appeal for me ... was almost exclusively nurtured by the purely combinatorial issues embodied in most scheduling problems – especially ones that seemed particularly 'simple'. "

Sections 1.1 and 1.2 of the Introduction treat the topics of 'sum-multicoloring' and 'monotone scheduling', respectively. In both sections, we start with the definition of the problem; then we provide some motivation, and cite the most relevant related work; finally we sketch the results presented in the thesis, with some intuition about the main ideas.

The thesis consists of two chapters, *Path multicoloring* and *Monotone scheduling*, according to the two main topics. We provide a more detailed outline at the beginning of each chapter. The order how we discuss the two problems is merely 'historical': it reflects the time order in which we investigated these questions.

1.1 Sum-multicoloring

The input of a graph multicoloring problem in its purest form consists of a simple undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and positive integer weights $x(v)$ on each node $v \in \mathcal{V}$. The output is a proper multicoloring, i.e., a function $\Phi : \mathcal{V} \to 2^{\mathbb{N}}$ which assigns a set $\Phi(v)$ of positive integers (colors) to each $v \in \mathcal{V}$ s.t. $|\Phi(v)| = x(v)$ and the sets assigned to adjacent vertices do not intersect.

Let $f(v) = \max \Phi(v)$ be the largest number assigned to v. A traditional optimization goal is to minimize $\max_{v \in \mathcal{V}} f(v)$, that is, the total number of colors used to color all the vertices. However, in the *sum multicoloring (SMC)* problems we aim at minimizing the function $\sum_{v \in \mathcal{V}} f(v)$. This objective function is useful if, e.g., SMC models the following scheduling problem:

The nodes of $\mathcal{G}$ represent jobs, each $v \in \mathcal{V}$ having size or **time demand** $x(v)$. The edges of $\mathcal{G}$ stand for pairwise conflicts between certain jobs, meaning that they cannot be processed at the same time, e.g., due to some non-shareable resource they use. With this interpretation $\mathcal{G}$ becomes a so-called **conflict graph**. The output Φ determines a proper schedule of the jobs, where conflicting jobs never receive the same time-unit.

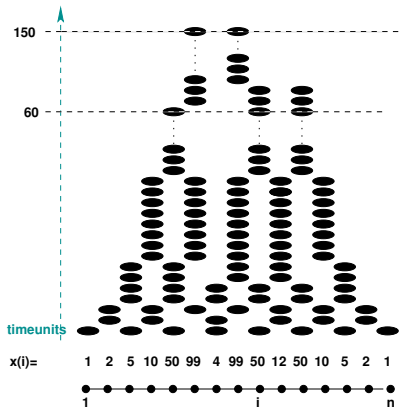

Figure 1.1: An instance of a sum multicoloring problem on a path conflict graph. The time demand $x(i)$ is written above each node i. The ellipses represent the assigned time-slots in an optimal solution.

The **finish time** of job v is modelled by $f(v)$. The traditional multicoloring problem corresponds to minimizing the *makespan* $\max_{v\in\mathcal{V}} f(v)$. In the sum multicoloring problem the *average* finish time of the jobs, or equivalently, the *sum* of finish times $\sum_{v\in\mathcal{V}} f(v)$ has to be minimized.

Notice that a classic multicoloring problem can be transformed into an ordinary coloring problem by replacing each node v with a clique of size $x(v)$ (cf. [49]). However, this reduction fails as soon as one regards the min-sum objective.

The view of SMC as a scheduling problem motivates the following distinction: in **non-preemptive SMC (npSMC)** the assigned sets $\Phi(v)$ must be contiguous, whereas in **preemptive SMC (pSMC)** the $\Phi(v)$ are arbitrary sets.

In the first part of this thesis we consider the preemptive sum multicoloring problem on *path* and *cycle* conflict graphs (see Fig. 1.1).

1.1.1 Motivation and background

As noted above, sum multicoloring problems find their primary applications in scheduling problems, where the jobs need the exclusive access to dedicated resources or machines, and they have to compete with other jobs for the use of these resources. An edge in the conflict graph $\mathcal{G}$ of jobs indicates that these two jobs use the same resource. Optimizing the makespan favours the system. However, from the jobs' point of view it is a reasonable goal to minimize the sum of completion times. *Resource-constrained scheduling* is the field dealing with these kinds of questions, even in quite general settings (the jobs might have to be assigned to machines *and* use dedicated resources; the resources might be shareable to a certain extent, etc.).

If in SMC all jobs are of unit length, we obtain the *sum coloring (SC)* problem. This special case has numerous applications. The first two we mention below belong to those rare examples, where the colors do not represent assigned time slots, but track assignment in the first case, resp. priorities in the second. These do not generalize to sum *multi*coloring.

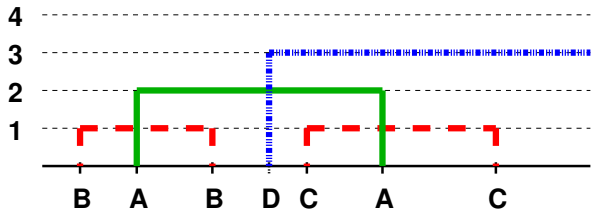

Figure 1.2: The 'over-the-cell routing' problem in VLSI design.

A basic motivation for the study of the SC problem appeared in the context of *VLSI design*, when Supowit [88] introduced the *optimum cost chromatic partition (OCCP)* problem (see also [57, 63, 87]). In OCCP we have to color a graph using a given set of colors $\{c_1, c_2, \ldots, c_k\}$, and minimize the color-sum. A special application of OCCP is a VLSI layout problem, called *over-the-cell routing* [10, 80]. Here we are given a set of two-terminal nets, and a set of parallel, horizontal tracks of distance $1, 2, 3, \ldots$ from the baseline where the terminals lie (see Fig. 1.2). The nets are routed with a vertical connection from both terminals to the assigned track. Overlapping nets have to use different tracks. The goal is to minimize the total wiring length, which is twice the sum of distances of nets from assigned tracks, plus the constant horizontal cost. This is the same as the sum coloring problem, restricted to *interval graphs*.

Distributed resource allocation has been another motivation for sum coloring [10]. Most of the early work done on scheduling jobs with pairwise conflicts was based on the so called *dining philosophers* paradigm [9, 23, 32, 71], which relates to distributed systems: processors competing for resources – e.g., for a communication path – acquire these resources according to some local protocol. Many resource allocation algorithms use a preprocessing, which results in a coloring of the nodes [26]. The color of a processor is proportional to the *maximum length of waiting chain* for this processor, which is one of the relevant measures describing a resource allocation algorithm. Minimizing the sum of colors means minimizing the average of this measure over all processors.

A third application is related to *traffic intersection control* [15, 22, 55]. Here a 'job' consists of a platoon of cars which take the same route through the intersection. Two jobs conflict, if the corresponding cars would collide in the intersection. It is not unnatural to consider the min-sum objective, in view of the improvements in vehicle detection technology, which is/will be able to detect the exact condition of a crossing. The problem generalizes to multicoloring if the length of a job is defined by the number of cars making up the job. On the other hand, note that this problem is also of *persistent* resp. *online* nature.

For other motivations for sum coloring like *train scheduling* or *storage allocation*, see [52]. Now we turn to sum multicoloring. Our first example is *multiprocessor task scheduling*, both as a concrete application, and as the model within resource-constrained scheduling, best fitting our multicoloring problem.

Dedicated multiprocessor task scheduling [1, 20, 34]. In an instance of this scheduling problem each task requires the exclusive access to a set of dedicated processors. Two tasks conflict, if they have a common processor to use.

If every task needs *two* preassigned processors, we obtain *biprocessor task scheduling*. Such biprocessor tasks arise in mutual diagnostic testing of processors [64], or in case of scheduling file transfers in a *data migration* problem [30, 58]. Note that the conflict graph in this case is a *line graph*[1], since the tasks – nodes of the conflict graph – can be associated with pairs of processors – that is edges in, e.g., the so called *transfer graph*. This gives rise to the research of *minimum sum edge multicoloring* on various graph classes [74, 76].

If each task uses up to k resources, the conflict graph is an intersection graph of a collection of sets of size at most k [52, 58].

Session scheduling in local area networks [24, 52]. In this application pairs of nodes want to communicate in a network. Assume that the routes between any two nodes are fixed. If some node has to send data to some other node, a *session* is established between the source and the destination. Sessions ('jobs'), whose routes share at least one link, are in conflict. On a path network this may lead to the SMC problem on interval graphs, on a ring network to the SMC on circular arc graphs. Note however, that – like with traffic intersections – models for most of these problems eventually require a persistent [14] or online [55] setting.

Access to files [52]. In this case the jobs model processes running on different processors in a distributed computing environment, and they share access to a set of files stored on the network file system. We assume that the underlying network is fully connected. The processes require read/write operations on certain subsets of the files, i.e., an exclusive access to these files throughout their execution. Two processes that need to work with the same file, have a conflict.

Related work.

The notion of sum coloring first appeared in the late 80's from two different sources. In graph theory, Kubicka [65] defined and studied the chromatic sum of graphs (see also [66, 67]). On the other hand, Supowit introduced the more general OCCP problem (see above), from its application in VLSI design [88].

The sum coloring problem was shown to be NP-hard on several graph classes, like line graphs [10], planar graphs [49]; or even hard to approximate within a constant factor like on bipartite graphs [13], and interval graphs [41]. Bar-Noy et al. investigated the SC problem in detail on general graphs, bounded-degree graphs and line graphs [10].

In 1999 Bar-Noy, Halldórsson, Kortsarz, Salman, and Shachnai introduced the sum multicoloring problem, and studied SMC on many different types of graphs in the preemptive, the non-preemptive and co-scheduling paradigms [12, 11]. Notice

[1] A line graph $L(\mathcal{G})$ is the edge-adjacency graph of a graph $\mathcal{G}$: two vertices of $L(\mathcal{G})$ are joined, if the corresponding edges share a vertex in $\mathcal{G}$.

	SC				SMC			
	lower bound		upper bound		non-preemptive		preemptive	
general graphs	$n^{1-\epsilon}$	*[10]	$n/\log^2 n$	$\Leftarrow$	$n/\log n$	*[12]	$n/\log^2 n$	*[12]
perfect graphs	$c > 1$	[13]	3.591	[38]	$\mathcal{O}(\log n)$	[12]	5.436	[38]
interval graphs	$c > 1$	[41]	1.796	[52]	11.273	[38]	5.436	$\Downarrow$
line graphs	NPC	[10]	2	[10]	7.682	[38]	2	[12]
partial k-trees			1	[56]	FPTAS	[49]	PTAS	[49]
trees			1	[65]	1	*[50]	PTAS	*[50]
paths			1	trivial	1	* $\Downarrow$	**1**	**thesis**

Table 1.1: Results for sum (multi)coloring problems on a few graph classes. An entry '1' stands for an exact polynomial algorithm. An arrow in the field means that the result follows by inference, either by containment of graph classes, or by SC being the special case of SMC. Most of the table was taken over from [38].

that hardness results for SC imply the hardness of the corresponding SMC problems. Recent research has mainly focused on the approximability and hardness of SC, pSMC, and npSMC on various graph classes that might be relevant for applications, like interval graphs, line graphs, bipartite graphs, and many others [38, 40, 49, 52, 58, 74, 80]. Besides, there are attempts to generalize the sum multicoloring problem in various ways, for instance by considering release dates or weighted versions [38, 52]. Table 1.1 shows the known approximation results for a few graph classes. Results marked with an '$*$', are sketched in Section 2.1.

Regarding the hardness of these problems on most considered graph classes, it is natural to search for types of graphs, for which exact polynomial algorithms exist. Currently trees constitute the boundary of what is efficiently solvable [50]:

Sum coloring. In [56] Jansen gave a polynomial algorithm for the OCCP problem on partial k-trees (graphs of treewidth at most k, cf. [19]). The result carries over to the SC problem.

npSMC. Halldórsson et al. studied the sum multicoloring problem on trees [50], and on partial k-trees [49]. For the *non-preemptive* SMC on trees they provided two efficient algorithms, which run in $\mathcal{O}(n^2)$ and $\mathcal{O}(np)$ time, respectively, where n is the number of vertices, and p is the maximum demand. On paths the first one has $\mathcal{O}(n \log p / \log \log p)$ running time.[2] In addition, on partial k-trees they gave a *fully* polynomial approximation scheme (FPTAS), i.e., they showed that the problem can be approximated within a factor of $(1 + \epsilon)$ in time polynomial in n and $1/\epsilon$.

pSMC. For the *preemptive* SMC on trees, Halldórsson et al. gave a polynomial time approximation scheme (PTAS). The hardness of pSMC on trees and paths was posed as an open question [51, 50]. The first answer was given by Marx [73], who proved that even on binary trees pSMC is *strongly* NP-hard (NP-hard even if p is

[2]Throughout the thesis, 'log' stands for logarithm to the base 2.

polynomial in n). With this result, the preemptive problem on trees turned out to be essentially harder than the non-preemptive version. As the second answer, in the thesis we provide the first polynomial algorithm on paths and cycles, running in time $\mathcal{O}(\min(n^2, n \log p))$.

In view of the above results, one might get the impression that a preemptive problem is in general more difficult than the non-preemptive counterpart. In order to make this picture more differentiated we quote the words of Halldórsson and Kortsarz [49]: "The preemptive and nonpreemptive cases turn out to require different treatments. Nonpreemptiveness restricts the form of valid solutions, which helps dramatically in designing efficient exact algorithms. On the other hand, approximation also becomes more difficult due to these restrictions."

Finally, let us mention a notion having obvious practical importance in sum coloring problems. Intuitively it is clear, that in general an optimal sum (multi)coloring of a graph $\mathcal{G}$ cannot be achieved using as many colors, as the chromatic number of $\mathcal{G}$. Indeed, for instance there exist trees for which an optimal sum coloring needs $\Omega(\log n)$ colors [67]. The **chromatic strength** of a graph is the minimum number of colors needed to color the graph optimally with regard to the min-sum objective. For results concerning chromatic strength the reader is referred to [47, 75, 77, 35, 87]. The same issue for multicoloring is considered in [49].

The WWW site [72] contains an up-to-date list of papers about sum multicoloring. Moreover, comprehensive surveys of related results can be found in [38, 49, 75]. See also Section 2.1, where we will sketch a few methods used to solve SMC-related problems.

1.1.2 Our contribution

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a path or cycle of n nodes, and $p = \max_{i \in \mathcal{V}} x(i)$, where $x(i)$ is the demand of node i. We provide the first exact, polynomial algorithm for the pSMC problem on these two types of conflict graphs, with running time $\mathcal{O}(\min(n^2, n \log p))$. Our result answers the question raised in [50], whether pSMC is efficiently solvable on paths. Besides, we identify a major gap between the solvability of the problem on binary trees – where it is NP-hard [73] – and graphs of maximum degree 2. For *large* p, our $\mathcal{O}(n^2)$ bound matches the $\mathcal{O}(n^2)$ bound for *non-preemptive* SMC on paths (or trees in general) given in [50].

Although the investigated graphs are of extremely simple structure, even on these simple graph classes the problem has proved to be far from trivial, and had previously been open since 1999 [51].

In [59] we gave a pseudo-polynomial algorithm for this problem, running in time $\mathcal{O}(n^3 p)$. In [61] we managed to improve on this earlier version, and achieve polynomial running time. We briefly sketch here the original idea, as well as the follow-up result concerning the structure of some 'nice' optimal schedules, which facilitated this improvement.

Let $\mathcal{G}$ be a path, and $\mathcal{V} = \{1, 2, \ldots, n\}$ denote its consecutive vertices. We add nodes 0 and $n + 1$ to the path, with demands $x(0) = x(n + 1) = 0$. We denote

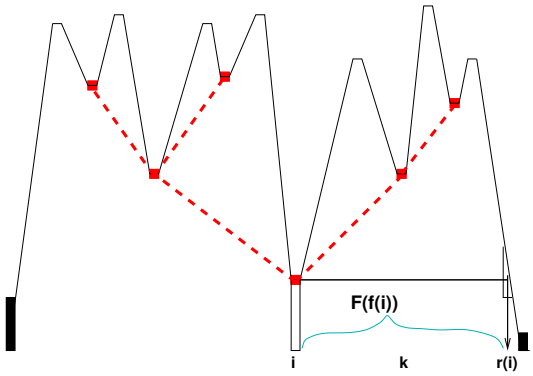

Figure 1.3: Minimum-tree of a block. One can best imagine the tree by depicting the vertices of the tree at the finish times of the respective loc-min nodes, the root being the loc-min of smallest finish time inside the block. The edges of the tree are drawn with broken lines.

by $\langle i, j\rangle$ the subpath $i, ..., j$. In a fixed schedule Φ, we distinguish so called local minimum *(loc-min)*, and local maximum *(loc-max)* nodes on the path: a loc-min node has smaller finish time, resp. a loc-max has larger finish time than any of its two neighbours. All other nodes are called *stairs*. We introduce a secondary objective function $\sum_{v\in\mathcal{V}} f^2(v)$. Restricted to optimal schedules with maximum value of $\sum_{v\in\mathcal{V}} f^2(v)$, the following hold:

The schedule of a loc-min node determines the schedule of the stairs up to the nearest loc-max's on both sides, in a greedy fashion; whereas a loc-max node receives the first time units idle on both sides. The hardness of this problem lies in the fact that loc-mins are not always *compact*, i.e., in general $f(i) \neq x(i)$ for a loc-min i. Let us denote by $r(i)$ and $\ell(i)$ the first nodes to the right and to the left of loc-min i with finish time less than $f(i)$ (see Fig. 1.3). The main idea is that any 'simultaneous permutation' of the time units in $\Phi(i)$ and $\Phi(r(i))$ results in the same optimum sum on the subpath strictly between i and $r(i)$. As a further consequence, $f(i)$ alone determines the optimum $\mathcal{F}(f(i)) := \sum_{j=i+1}^{r(i)-1} f(j)$ on this subpath. This observation facilitated the following dynamic programming algorithm in [59]:

Two scheduled subpaths are glued at a loc-min, proceeding from short subpaths to longer ones, from large loc-mins to smaller ones. However, on the subpath $\langle i+1, r(i)-1\rangle$ the optimum $\mathcal{F}(f(i))$ for every possible $f(i)$ has to be determined (since $f(i)$ depends also on what will be glued to the left of i).

The result of [61] is that the possible changes in $f(i)$ cannot modify the structure on $\langle i, r(i)\rangle$: the order of finish times of all the nodes remains the same. Moreover, by increasing $f(i)$ to $f(i)+\Delta$ – within a certain range –, we increase the finish times of all the loc-mins and decrease the finish times of all the loc-max's on $\langle i+1, r(i)-1\rangle$ by Δ. In total, we decrease $\mathcal{F}(f(i))$ by Δ. The optimum on $\langle i+1, r(i)-1\rangle$ is a simple linear function of the form $\mathcal{F}(f(i)) = \mathcal{C} - f(i)$. The only question for the dynamic algorithm is the constant $\mathcal{C}$ in this function and the range of possible $f(i)$ values.

What is the explanation for this determined structure on $\langle i, r(i)\rangle$? A subpath between nearest compact loc-mins will be called a *block* – notice that blocks are easy

to connect by a straightforward dynamic programming algorithm. Inside a block, the loc-mins are non-compact, and there must be 'large' differences between their demands. The loc-mins form a binary tree called *minimum-tree* (see Fig. 1.3), and the finish times of loc-mins on growing levels of this tree grow exponentially. In other words, if there are many loc-mins of similar demands, not separated by nodes of smaller demands, then some – typically, loc-mins of the same parity – will be compact. The exponential growth within a block implies that for fixed i there are just a constant number of candidates for $r(i)$, and just one or two candidates to be the loc-min finished next, i.e., the child of i in the minimum-tree.

The fast growth within a block is quite intuitive. However, the order of finish times must remain the same already on the lowest levels above $f(i)$, meaning roughly, that even the smallest differences between finish times on $\langle i, r(i)\rangle$ should exceed potential changes in $f(i)$. The proof of these results is lengthy and technical; it makes use of a handful of simple operations, and a tricky induction argument.

1.2 Monotone scheduling

$Q||C_{\max}$ denotes the offline task scheduling problem on *related* or *uniform* machines. In an instance of this problem we are given a **speed vector** $\langle s_1, s_2, \ldots, s_m\rangle$ representing the speeds of m machines, and a **job vector** $\langle t_1, t_2, \ldots, t_n\rangle$, where t_j is the **size** or **demand** of the jth job. Although job sizes usually have integral values, due to technical reasons here we assume that both machine speeds and job sizes are positive real numbers. Removing this assumption does not influence our results.

The goal is to assign the jobs to the machines, so that the overall finish time is minimized: if the set of jobs assigned to machine i is $\{t_{j_\gamma}\}_{\gamma=1}^{\Gamma}$ then the **work** assigned to i is $w_i := \sum_{\gamma=1}^{\Gamma} t_{j_\gamma}$ and the **finish time** of i is $f_i := w_i/s_i$; the **makespan** to be minimized is $\max_{i=1}^{m} f_i$.

This problem is NP-hard even on two identical machines [39], but it has polynomial approximation schemes [53, 54]. A classic, simple approximation algorithm for $Q||C_{\max}$ is the so called *'largest processing time first'* heuristic, or LPT for short [42]. This algorithm picks the jobs one by one in decreasing order of size, and always assigns the next job to the machine where it will have the smallest completion time.

In the above scheduling context, a *monotone* algorithm is defined as follows:

Definition 1.1 *Let A be an approximation algorithm for the $Q||C_{\max}$ problem. Suppose that on input $\langle s_1, \ldots, s_{i-1}, s_i, s_{i+1}, \ldots, s_m\rangle$ and $\langle t_1, t_2, \ldots, t_n\rangle$, A assigns work w_i to machine i; and on input $\langle s_1, \ldots, s_{i-1}, s'_i, s_{i+1}, \ldots, s_m\rangle$ and $\langle t_1, t_2, \ldots, t_n\rangle$, A assigns work w'_i to machine i. The algorithm A is* **monotone**, *if $(s_i \le s'_i) \Rightarrow (w_i \le w'_i)$ for $1 \le i \le m$.*

In the second part of the thesis we modify LPT so as to obtain a simple monotone algorithm, and investigate the worst case ratio of this new algorithm. This will lead to focusing on the following two types of speed vectors:

$(*)$ (*one fast machine*): $\quad s_1 = s_2 = \ldots = s_{m-1} = 1, s_m = s > 1;$

$(**)$ (*2-divisible speeds*): $\quad s_i = 2^{l_i}, \quad l_i \in \mathbb{Z}.$

1.2.1 Motivation and background

The motivation for the research on monotone algorithms originates in *mechanism design.* Before we proceed to a short historical background, we present our framework concerning *truthful mechanisms*:

We assume that the machines are owned by selfish agents, and the speed of each machine is private information to its agent. A **mechanism** is a pair $\mathcal{M} = (A, P)$, where A is an approximation algorithm to the scheduling problem, and $P = (P_1, P_2, \ldots, P_m)$ is a **payment scheme** (payment function). Let the job vector be fixed and public. Every machine (agent) i reports a **bid** b_i to be the inverse of her speed. With the job vector and the bid vector $\langle b_1, \ldots, b_m \rangle$ as input, $\mathcal{M}$ schedules the jobs using algorithm A, and pays each machine using the payment scheme P, where P depends on the schedule and on the bids. The **profit** of machine i is defined by $P_i - w_i/s_i$. The mechanism is **truthful**, if for every job vector, for every i and every s_i, and every possible bid vector of the other machines $b_{-i} := \langle b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_m \rangle$, bidding truthfully, i.e., $b_i = 1/s_i$ maximizes the profit of i.

The goal in general is to search for truthful mechanisms with an *efficient algorithm A* having a *good approximation* bound and an *efficient*ly computable *payment* scheme *P*. In particular, an **α-approximation mechanism** is one whose algorithm A provides an α-approximation. We say that the algorithm A **admits** a truthful payment scheme, if there exist payments P s.t. the mechanism (A, P) is truthful.

Applications of the mechanism design framework to diverse optimization problems arising in economics, game theory, and recently in computer science and networking constitute a widely studied area (cf. [5, 81, 84]). The above formulation concerning scheduling is one of the numerous examples.

Papadimitriou [84] writes that mechanism design could alternatively be called 'inverse game theory': "If Game Theory strives to understand rational behaviour in competitive situations, the scope of Mechanism Design is even grander: Given desired goals (such as to maximize a society's total welfare), design a game in such a clever way that individual players, motivated solely by self-interest, end up achieving the designer's goals."

Traditional mechanism design focuses on voting and auction type problems in economics, and considers situations when *players* or *agents* might try to manipulate the system and lie in order to maximize their own profit. Mechanisms that are able to arrange things – by making *payments* to the players – so that a rational player will never find it in her self-interest to lie, are called *truthful*. For each player, the profit to maximize, now consists of the payment plus the player's valuation of the outcome of the mechanism.

The most famous result in this field is the Vickrey-Clarke-Groves (VCG) mechanism [28, 46, 89], which relates only to so called 'utilitarian' objective functions – maximizing the sum of the agents' valuations –, typically modeling social welfare.

The emergence of the Internet as a primary platform for distributed computation made it unavoidable to consider similar socio-economic issues in theoretical computer science. Nisan and Ronen [82] did a pioneering work by first applying the standard tools of mechanism design to classic optimization problems in computer science,

including shortest paths, minimum spanning trees, and scheduling on unrelated machines. They look at general objective functions, since the utilitarian objective is not typical in these algorithmic models. Observe that the objective of $Q||C_{\max}$ is also non-utilitarian, since we aim at minimizing the maximum finish time, not the sum of finish times. Moreover, the same authors also show that even in the utilitarian model, the VCG payments do not induce truth telling, in case the output algorithm is suboptimal [83].

For a more detailed history of algorithmically-oriented mechanism design see, e.g., the paper of Archer and Tardos [5], which will be our starting point concerning results related to our work.

Related work.

The paper [5] studies the case, when each agent's secret data can be expressed by a single positive real number (s_i or $1/s_i$). The authors show that in models where the profit function has the above form $P_i - w_i/s_i$, a truthful mechanism $\mathcal{M} = (A, P)$ exists if and only if A is a monotone algorithm. In this case they also provide an explicit formula for the payment function (notice that the following notion of a *decreasing output function* corresponds to a monotone algorithm):[3]

Theorem 1.1 [5] *The output function admits a truthful payment scheme if and only if it is decreasing (i.e. $w_i(b_{-i}, b_i)$ is a decreasing function of b_i for all i and b_{-i}). In this case the mechanism is truthful if and only if the payments $P_i(b_{-i}, b_i)$ are of the form*

$$h_i(b_{-i}) + b_i w_i(b_{-i}, b_i) - \int_0^{b_i} w_i(b_{-i}, u)\, du$$

where the h_i are arbitrary functions.

Among other examples, Archer and Tardos consider the problem $Q||C_{\max}$. They show that – by some fixed order of the machines – the lexicographically minimal optimal solution is monotone. They also provide a fast *randomized* 3-approximation algorithm, that allows a mechanism, that is *truthful in expectation*, meaning that truth-telling maximizes the *expected* profit of each agent. This was later improved to a randomized 2-approximation mechanism [4].

The same problem, i.e., finding an efficient monotone approximation algorithm for $Q||C_{\max}$ is studied by Auletta, De Prisco, Penna, and Persiano [8]. The authors provide a deterministic, monotone $(4+\epsilon)-$approximation algorithm. They conjecture that the greedy heuristic LPT is monotone, if machine speeds are 2-divisible. They apply a *variant* of LPT, which is combined with the optimal schedule of the largest jobs, in order to give a reasonable approximation bound. As a consequence, the resulting algorithm is only polynomial in the number of jobs n, in particular,

[3] We remark that, in a different context, results analogous to Theorem 1.1 had been obtained by Myerson [79] two decades earlier. He had considered *incentive-compatible* (i.e. truthful) single-object auction mechanisms with *one-dimensional types* (i.e. one-parameter agents) and random valuation functions. For more on this and on monotonicity requirements in mechanisms with multi-dimensional types (like unrelated machine scheduling) see also [16].

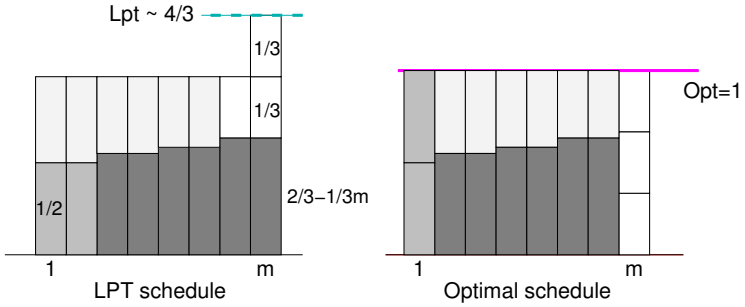

Figure 1.4: Sketch of the worst case example of Graham for identical machines.

it runs in time $\Omega(\exp(m^2/\epsilon))$. This result was considerably improved by Andelman, Azar, and Sorani [3], who presented a FPTAS for the case of constant m, and a 5-approximation algorithm for arbitrary m.

The papers cited in this paragraph will be sketched in more detail in Section 3.1.

Performance guarantee of LPT**.** On *identical* machines, the 'largest processing time' algorithm was first studied by Graham [45]. For a particular instance of the scheduling problem let *Lpt* denote the makespan produced by the LPT schedule, and *Opt* denote the optimum makespan. Graham's result is that on m identical machines $Lpt/Opt = \frac{4}{3} - \frac{1}{3m}$ in the worst case (see Fig. 1.4).

The approximation ratio of LPT for *arbitrary* machine speeds was first considered by Gonzalez, Ibarra, and Sahni in [42], where the authors prove that $Lpt/Opt \leq \frac{2m}{m+1} < 2$, whereas for any $\epsilon > 0$ an instance exists so that $Lpt/Opt > 3/2 - \epsilon$. These bounds were later improved to $(1.512, 19/12)$ by Dobson [33], respectively to $(1.52, 1.67)$ by Friesen [37].

The case of one fast machine (case $(*)$) has been studied in a number of papers. Liu and Liu [70] give approximation bounds in terms of m and s for a variation of LPT, and for *list schedules* – i.e., the case when jobs are given in any fixed order.

Gonzalez, Ibarra, and Sahni [42] obtain the lower and upper bounds $\frac{4}{3} - \epsilon < Lpt/Opt \leq \frac{3}{2} - \frac{1}{(2m)}$. Their lower bound instance differs from that of Graham for identical machines: here for any $m \geq 3$ and any ϵ an instance exists s.t. $\frac{4}{3} - \epsilon < Lpt/Opt$. For $m = 2$ they prove the tight bound of $\frac{1+\sqrt{17}}{4}$.

Cho and Sahni [25] analyze general list schedules for both arbitrary machine speeds and for case $(*)$. For the latter they obtain the tight bound $\frac{1+\sqrt{5}}{2}$ if $m = 2$, and $3 - 4/(m+1)$ if $m \geq 3$.

Li and Shi [69] consider the same special case, and suggest better heuristics than list scheduling for the online problem. Finally, for $m = 2$, Mireault, Orlin, and Vohra [78] provide a complete analysis of Lpt/Opt in terms of s_2/s_1.

Concerning the paper [42], see also Section 3.1.4.

1.2.2 Our contribution

As our primary result, we show a fast and simple, deterministic *monotone* algorithm for $Q||C_{\max}$, with approximation ratio 2.8. With input jobs ordered by size, it runs

in time $\mathcal{O}(m(n + \log m))$. This is an improvement over the 5-approximation bound of Andelman et al. [3]. As compared to the algorithms of Archer and Tardos [4, 5], no randomization is needed, and a stronger definition of truthfulness is fulfilled. An earlier version of this work was presented in [60]. Our approach can be sketched as follows:

We prove the conjecture of Auletta et al. [8], that LPT is monotone on 2-divisible machines. For arbitrary input speeds, the monotone algorithm LPT* is as simple as to run LPT with machine speeds rounded to powers of 2, and eventually reorder machines of equal rounded speeds according to their received work.

We show the monotonicity of LPT, by comparing two schedules denoted by I and II, where both are results of LPT on the same input, except that the machine speed s_k of I is increased to $s'_k = 2s_k$ in II. Let k' be the machine of doubled speed in schedule II. The proof involves a rather technical case distinction based on the number of jobs assigned to machine k' in II, and on the ratio t_n/t_a, where t_a is the first job assigned to k'. If t_n is not much smaller than t_a, then a simple counting of the jobs assigned to each machine yields the proof. If $t_n \ll t_a$, then we need to compare the total work received by each machine in I, respectively in II.

Worst case bounds. After the proof of monotonicity, our goal is to show a good approximation bound for LPT in case of 2-divisible speeds (case $(**)$).

As an additional benefit, we obtain new results for 'one fast machine' (case $(*)$). This special case was first studied 30 years ago [70], and to the best of our knowledge, its worst case ratio was only known to be in the interval $[\frac{4}{3},\ \frac{3}{2} - \frac{1}{2m}]$ [42]. Moreover, Gonzalez et al. [42] conjectured the lower bound 4/3 to be tight. We show that the conjecture does not hold: we provide an asymptotically tight bound of $\frac{\sqrt{3}+1}{2} \approx 1.3660$ for the worst case of Lpt/Opt in case $(*)$, and lower and upper bounds within $[1.367, 1.4]$ in case $(**)$. These results appeared in [62].

We present an instance (Instance A) of the scheduling problem with speed vector $\langle 1, 1, \ldots, 1, 2^r \rangle$ $(r \in \mathbb{N})$, so that for this instance $Lpt/Opt > \frac{\sqrt{3}+1}{2} - \epsilon$ for arbitrary $\epsilon > 0$, if r and m are large enough – we remark that using $s_m = 2^r$ instead of an arbitrary $s > 1$ is not essential. With this we improve on the previously known lower bound $4/3 - \epsilon$ for the approximation ratio of LPT in case $(*)$ as well as in case $(**)$. After that, we show that the new asymptotic lower bound $\frac{\sqrt{3}+1}{2}$ is actually tight in case $(*)$, i.e., for any instance with one fast machine $Lpt/Opt < \frac{\sqrt{3}+1}{2}$ holds.

On the other hand, for case $(**)$ we show that the lower bound $\frac{\sqrt{3}+1}{2}$ is not tight: we construct an instance on 2-divisible machines having asymptotic worst case ratio $\frac{(\sqrt{409}+29)}{36}$, where $\frac{(\sqrt{409}+29)}{36} \approx 1.3673 > \frac{\sqrt{3}+1}{2}$. However, this instance relies on calculation with exact job sizes, completion times etc., and is valid only if LPT favours faster machines in case of ties. Assuming that LPT breaks ties arbitrarily, we give another instance with asymptotic ratio $955/699 \approx 1.3662 > \frac{\sqrt{3}+1}{2}$. Both of these instances are further developed variants of Instance A. Our contribution here is twofold: first, the slight improvement over $\frac{\sqrt{3}+1}{2}$ is of theoretical interest; second, the new instances give an impression about how troublesome it might be to obtain tight approximation bounds for 2-divisible machines. Instead, with some additional

effort, following the same lines as in case $(*)$ it was feasible to prove an upper bound of 1.4 for 2-divisible machines.

In [60] we gave a short proof that on 2-divisible machines, LPT is a 1.5-approximation algorithm. Chapter 3 begins with this proof, since it is relies on the same basic idea as the following, much more involved monotonicity and upper-bound proofs, and thus provides a good basis for understanding all the other arguments.

Truthful mechanism. In order to complete the description of a truthful mechanism, in the end of Chapter 3 we return to our monotone algorithm LPT*. We show that the upper bound 1.4 for LPT immediately yields the upper bound of 2.8 for Lpt^*; on the other hand – though less obviously –, the instance that provides the lower bound $\frac{\sqrt{3}+1}{2}$ can be turned into an instance for LPT*, showing that the approximation ratio Lpt^*/Opt can get arbitrarily close to $\sqrt{3}+1 \approx 2.732$.

After that, we integrate LPT* into the mechanism design framework: we consider the payment function P, that complements LPT* to a truthful mechanism $\mathcal{M} =$ (LPT*,P). We show that – just like in case of the monotone algorithms in [3, 5, 8] – the payments can be computed in polynomial time. Moreover, since we use rounded machine speeds, the work curve is a step function, so that calculating the integral term for the payment becomes very simple. Finally we turn to the issues of *voluntary participation* (the profit must not be negative), and *frugality* (the total payment exceeds the total cost by at most a logarithmic factor), and prove analogous results to those in [5].

Chapter 2

Path multicoloring

In this chapter we show a fast algorithm for the preemptive sum-multicoloring (pSMC) problem on path conflict graphs. With minor modifications, the algorithm is applicable on cycles. We discuss the necessary changes for cycles at the very end of the chapter.

Concerning these two graph classes, the special feature of the maximum degree being 2, allowed us to invent and make use of a completely independent framework. Thus, our method is self-contained, and is not based on previous work. However, in order to put our problem in perspective, we start the chapter with an informal description of a few related results (see also Table 1.1).

Recall that the input of SMC is given by a conflict graph of jobs $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and integer time demands $x(v)$ $(v \in \mathcal{V})$ for each job. The goal is to schedule the jobs with respect to the conflicts, so that the sum of finish times of jobs $\sum_{v \in \mathcal{V}} f(v)$ is minimized. For basic terminology, and a short intuition about our result, we refer back to Section 1.1.

Outline.

Section 2.1 provides a short outlook on some related problems and solutions. We turn to the pSMC problem on paths in Section 2.2, where we fix terminology and notation. We note that further definitions appear in Sections 2.3 and 2.4, as was reasonable for the flow of discussion. In Section 2.3 we narrow our focus to a particular type of schedule, and prove a couple of cute structural properties of these schedules. There we also give more insight into the basic ideas that facilitate a polynomial algorithm.

Sections 2.4 and 2.5 introduce elementary proving techniques, respectively demonstrate how to apply these techniques on some technical lemmas. The fast reader may consider to omit all but the definitions in this part.

Section 2.6 contains our main result, the *uniqueness theorem* – preceded by much preparation. We collected all the long and technical details and proofs from this section into Section 2.7. The latter can be skipped as a whole without losing the plot. Due to the uniqueness theorem, it is possible to give an exact solution to the pSMC problem on paths efficiently. In Section 2.8 we discuss auxiliary results that further reduce the running time.

The final, $\mathcal{O}(\min(n^2, n \log p))$ algorithm is described in Section 2.9, with Subsection 2.9.3 treating the similar case of cycle conflict graphs. We close the chapter with a discussion section.

At the beginning of each section, we provide some intuition about their contents. In order to get a high level impression about our result, we suggest to read only Sections 2.2, 2.3, 2.6, and 2.9.

2.1 Related work

2.1.1 Approximation of SC and SMC on general graphs [10, 11].

In [10], the following heuristic called MaxIS is suggested for the sum *coloring* problem (i.e., when all jobs have unit length): find a maximum independent set $\mathcal{V}' \subseteq \mathcal{V}$ in the conflict graph, assign the next smallest time-unit to the jobs in $\mathcal{V}'$, and finally delete the jobs of $\mathcal{V}'$ from the graph; iterate the process until all jobs are scheduled. This is shown to yield an efficient 4-approximation algorithm for SC on graph classes where a maximum cardinality independent set is polynomially computable, respectively a 4ρ-approximation, whenever an efficient ρ-approximation of maximum independent set is known.

Unfortunately, in general the independent set problem is not approximable within a factor of $\Omega(n^{1-\epsilon})$ for any $\epsilon > 0$, by the breakthrough result of Arora et al. [7]. In fact, based on an analogous theorem about the inapproximability of the chromatic number [36], the authors of [10] prove the same approximation lower bound of $\Omega(n^{1-\epsilon})$ for SC on general graphs. In particular, they show by a simple argument that any polynomial time $\rho(n)$-approximation algorithm for SC can be used to get a polytime $\mathcal{O}(\rho(n) \log n)$ approximation for the chromatic number.

The *multi*coloring (SMC) problems on general graphs (and also on special graph classes) are considered in [11]. The paper demonstrates that applying some straightforward variant of MaxIS in either the preemptive or the non-preemptive case would lead to bad approximation bounds. Instead, they suggest cute modifications in the two cases:

Non-preemptive SMC:

1. Round up each demand $x(v)$ to the nearest power of 2, with $x'(v)$ being the rounded demand.
2. Associate weight $1/x'(v)$ with node v.
3. Add edge (u, v) to the conflict graph if $x'(u) \neq x'(v)$.
4. Run MaxIS on the modified graph, by always selecting *weighted* maximum independent sets. In the selected set run all jobs to completion, then select the next independent set from the remaining graph (with original weights).

Observe, that the weights are inversely proportional to the time lengths, since favouring a long job against a short one would involve a large delay of the short job relative to its length.

The above algorithm is an $\mathcal{O}(\rho \log \min(n, p))$ approximation for npSMC, where ρ is the approximation ratio of the weighted independent set algorithm. The best known ratio for arbitrary graphs is $\rho = \mathcal{O}(n/\log^2 n)$ [48], which implies an approximation bound of $\mathcal{O}(n/\log n)$ for npSMC in the general case.

Preemptive SMC:

1. Associate weight $1/x(v)$ with node v.
2. Run MaxIS on the weighted graph, by always assigning the next time unit to jobs in the selected independent set. Delete a job v only if it has received $x(v)$ time units. The weight of v does not change until deletion.
3. For every t that is among the first $\lceil x(v)/2 \rceil$ time units assigned to v in 2., then eventually assign the time units $2t-1$ and $2t$ to v.

If the algorithm halts after step 2., then it yields a 4ρ-approximation for the *minimum sum of averages* objective, where the averages are taken over the received time units independently for each node. Duplicating the time units received up to the *median* in step 3., provides a 16ρ-approximation for the minimum sum of finish times objective. Applying the weighted independent set algorithm of [48] with $\rho = \mathcal{O}(n/\log^2 n)$, we obtain a $\mathcal{O}(n/\log^2 n)$-approximation algorithm for pSMC on arbitrary graphs.

2.1.2 Non-preemptive SMC on trees and paths [50].

In the non-preemptive problem, the assigned sets $\Phi(v)$ are contiguous. Therefore, in an optimum schedule, for any vertex v there is a simple path $v_0, v_1, \ldots, v_m = v$ called a *grounding sequence of* v, so that $\Phi(v_0) = [1,\ x(v_0)]$, $\Phi(v_1) = [x(v_0) + 1,\ x(v_0) + x(v_1)], \ldots,\ \Phi(v_m) = [(\sum_{i<m} x(v_i)) + 1,\ (\sum_{i<m} x(v_i)) + x(v_m)]$. Since in trees there is a unique path connecting any fixed pair of nodes, any node v has n possible grounding sequences, and at most n possible schedules $\Phi(v)$.

Assume that the tree is rooted in some vertex r. A simple dynamic programming algorithm, which proceeds from leaves to root, can compute a matrix A, where $A[u, v]$ is the optimum sum of finish times on the subtree T_u, given that u has a grounding sequence ending in v. The straightforward dynamic programming algorithm runs in $\mathcal{O}(n^3)$ time. A more careful implementation yields $\mathcal{O}(n^2)$ running time.

In the special case when the tree is a path, the authors of [50] show that the demands in a grounding sequence grow exponentially (in fact, $x(v_m) = m^{\Omega(m)}$), since otherwise it would be worth 'restarting' the sequence from time 0 at some internal vertex. Therefore, for the maximum demand $p = m^{\Omega(m)}$ holds, and any grounding sequence has length $\mathcal{O}(\log p/\log\log p)$, which facilitates a running time of $\mathcal{O}(n \cdot \log p/\log\log p)$ for the above algorithm.

In case of trees, there is also a trivial bound of $\mathcal{O}(d(v) \cdot p)$ on the number of potential schedules of a node v, where $d(v)$ is the degree of v. For small p, a modified dynamic programming algorithm achieves $\mathcal{O}(\sum_v d(v) \cdot p) = \mathcal{O}(np)$ time complexity.

2.1.3 The hardness of preemptive SMC on binary trees [73].

The NP-hardness of pSMC on binary tree conflict graphs is proven by a reduction from the maximum independent set problem. First of all, the *list multicoloring* problem is considered. In the input of list multicoloring, every node in the conflict graph has an integer (color-)demand, and also a set of allowed colors. The goal is to compute a valid multicoloring, so that each node receives the required number of colors, and only from its allowed set. There is a simple reduction from the independent set problem on an arbitrary graph $\mathcal{G}$, to list multicoloring of a star conflict graph: the allowed 'colors' of the center node are all the nodes of $\mathcal{G}$; the leaves of the star correspond to the edges of $\mathcal{G}$, each edge having only its two endpoints as allowed colors. The edges have color demand 1. There is an independent set of size K in $\mathcal{G}$, if and only if there is a list multicoloring of the star with demand K on the center node.

We note here, that as opposed to list multicoloring, for pSMC (and npSMC) a rather straightforward algorithm exists for star conflict graphs.

It is easy to modify the above reduction for binary trees, by using a path of length $2m-1$ instead of the center node of the star, where m is the number of edges in $\mathcal{G}$. The major difficulty in reducing the independent set problem to pSMC on binary trees is that in a sum multicoloring problem no allowed color-lists are given in the input. The author of [73] manages to 'simulate' the color-lists by attaching certain binary tree graphs called *penalty gadgets* to the nodes of the original binary tree, which force the original nodes to use only the allowed color set, or otherwise they incur a big loss relative to the optimum cost. Furthermore, both the created binary tree and the demands remain polynomial in the size of $\mathcal{G}$, implying that pSMC is *strongly* NP-complete on binary trees.

2.1.4 PTAS for preemptive SMC on trees [49, 50].

For any given $\epsilon > 0$, there exists a restricted type of schedule of cost at most $(1+\epsilon)^2 \cdot opt$, so that there are at most $(1/\epsilon)^{\mathcal{O}(\log p)}$ possible (restricted) schedules of each node v. We obtain the restricted solutions as follows:

A short argument shows that an optimum solution on bipartite graphs uses at most $\mathcal{O}(p \cdot \log n)$ time units. This time spectrum is partitioned into $\mathcal{O}(\log p)$ layers $[q_{i-1},\ q_i)$, where $q_i = (1+\epsilon)^i$. Each layer is divided evenly into $1/\epsilon$ time sections, and any such section is either completely assigned to a node, or not at all. Furthermore, within a single layer, the *number* of assigned sections determines the schedule: let $\mathcal{V} = \mathcal{A} \cup \mathcal{B}$ be a bipartition of the tree, then the nodes in $\mathcal{A}$ get the first consecutive sections, and the nodes in $\mathcal{B}$ get the last consecutive sections of the layer.

The error of the restricted schedule comes from two factors: In every layer, and thus in the topmost nonempty layer of a node, the schedule is not the optimal schedule, which may increase the finish time by a factor of $(1+\epsilon)$. On the other hand, the number of assigned time units inside a layer must be a multiple of $\epsilon \cdot (q_i - q_{i-1})$, which may blow up the optimum schedule size in every layer by another factor of $(1+\epsilon)$.

Of course, we need to take care that for neighboring nodes, the total number of received sections does not exceed $(1+\epsilon)(q_i - q_{i-1})$. Schedules of neighboring nodes are *consistent*, if they fulfil this condition for every layer.

A dynamic programming algorithm which proceeds from leaves to root, can compute the optimum on the subtree T_u for every node u, and for all the $(1/\epsilon)^{\mathcal{O}(\log p)}$ different restricted schedules of u : the algorithm selects an optimal schedule of each child of u, from those consistent with the schedule of u. This algorithm actually yields a $(1+\epsilon)$-approximation in time $n \cdot (p \cdot \log n)^{\mathcal{O}(1/\epsilon \cdot \log(1/\epsilon))}$.

An additional trick can bound the number of *segments* – i.e., nonempty layers – for each node by $\mathcal{O}(1/\epsilon^3 \cdot \log^2(1/\epsilon))$, so as to obtain $n \cdot 2^{\mathcal{O}(1/\epsilon \cdot \log(1/\epsilon))^3}$ running time. This method partitions the graph into subgraphs $\mathcal{G}_j$, such that the highest demand differs from the lowest demand by some constant factor in each $\mathcal{G}_j$. After that, it solves the problem using the previous algorithm independently on the subgraphs, and concatenates the solution along the time axis. The trick lies in a careful subdivision of the range of demands, which in turn determines the partition into subgraphs [49].

2.2 Preliminaries

Now we turn to our main topic. We consider preemptive SMC, and until the last section we assume that $\mathcal{G}$ is a path, where $\mathcal{V} = \{1, 2, \ldots, n\}$ denote the consecutive nodes (jobs). The function $\Phi : \mathcal{V} \to 2^{\mathbb{N}}$ is a **(proper) schedule**, if $|\Phi(i)| = x(i)$ and $\Phi(i) \cap \Phi(i+1) = \emptyset \quad (1 \leq i \leq n-1)$. For an illustration to the definitions see Figure 2.1.

We will refer to time units as to **levels**. Given a schedule Φ, we say that i is **black** on level φ, if $\varphi \in \Phi(i)$, and i is **white** on level φ if $\varphi \notin \Phi(i)$. For $i < j$ we will also say, that the ordered pair (i, j) is **black-white, black-black,... etc.** on level φ. Note that $(i, i+1)$ cannot be black-black on any level.

Definition 2.1 *An (i, j) pair* **has a conflict** *or* **is conflicting on level** φ*, if*
– i and j are of the same parity and (i, j) is black-white, or white-black, or
– i and j are of opposite parity and (i, j) is black-black, or white-white.

Observe that here the term 'conflict' means something else than a conflict (edge) in the conflict graph.

Obviously, if (i, j) is conflicting on level φ, then $\exists k \in \langle i, j-1 \rangle$ s.t. $(k, k+1)$ is white-white on level φ.

Definition 2.2 *For any ordered pair of nodes (i, j) and any fixed schedule Φ, we may consider the number of levels under some fixed level Γ, where (i, j) is black-black, white-black, white-white and black-white, respectively. We call the 4-tuple of these numbers the* **scheme of** (i, j) **below** Γ.

Definition 2.3 *Given a schedule, we say that node i is*
– a **loc-min,** *if $f(i-1) > f(i)$ and $f(i) < f(i+1)$, or $i = 0$, or $i = n+1$;*
– a **loc-max,** *if $f(i-1) < f(i)$ and $f(i) > f(i+1)$;*

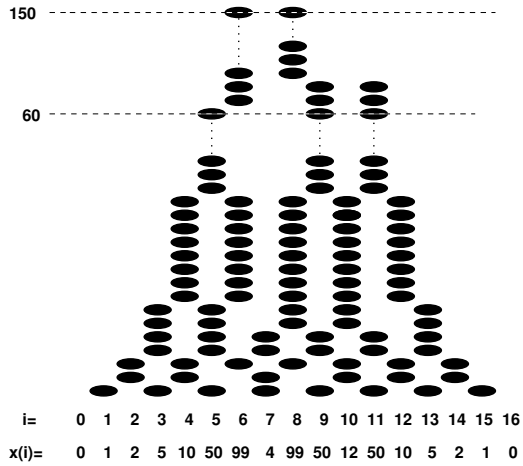

Figure 2.1: A solution for the given pSMC problem on a path of length 15. $\langle 0, 16\rangle$ is a block. Non-compact loc-mins are nodes 7 and 10. They are conflicting on level 2. Further examples: $r(7) = 14$, and $\ell(10) = 7$; $pit(7, 14) = 10$; $top(7, 10) = 8$; $pit(0, 16) = 7$.

– *a* **stair** *otherwise, in particular a* **stair-up,** *if* $f(i-1) < f(i) < f(i+1)$, *and a* **stair-down,** *if* $f(i-1) > f(i) > f(i+1)$;
– **compact,** *if* $f(i) = x(i)$. *Note that* 0 *and* $n+1$ *are compact.*

If both i *and* j *are compact with no compact nodes between them, then we call* $\langle i, j\rangle$ *or* (i, j) *a* **block**.

We provide with notation the following 4 nodes concerning a loc-min i :

Definition 2.4 *If* i *is a loc-min in a schedule, then we denote by* $r(i)$ *the first node to the right of* i *s.t.* $f(r(i)) < f(i)$ *and we denote by* $\ell(i)$ *the first node to the left of* i *s.t.* $f(\ell(i)) < f(i)$. *Furthermore,* $R(i) := r(i) - 1$ *and* $L(i) := \ell(i) + 1$.

Note that $R(i)$ $(L(i))$ is either a stair-down (stair-up) – like $R(10) = 12$ on Fig. 2.1 – or a loc-max – like $L(10) = 8$. The values $f(i)$, $r(i)$, $\ell(i)$, etc. may carry a subscript $_\Phi$ when they are relative to a schedule Φ that is not clear from the context. We end this section with the definition of our elementary operation:

Definition 2.5 *Let* φ, *and* ψ *be some levels such that* $\forall k \in \langle i, j\rangle$, $f(k) \geq \max(\varphi, \psi)$ *in a schedule* Φ. *We say that we* **exchange the levels** φ **and** ψ **on** $\langle i, j\rangle$, *if* $\forall k \in \langle i, j\rangle$ *we make* k *white (black) on level* φ *if and only if according to* Φ *it was white (black) on level* ψ, *and vice versa.*

The exchanging operation results in a proper schedule on $(i-1, i)$ (resp. on $(j, j+1)$), if $(i-1, i)$ was white-white in Φ on either φ or ψ. If on the other hand, $(i-1, i)$ becomes black-black on either φ or ψ, and $i-1$ or i is a loc-max, then we can correct the black-black level by increasing the finish time of this loc-max by one. If this correction is needed, we say that the level-exchange **costs 1** on the left (resp. on the right), otherwise we say that it **is for free** or **costs 0** on the left (on the right).

2.3 A simple type of schedule

We would like to restrict our attention to schedules of the simplest possible structure. To this end we optimize the following two objectives in this order: we minimize $\sum_{i=1}^{n} f(i)$, and maximize $\sum_{i=1}^{n} f^2(i)$. Intuitively, in such a schedule small $f(i)$ values are as small, and large $f(i)$ are as large as possible.

Definition 2.6 *A schedule Φ is called* **optimal schedule** *or a* **solution** *if the sum of finish times $\sum_{i=1}^{n} f_\Phi(i)$ is minimum over all schedules, and among those having minimum sum of finish times $\sum_{i=1}^{n} f_\Phi(i)^2$ is maximum.*

In the rest of the section we prove a couple of nice properties of optimal schedules. Having a clearer view about their characteristics, we provide further definitions, and more intuition about the main result which facilitates a fast algorithm.

From here on we analyze an arbitrary fixed optimal schedule Φ. For simplicity we will later refer to the properties below by (P1) to (P4). Notice that any schedule starts with a compact loc-min 0, that is followed by alternating loc-max's and loc-mins, till the last compact loc-min $n+1$.

Lemma 2.7 (P1) *In Φ, stairs and loc-max's are scheduled in the following greedy way: e.g., a stair-up s is black on the smallest $x(s)$ levels where $s-1$ is white; a loc-max m is black on the smallest $x(m)$ levels where both $m-1$ and $m+1$ are white.*

Proof. For illustration see Figure 2.2. We need to show that $(s-1, s)$ is not white-white on any level below $f(s)$. Suppose $(s-1, s)$ is white-white on level $\varphi < f(s)$. Let m be the first local maximum to the right of s, and let's exchange the levels φ and $f(s)$ on $\langle s, m\rangle$. Now we decreased $f(s)$ by at least one, and got a proper schedule on $\langle s-1, s\rangle$. If it is also a proper schedule on $\langle m, m+1\rangle$, then we decreased the optimum sum, a contradiction; if it is not a proper schedule on $\langle m, m+1\rangle$, then we correct it by increasing $f(m)$ by one (the exchange costs 1 on the right). Now we have the same sum of finish times, but increased the sum of squares of finish times, a contradiction. Thus, we have shown that $(s-1, s)$ is either black-white or white-black on any level below $f(s)$.

It is now also straightforward that a loc-max is scheduled greedily. □

Corollary 2.8 *The schedule of a loc-min determines the schedules of stairs leading up to the next loc-max's on both sides. If, e.g., s is a stair-up, then $f(s) = x(s-1) + x(s)$.*

Lemma 2.9 (P2) *Let i be a non-compact loc-min, and let $\ell(i) \le k \le r(i)$. Then*
– the (i, k) pair is not conflicting on the level $f(i)$;
– the (i, k) pair is not conflicting on any level where i is white.

Proof. Suppose for example, that $i < k \le r(i)$, and (i, k) is conflicting on the level $f(i)$. There must be an $i \le s < k$ such that $(s, s+1)$ is white-white on level $f(i)$. Let m be the first local maximum to the left of i. Now all the finish times on

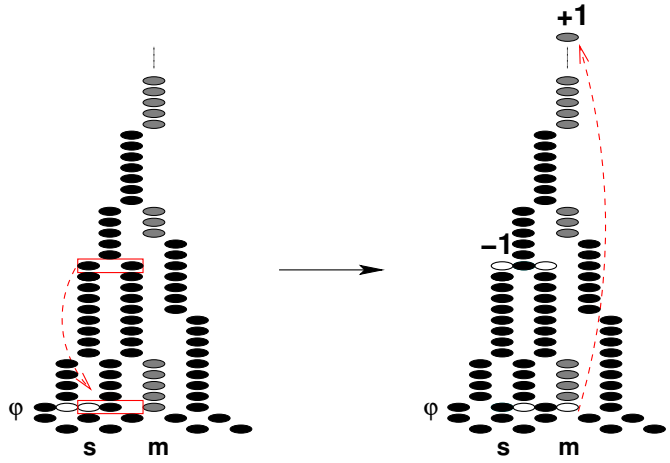

Figure 2.2: Proof of property (P1).

$\langle m, s\rangle$ are not smaller than $f(i)$. Since i is non-compact, there is a level $\varphi < f(i)$ where i is white. Let's exchange the levels φ and $f(i)$ on $\langle m, s\rangle$. Since $(s, s+1)$ was white-white, this remains a proper schedule on $\langle s, s+1\rangle$ and it reduces $f(i)$ by at least 1. If it is not a proper schedule on $\langle m-1, m\rangle$, we can correct it by increasing $f(m)$ by 1. This increases the sum of squares of finish times, a contradiction.

If (i, k) is conflicting on the level φ, the argument is essentially the same. □

Corollary 2.10 *Let i be a non-compact loc-min. Then $(i, r(i))$ is black-white on level $f(i)$, and they have no conflict, so i, and $r(i)$ are of different parity, i and $R(i)$ are of the same parity. The same holds for nodes i, $\ell(i)$ and $L(i)$. Furthermore, $\ell(i) \leq i-3$ and $i+3 \leq r(i)$.*

Corollary 2.11 *If i is a non-compact loc-min, then there are no compact nodes inside $\langle \ell(i), r(i)\rangle$, since any node is white either on level $f(i)$ or on the levels where i is white.*

Corollary 2.12 *Let i be a non-compact loc-min and let α denote the number of levels where $(i, r(i))$ is black-black. Then $\alpha = x(i) + x(r(i)) - f(i)$. Symmetric statement holds for $(\ell(i), i)$.*

Lemma 2.13 (P3) *If $i < j$ are loc-mins, $j < r(i)$ and $f(i) = f(j)$, then i and j are compact.*

Proof. Assume that, for example, i is non-compact, and it is white on some level $\varphi < f(i)$. Now i and j must be of the same parity, since on level $f(i) = f(j)$ they are both black and they cannot be conflicting according to (P2). In turn, by (P2) j is also white on φ.

Let m_i and m_j be the nearest loc-max's to the left of i, and to the right of j, respectively. Now we exchange the levels $f(i)$ and φ on $\langle m_i, m_j\rangle$. This operation costs at most 2 on the two sides, but we decreased both $f(i)$ and $f(j)$ by at least 1. Moreover, the sum of squares of finish times increased, contradicting the optimality of Φ. □

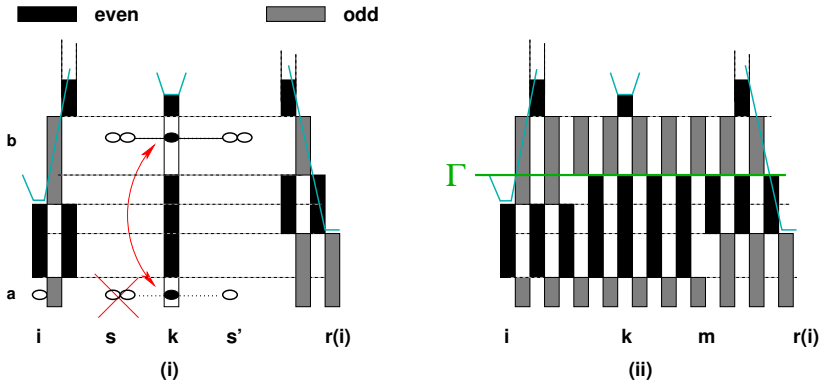

Figure 2.3: (i) Proof of property (P4); (ii) Corollary 2.16.

Definition 2.14 *Let i be a non-compact loc-min. We denote by $pit(i, r(i))$ the loc-min of smallest finish time between i and $r(i)$. If there is no loc-min between them, then $pit(i, r(i)) = \emptyset$ and in this case $top(i, r(i))$ denotes the unique loc-max in $\langle i, r(i)\rangle$. The definition of nodes $pit(\ell(i), i)$ and $top(\ell(i), i)$ is analogous.*

If $\langle g, h\rangle$ is a block, we define $pit(g, h)$ and $top(g, h)$ analogously.

Property (P2) implies that $k = pit(i, r(i))$ is non-compact, and due to (P3), the $pit()$ is well-defined, since there can be only one such node. Note that if $k = pit(i, r(i))$, then $\langle i + 1, \ell(k)\rangle$ is a series of stair-ups and $\langle r(k), r(i) - 1\rangle$ is a series of stair-downs.

Now if $g < h$ are nearest compact nodes, s.t. $x(g) > x(h)$, then $r(g) \leq h$. Instead of $\langle g, r(g)\rangle$, the block $\langle g, h\rangle$ is of relevance. Corollary 2.11 implies, that all loc-mins inside $\langle g, h\rangle$ have finish time above $x(g)$. Therefore, $\langle r(g), h\rangle$ is a series of stair-downs. In this sense blocks can be regarded as 'maximal $(i, r(i))$ or $(\ell(i), i)$ pairs', and it is reasonable to extend the definitions of $pit()$ and $top()$ to blocks.

If $k = pit(g, h)$ in some block $\langle g, h\rangle$, then a level $\varphi < f(k)$ may exist, where k is black and is conflicting with both $\ell(k)$ and $r(k)$. The next lemma states, that this cannot happen to any other loc-min inside the block.

Lemma 2.15 (P4) *If i is a non-compact loc-min and $k = pit(i, r(i))$ (or $k = pit(\ell(i), i)$) then there is no level below $f(k)$ where k has conflict with both $\ell(k)$ and $r(k)$.*

Proof. (See Figure 2.3.) Suppose we have $\ell(k) \leq s < k \leq s' < r(k)$ such that $(s, s + 1)$ and $(s', s' + 1)$ are both white-white on level $\psi < f(k)$. Since i is non-compact, there is a level $\varphi < f(i)$, where i is white. Property (P2) implies that node s is white either on level $f(i)$ or on level φ. W.l.o.g. assume that s is white on level φ. Now we exchange the levels ψ and φ on $\langle s+1, s'\rangle$, and denote the resulting schedule by Φ'. Now Φ' has the same finish times as Φ, moreover Φ' is a proper schedule, since $(s, s + 1)$ and $(s', s' + 1)$ were white-white on ψ. Note also, that $(s, s + 1)$ is white-white on level φ. Now we modify Φ' as in the proof of (P2), and obtain a schedule better than optimal, a contradiction. □

Corollary 2.16 *Let i be a non-compact, even loc-min, and $\psi \leq f(i)$ an arbitrary level.* (P1), (P2) *and* (P4) *imply the following on $\langle i, r(i)\rangle$:*

If $(i, r(i))$ have no conflict on level ψ, then exactly the odd or exactly the even nodes are black on level ψ.

If $(i, r(i))$ have a conflict on level ψ, then there is a loc-max m, s.t. on $\langle i, m-1\rangle$ the even nodes, on $\langle m+1, r(i)\rangle$ the odd nodes are black on level ψ.

Definition 2.17 *Let i be a non-compact, even loc-min. We call the levels where $(i, r(i))$ (resp. $(\ell(i), i)$) have no conflicts* **clear** *levels, in particular* **clear odd** *or* **clear even** *levels if the odd, resp. the even nodes are black on these levels.*

A level ψ where $(i, r(i))$ have a conflict, is an **even-odd level** *if it is clear even on $\langle i, m-1\rangle$ and clear odd on $\langle m+1, r(i)\rangle$. We say that $\langle i, m\rangle$ is the* **even part** *and $\langle m, r(i)\rangle$ is the* **odd part** *with respect to level ψ.*

Consider the subpath $\langle 10, 13\rangle$ in Fig. 2.1. Levels 6 and 7 are even-odd, all the other levels are clear levels.

2.3.1 The optimum on $\langle i, r(i)\rangle$

Let us fix a subpath $\langle i, j\rangle$, and a (partial) instance $\mathcal{I} := (x(i), x(i+1), ..., x(j))$ of the pSMC problem. We will consider all cases when $\mathcal{I}$ is part of an instance on a longer path, and all solutions (if existent) where i is a *non-compact* loc-min and $j = r(i)$. In this subsection we discuss only the potential $(i, r(i))$ pairs. The pairs $(\ell(i), i)$ can be handled symmetrically.

In the next proposition we claim that in any of these cases the optimum sum of finish times $\mathcal{F} := \sum_{k=i+1}^{r(i)-1} f(k)$ depends only on $f(i)$.

Proposition 2.18 *Let $\mathcal{I}$ be as defined above. Suppose that Φ is a solution to some extension of $\mathcal{I}$, s.t. i is a non-compact loc-min, and $j = r_\Phi(i)$. Similarly, let Ψ be a solution to another extension of $\mathcal{I}$, s.t. i is a non-compact loc-min in Ψ, and $j = r_\Psi(i)$ holds as well. Now if $f_\Phi(i) = f_\Psi(i)$, then $\mathcal{F}_\Phi = \sum_{k=i+1}^{r(i)-1} f_\Phi(k) = \sum_{k=i+1}^{r(i)-1} f_\Psi(k) = \mathcal{F}_\Psi$.*

Proof. Assume the contrary, that e.g., $\mathcal{F}_\Phi < \mathcal{F}_\Psi$. According to (P2) and Corollary 2.12, in both Φ and Ψ, the scheme of (i, j) below $f(i)$ is as follows: the number of white-white levels is 0, the number of black-black levels is $\alpha = x(i) + x(j) - f(i)$, the number of black-white levels is $x(i) - \alpha$ and the number of white-black levels is $x(j) - \alpha$. Now we permute the levels in Φ on $\langle i, j\rangle$, below $f(i)$, so that the schedule $\Phi(i)$ turns into $\Psi(i)$ and $\Phi(j)$ turns into $\Psi(j)$. This is possible, since the schemes of (i, j) were the same in the two schedules. Let's call this new partial schedule Φ'. Notice that the permutation may modify the schedule on levels not higher than $f(i)-1$. Therefore, it does not affect the finish times on $\langle i+1, j-1\rangle$, and so $\mathcal{F}_{\Phi'} = \mathcal{F}_\Phi < \mathcal{F}_\Psi$. Now on $\langle i, j\rangle$ we can exchange the schedule Ψ for Φ' and obtain a better optimum, a contradiction. □

We will consider all the $f(i)$ that ever occur in an optimal schedule where i is a non-compact loc-min, and $j = r(i)$, and fix one solution on $\langle i, r(i)\rangle$ for each of them.

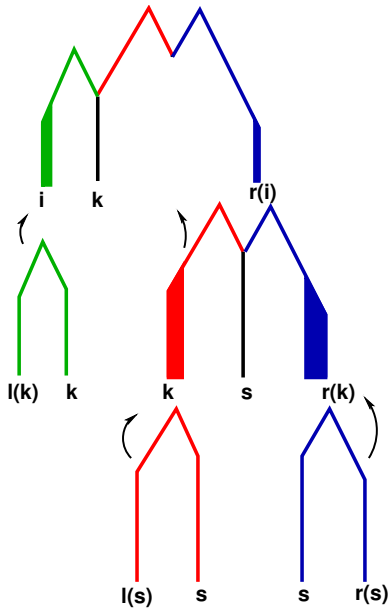

Figure 2.4: The dynamic programming strategy: Solutions on $\langle \ell(s), s\rangle$ and $\langle s, r(s)\rangle$ can be glued at s to form a solution on $\langle k, r(k)\rangle$, where $s = pit(k, r(k))$. The optimum function on $\langle k, r(k)\rangle$ is obtained from the optimum functions on $\langle \ell(s), s\rangle$ resp. on $\langle s, r(s)\rangle$, etc.

Here we only talk about fixed solutions modulo permutation of the levels below $f(i)$, but with fixed finish times on $\langle i, r(i) - 1\rangle$.

Recall that α denotes the number of black-black conflicts of $(i, r(i))$. Instead of treating the optimum as a function of $f(i)$, we will look at it as a function of α. Since $\alpha = x(i)+x(j)-f(i)$, this makes just a tiny technical difference. Let Φ_α stand for the corresponding fixed solution on $\langle i, r(i)\rangle$, and let $\mathcal{F}(\alpha)$ be the sum of finish times on $\langle i+1, r(i)-1\rangle$ in Φ_α. Those α values that correspond to an occurrence of $f(i)$ constitute the **domain** $\mathcal{D}_\mathcal{F}$ of $\mathcal{F}(\alpha)$. (The notation Φ_α, $\mathcal{F}(\alpha)$, and $\mathcal{D}_\mathcal{F}$ is always used for fixed i, j and $\mathcal{I}$.)

Observe that by Proposition 2.18, $\mathcal{F}(\alpha)$ does not depend on which of the optimal schedules we chose to be Φ_α. In our main result called **uniqueness theorem** (Theorem 2.3) we prove a much stronger statement: we show that Φ_α is essentially the same solution for all α, in the sense that for the range of possible α values, the order of finish times on $\langle i, r(i)\rangle$ is fixed. As a consequence, $\mathcal{F}(\alpha)$ is a simple linear function, namely $\mathcal{F}(\alpha-\Delta) = \mathcal{F}(\alpha)-\Delta$. We will apply a fast dynamic programming algorithm to compute the domain and one value of $\mathcal{F}(\alpha)$ for each potential $\langle i, r(i)\rangle$ (resp. $\langle \ell(i), i\rangle$) subpath. This computation proceeds from the short subpaths to longer ones, and constitutes Phase 1 of the scheduling algorithm (see Fig. 2.4).

Next, we fix a level $\Gamma(i, r(i))$ which, according to Proposition 2.20 below, separates the 'low levels' and the 'high levels' concerning $\langle i, r(i)\rangle$ (see Figure 2.3 (ii)). We will simply write Γ, whenever $(i, r(i))$ is clear from the context.

Definition 2.19 $\Gamma(i, r(i)) := \min(x(i) + x(i+1), x(r(i)) + x(r(i)-1))$.

Proposition 2.20 *In any of the fixed solutions* Φ_α, $f(i) \leq \Gamma$, *and either* $pit(i, r(i)) = \emptyset$, *or* $f(pit(i, r(i))) > \Gamma$.

Proof. By (P2), there are no white-white levels of $(i, i+1)$ below $f(i)$, so $f(i) < x(i) + x(i+1)$. Similarly, there are no white-white levels of $(r(i)-1, r(i))$, since such a level would be a conflicting (white-white) level of $(i, r(i))$. Therefore, $f(i) \leq x(r(i)) + x(r(i)-1)$ holds as well.

Let $k = pit(i, r(i))$. Since k is non-compact, it has the same parity as $L(k)$ and $R(k)$ (see Corollaries 2.10 and 2.11). On the other hand, $i+1$ and $r(i)-1$ are of different parity, so either $f(k) > f(i+1) \geq x(i) + x(i+1)$ or $f(k) > f(r(i)-1) \geq x(r(i)) + x(r(i)-1)$. □

Decreasing $f(i)$ by Δ corresponds to the following change in the scheme of $(i, r(i))$ *below* $\Gamma : \alpha$ increases by Δ, and equivalently, the number of white-white conflicts increases by Δ, the number of black-white levels and white-black levels both decrease by Δ.

We end the section with the description of two procedures. The input of f-TIME is (i, j), $k \in \langle i, j \rangle$, the partial instance $x(i), x(i+1), \ldots, x(j)$, and α. The procedure computes the finish time $f_\alpha(k)$, *given that* i is a non-compact loc-min, $j = r(i)$, $k = pit(i, j)$, and (i, j) have α black-black conflicts below Γ. In this pseudo-code we assume w.l.o.g., that k and i are even. All other cases are analogous. We use the short notation $f(u)$ for $x(u-1) + x(u)$ if u is supposed to be a stair-up, resp. $f(v)$ for $x(v+1) + x(v)$ if v must be a stair-down. Here we do not check whether $k = pit(i, j)$ is indeed feasible.

Procedure f-TIME

```
u := i + 1;   v := j − 1;                                   (current stairs)
Γ := min(f(u), f(v));
β := Γ − (x(j) − α);          (the number of black levels of k, below min(f(u), f(v)))
while β < x(k) do
  if f(u) ≤ f(v) then
    u := u + 1;
    if f(u) ≤ f(u − 1) then
      output f_α(k) = ∞.                                    (u is not a stair-up)
    end if
  else
    v := v − 1;
    if f(v) ≤ f(v + 1) then
      output f_α(k) = ∞.                                    (v is not a stair-down)
    end if
  end if
  if either u or v is even then
    β := β + (min(f(u), f(v)) − max(f(u − 1), f(v + 1)));
  end if
end while                                                   (now β ≥ x(k))
```

$f_\alpha(k) := \min(f(u), f(v)) - (\beta - x(k))$;
output $f_\alpha(k)$.

Similarly, $\widehat{f}$-TIME computes the finish time $\widehat{f}_\alpha(m)$, given that m is a unique loc-max in $\langle i, j\rangle$. We assume w.l.o.g. that i and m are even.

Procedure $\widehat{f}$-TIME

$u := i + 1$; $\quad v := j - 1$;
$\Gamma := \min(f(u), f(v))$;
$\beta := x(i) - \alpha$; (the number of black levels of m, below $\min(f(u), f(v))$)
$f(m) := \infty$;
while $\beta < x(m)$ and ($u < m$ or $m < v$) **do**
  if $f(u) \le f(v)$ **then**
    $u := u + 1$;
    if $f(u) \le f(u-1)$ **then**
      output $\widehat{f}_\alpha(m) = \infty$. ($u$ is not a stair-up)
    end if
  else
    $v := v - 1$;
    if $f(v) \le f(v+1)$ **then**
      output $\widehat{f}_\alpha(m) = \infty$. ($v$ is not a stair-down)
    end if
  end if
  if $u = m = v$ **then**
    $\widehat{f}_\alpha(m) := \max(f(m-1), f(m+1)) + (x(m) - \beta)$;
    output $\widehat{f}_\alpha(m)$.
  else
    if both u and v are even **then**
      $\beta := \beta + (\min(f(u), f(v)) - \max(f(u-1), f(v+1)))$;
    end if
    if $\beta \ge x(m)$ **then**
      output $\widehat{f}_\alpha(m) = \infty$. ($m = top(i, j)$ is impossible)
    end if
  end if
end while

The correctness of the procedures follows from (P1)–(P4). The running time is proportional to the number of stairs having finish time below $f_\alpha(k)$; for $\widehat{f}$-TIME it is $\mathcal{O}(j - i)$.

2.3.2 Minimum-tree

In order to put all this in a more global perspective, we associate a binary tree with each block in a solution (see Fig. 1.3):

Definition 2.21 *If $\langle g, h\rangle$ is a block then the (non-compact) loc-mins inside form a binary tree in the following way: $i := pit(g, h)$ is the root of the tree, and its two*

children are $pit(\ell(i), i)$ *and* $pit(i, r(i))$*, etc. The loc-max's can be regarded as the leaves of this tree. We will call this tree the* **minimum-tree** *of* (g, h)*. Since the term 'level' means time unit, we will say that the root* i *is on the* **1st floor***, the children of* i *are on the* **2nd floor***, etc. of the minimum-tree.*

Since blocks are easy to connect to each other, till the very end we will stay within one block. Eventually, blocks can be connected by a straightforward dynamic programming algorithm.

In Phase 1 the algorithm proceeds from leaves to root and composes the minimum-tree of a (potential) block. In Phase 2, the minimum-tree is given, and we do the scheduling by proceeding from root to leaves. Having the minimum-tree fixed, the root i is the only loc-min that can have (black) levels conflicting with both $\ell(i)$ and $r(i)$, and so $f(i)$ is not automatically implied. See for example level 2 of loc-min 7 on Fig. 2.1. But if the optimal $f(i)$ is fixed – i.e., it is also computed in Phase 1 –, it essentially determines $\Phi(i)$, whereas the latter determines the schedule of all the other loc-mins – by (P2) and (P4) –, and the schedule of the whole block – by (P1) (see Fig. 2.3 (ii)).

Observe that inside a block $\langle g, h\rangle$, or inside any $\langle i, r(i)\rangle$, there are one more loc-max's than loc-mins. The uniqueness theorem will imply that increasing the finish time of the root in a minimum-tree by Δ increases the finish time of every other loc-min and decreases the finish time of every loc-max by the same value, resulting in an improvement of Δ on the whole block. However, each increase must mean exchanging a black level below the finish time for a white level above the finish time of a loc-min, both having conflicts on both sides. This condition bounds from above the potential values of Δ in a recursive manner.

2.4 Basic operations

We can streamline the technical proof of Theorem 2.3 by introducing the basic operations (O1) – (O5) in advance. First of all, we need further definitions:

Definition 2.22 *Let* s *be a stair-up (stair-down) in a fixed solution* Φ*. The* **rung of** s *is the set of levels* $S := [f(s-1)+1, f(s)]$*, (resp.* $S := [f(s+1)+1, f(s)]$*). For simplicity we use the respective capital letters to denote rungs. We will also use the short notations:* $\underline{S} := \min S$*, and* $\overline{S} := \max S = f(s)$*.*

Let m *be a loc-max, and* $f(m-1) < f(m+1)$*. The* **rung of** m *is the set of black levels above* $f(m-1)$*, i.e.,* $M := \Phi(m) \cap [f(m-1)+1, f(m)]$*. If* $f(m-1) > f(m+1)$*, the definition is symmetric.*

The rung of a stair is the topmost set of contiguous black levels of the stair. In a series of stairs the rungs of these stairs partition the levels above the loc-min. We note that rungs of different nodes may refer to the same level-sets. Arbitrary level sets (i.e., not rungs) will also be denoted by capital letters.

Definition 2.22 had to be extended to loc-max's, since $L(i)$ or $R(i)$ might be a loc-max. In order to simplify the discussion, we will regard a loc-max to be a 'last

stair' on the respective side (if, e.g., in the discussion m is playing the role of $L(i)$, then $f(m-1) < f(m+1)$, and we regard m as a stair-up). For an example of such a 'last stair-up', see node $8 = L(10)$ on Fig. 2.1. Due to (P2), if i is a non-compact loc-min, and, e.g., $L(i) = m$ is a loc-max, then $(m-1, m)$ is not white-white under the level $f(i)$ (so the rung M is contiguous under $f(i)$). When we talk about rungs of stairs, we usually mean such 'last stairs', i.e., loc-max's as well. Since below $f(i)$ the schedule of $L(i)$ behaves like a stair, in most cases this won't cause difficulties, and the statements remain true. We deal with the problematic cases explicitly. In the next definition s is a stair in this broad sense.

Definition 2.23 *We say that s is a* **second stair** *resp. S is a* **second rung** *if $s-1$ or $s+1$ is a loc-min. We call a stair or rung* **high stair**, *resp.* **high rung** *otherwise.*

Let i be a non-compact loc-min, $c = L(i)$ and $d = R(i)$. Now C and D denote the rungs of c and d, respectively, and $f(i) \in C \cap D$. We say that i **(or $f(i)$) ends in C on the left and in D on the right.**

In the rest of the section we fix the node k, and some sets of levels C and D below $f(k)$, where k is black and white, respectively. Let $|C| \leq |D|$. We will exchange the levels of C for some levels in D on a subpath containing k. The five basic exchange operations below restrict the subpath to the left of k (see Figure 2.5). We can use symmetric versions of (O1)–(O5) for the right side.

In our proofs we will frequently use combinations of these exchange operations (one type on the left and one type on the right side), in order to exclude the optimality of some solution.

(O1) Let k be a stair-down or a loc-min, let $i < k$ and every node in $\langle i+1, k\rangle$ have finish time above $C \cup D$. Let either $B \subset C$ or $B \subset D$, s.t. i and k have a conflict in the levels of B. Due to Corollary 2.16, there is a node $i \leq m < k$ s.t. $(m, m+1)$ is white-white on these levels and m or $m+1$ is a loc-max. Now we can exchange the levels of C for levels of D on $\langle m+1, k\rangle$, so that it costs $(|C| - |B|)$ on the left. In general, exchanging any two level sets C' and D' below $f(k)$, costs at most $|C'|$ on the left.

(O2) Let $k = c$ be a stair-down, $c = R(i)$, and C denote the rung of c. If $f(i+1) \geq f(c)$, then we can exchange the level set C for some levels of D for free on the left: Let i be black on $B \subset D$ and white on $A \subset D$. Since c and i have the same parity, they have a conflict on the levels of B and also on $[f(i)+1, f(c)]$. Exchanging any of these levels (with any other level) is for free. The remaining levels to exchange should be the topmost levels of i and some levels of A. These we exchange on $\langle m, c\rangle$, where m is the first loc-max to the left of i. Now we increased $f(m)$ but decreased $f(i)$ by the same amount. The same holds for any set of topmost black levels of k instead of the whole rung C.

(O3) Let $k = c$ be a high stair-up (or loc-max), and C the rung of c. Making the levels C white and some levels in D black in c, costs $|C|$ on the left since we have to increase $f(c-1)$ by $|C|$. Similarly, exchanging any other level set $C' \supseteq C$ of k costs at most $|C'|$ on the left.

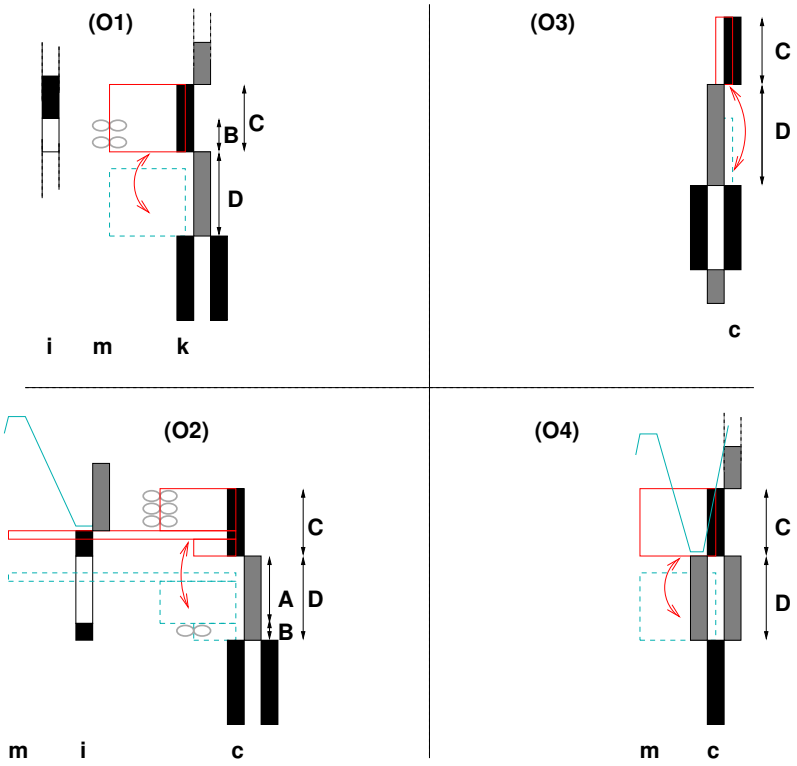

Figure 2.5: Operations of type (O1) – (O4).

(O4) Let $k = c$ be a second stair-up (or loc-max), C the rung of c. Now $c-1$ is a loc-min, and let us also assume that $f(c-2) > f(c)$ This operation is a combination of (O3) and (O1): We exchange C with levels of D. Let m be the first loc-max to the left of $c-1$. The exchange costs $|C|$ at $c-1$, because we increase $f(c-1)$ (see (O3)), and it costs at most $|C|$ on the left – at m –, because we exchanged levels of the loc-min $c-1$ (see (O1)). Exchanging any level set $C' \supseteq C$ of k costs at most $|C'| + |C|$ on the left.

(O5) Let $k = c$ be a second stair-up and suppose $f(c-2) \leq f(c)$. Now we only pay $|C|$ at $c-1$, and the rest is for free, because we decrease $f(c-2)$ as much as we modify it. So, if $f(c-2) \leq f(c)$, then moving the levels of C implies the same costs as in (O3).

Definition 2.24 *We say that we* **place** C **into** D *if we exchange the levels of* C *for some levels in* D *on a subpath that contains* k*. By this we mean exchanging on the subpath specified in* (O1) – (O5)*, if (on a side) the respective setting corresponds to one of these operations.*

Corollary 2.25 *Suppose that we place some levels* C' *of node* k *(including the rung of* k*) into some levels* D*. If* k *is not a second stair, the operation costs at most* $2|C'|$*. If* k *is a second stair on one side, it costs at most* $3|C'|$*. If* k *is a second stair on both sides, then we have the trivial setting that both* $k-1$ *and* $k+1$ *are compact.*

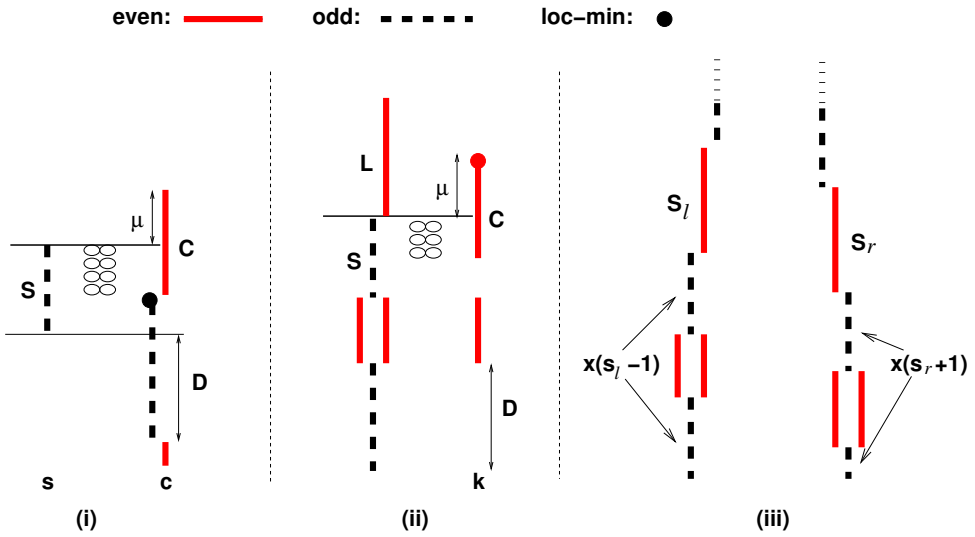

Figure 2.6: (i) Proposition 2.26, (ii) Proposition 2.27, and (iii) Proposition 2.28. The contiguous level sets are drawn with solid line segments in the schedule of an even node, resp. with broken line segments for odd nodes. The finish time of a loc-min node is marked by a dot.

Proof. If k is a loc-min, then we apply (O1) on both sides; if it is a high stair, we apply (O1) on one side and (O3) on the other side; if k is a (high) loc-max, we apply (O3) on both sides. If k is a second stair (or loc-max), then we apply (O4) on one side. Given that C' includes the rung of k (in case k is a stair or loc-max), we obtain the costs stated in the corollary.

If both neighbors of k are loc-mins, then the neighbor of higher demand is compact by Corollary 2.10, and the other neighbor is compact by Corollary 2.11. □

We finish the section with three simple applications of (O1) – (O5) (see Fig. 2.6).

Proposition 2.26 *Let C be a second rung of a stair-up c and the loc-min $c-1$ end in some rung S on the left. Let D denote the set of white levels of c below $\underline{S}$. If placing C into D is for free on the right, and $|C| \le |D|$, then $|S| \le |C|$.*

Proposition 2.27 *Let k be an even loc-min, and $f(k)$ end in L on the left, where L is a high even rung and S is some odd rung below L. Let C be the set of black levels of k above $\underline{S}$, and D be the set of white levels of k below $\underline{S}$. If $|C| \le |D|$, then $|S| \le |C|$.*

Proof of Propositions 2.26 and 2.27. Placing C into D costs at most $|C|$ at $c-1$ in Proposition 2.26 by (O4)-(O5), resp. at most $|C|$ on the right in Proposition 2.27 by (O1). Moreover, if C has μ levels above $\overline{S}$, then placing C down costs μ on the left by (O1).

The costs of placing down amount to at most $|C| + \mu$ in both cases, whereas the finish time $f(k)$ would decrease by at least $|S| + \mu$. Thus $|S| + \mu \le |C| + \mu$. □

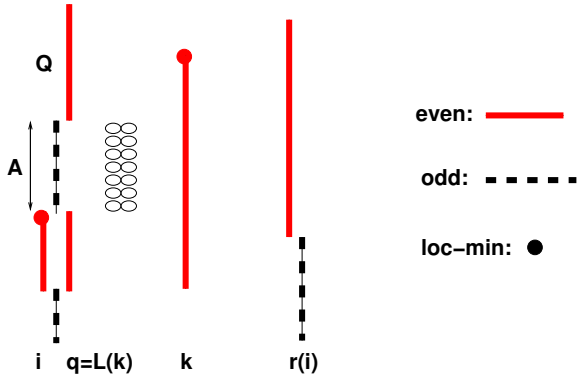

Figure 2.7: Lemma 2.29.

Proposition 2.28 *Let s_ℓ be a high stair-up and s_r be a high stair-down, s.t. $s_\ell < s_r$ and they are both even. Furthermore, the rungs of $s_\ell - 1$ and $s_r + 1$, resp. the rungs of $s_\ell + 1$ and $s_r - 1$ intersect, and $f(k) \geq \max(\overline{S}_\ell, \overline{S}_r)$ for all $s_\ell < k < s_r$. Then either $|S_\ell| \geq x(s_\ell - 1)$ or $|S_r| \geq x(s_r + 1)$.*

Proof. Suppose that $|S_\ell| < x(s_\ell - 1)$ and $|S_r| < x(s_r + 1)$. Now both S_ℓ and S_r fit into the white levels below them, and we place down both S_ℓ and S_r. This costs $|S_\ell|$ and $|S_r|$ on the two sides by (O3), but costs 0 between s_ℓ and s_r : obviously, it is for free on the conflicting levels $S_\ell \setminus S_r$ and $S_r \setminus S_\ell$ by (O1). It is also for free on the common levels $S_\ell \cap S_r$, because either we exchange the complete level on $\langle s_\ell - 1, s_r + 1\rangle$, or we place the two sides *into* conflicting levels. □

2.5 The size of the 'lowest rungs'

Suppose that $k = pit(i, r(i))$ for some pair $(i, r(i))$. This section includes two technical lemmas, which provide lower bounds on the sizes of rungs or demands of $L(k)$ and $R(k)$. The loc-mins in a minimum-tree end in different pairs of rungs. Clearly, the larger these rungs are, the larger the difference between the demands inside the block must be.

Lemma 2.29 *Let Φ be a solution on $\langle 1, n\rangle$ and i a non-compact loc-min. Suppose that $k = pit(i, r(i))$ and q is a high stair, s.t. $q \leq L(k)$. If A is the set of levels below $\underline{Q}$, where q is white and conflicting with $R(k)$, then $|Q| \geq |A|$. A symmetric statement holds for a high stair s, when $R(k) \leq s$.*

Proof. If there are loc-mins ending in Q, then let $k' \leq k$ be the closest loc-min to q. Otherwise let $k' = k$. Now by (P2) the pair (q, k') has (white-black) conflicts on the levels of A. Suppose that $|Q| < |A|$ holds. We apply (O3) on the left and (O1) on the right to place Q into A. This costs $|Q|$ on the left and 0 on the right, but $f(q)$ decreases by more than $|Q|$, so we got a better optimum sum, a contradiction. □

Unfortunately, the above lemma does not hold if q (or s) is a second stair. The best we can say about second stairs is stated in Lemma 2.31. The lemma is not

applicable on the first floor of a minimum-tree, but this won't influence essentially our running time bounds. Case $(**)$ will cause most of the difficulties in the proof of Theorem 2.3.

Observe, that for $k = pit(i, r(i))$ only at most one of $L(k)$ or $R(k)$ is a second stair, since $L(k)$ and $R(k)$ are of the same parity, and i and $r(i)$ are of different parity.

Recall that $\Gamma = \min(x(i) + x(i+1), x(r(i)) + x(r(i)-1))$.

Proposition 2.30 *If i is a non-compact loc-min in a solution, and A is the set of levels where $(i, r(i))$ has black-black conflicts below Γ, then $x(i+1) > |A|/2$ and $x(r(i)-1) > |A|/2$.*

Proof. Let $d = i + 1$. Suppose that $x(d) \leq |A|/2$. Now we place all the black levels of d into the bottom levels of A. This is for free on the right, even if some loc-mins end in d, because in this case the $|A|$ conflicts are between d and the loc-mins. It costs at most $|D| + x(d)$ on the left, but $f(d)$ is decreased by at least $|D| + |A|/2$. If $x(d) = |A|/2$ and the sum of finish times did not decrease, then the sum of squares of finish times increased: $f(i)$ increased to $f(d)$; $f(d)$ decreased below $f(i)$, and $f(m)$ increased for some loc-max m above $f(d)$. Thus, we got a better solution, a contradiction. The proof of $x(r(i)-1) \geq |A|/2$ is symmetric. □

Lemma 2.31 *Let Φ be a solution on $\langle 1, n\rangle$ and i a non-compact loc-min. Let A be the set of levels where $(i, r(i))$ have black-black conflicts below Γ. Furthermore, let $e = R(i) = r(i) - 1$, and E be the rung of e. Then at least one of the following holds:*
$(*)$ *$x(i+1) > |A|$;*
$(**)$ *E is a high rung and $|E| > |A|$;*
$(***)$ *$r(i)$ is compact and i is the root of a minimum-tree.*

Proof. Let i be odd. We start from $f(i)$ to the right, and make a zigzag walk downwards (see Figure 2.8). Let $s_1 = R(i)$, i.e. i ends in S_1 on the right. If S_1 is a second rung, then let $m_1 = s_1 + 1$ and if m_1 is non-compact, let $s_2 = L(m_1)$. Note that $s_2 < i$. If S_2 is a second rung, then let $m_2 = s_2 - 1$ and if m_2 is non-compact, then let $s_3 = R(m_2) > s_1$. We terminate the zigzag walk if a loc-min m_{t-1} ends in S_t and either

(1) S_t is a high rung, or
(2) S_t is a second rung above a compact loc-min m_t.

Let $m_0 = i$ and S_0 be the rung of $i + 1$. Now we have a series of second rungs (except for the last): $S_0, S_1, ..., S_t$ and the following hold:

$\{S_{2\tau}\}$ are disjoint, and they are rungs of even nodes to the left of $\langle i, r(i)\rangle$, whereas $\{S_{2\tau+1}\}$ are also disjoint, and they are rungs of odd nodes to the right of $\langle i, r(i)\rangle$. Moreover, $[f(m_{2\tau})+1, f(m_{2\tau-1})]$ are clear even level-sets and $[f(m_{2\tau+1})+1, f(m_{2\tau})]$ are clear odd level-sets on a subpath enclosing $\langle i, r(i)\rangle$. This gives a partition of the levels between $\underline{S_t}$ and $f(i)$. The conflicting levels A are below $f(i)$, consequently they are all below $\underline{S_t}$. Moreover, the levels of A are clear even to the right of $r(i)$, and clear odd to the left of i. These statements are easy to verify by (P1) – (P4).

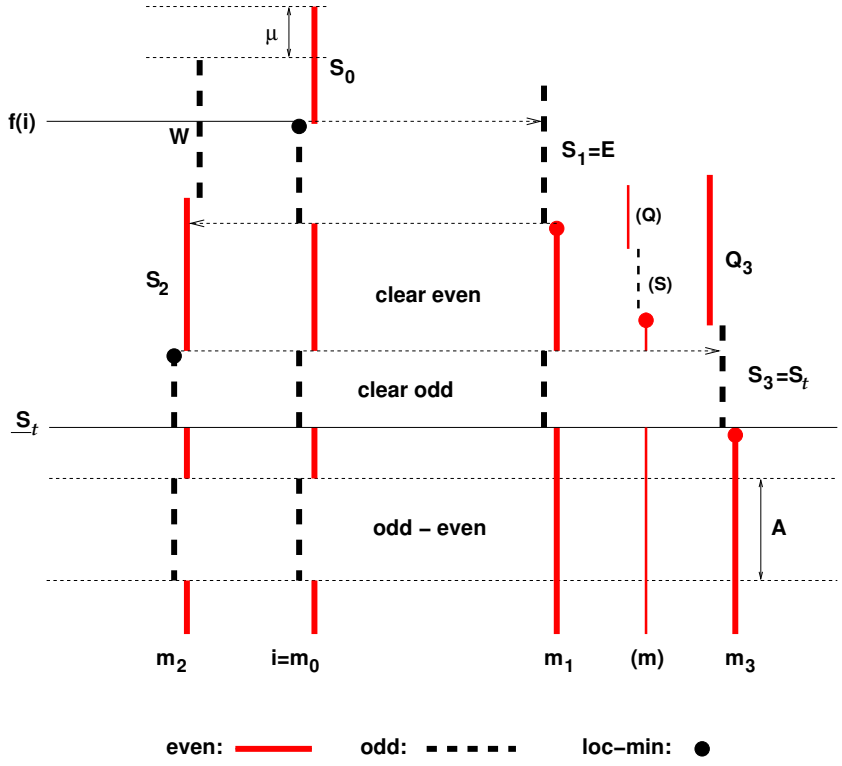

Figure 2.8: Illustration to Lemma 2.31.

We will call m an *inserted loc-min* if it is between m_τ and $s_{\tau+2}$ for some τ, and $f(m_\tau) > f(m) > f(m_{\tau+1})$. Rungs of neighbors of inserted loc-mins will be called *inserted rungs.*

We claim that the sizes of the rungs on each side are decreasing, or they have size larger than $|A|$:

Claim 2.32 *Suppose that $|S_\tau| \le |A|$. In this case $|S_\tau| > |S_{\tau+2}|$, and if there is an inserted rung S between m_τ and $s_{\tau+2}$ then $|S_\tau| > |S| > |S_{\tau+2}|$.*

Proof. Suppose w.l.o.g. that τ is odd, so we are to the right of $\langle i, r(i)\rangle$.

First we consider the case, when there is no inserted loc-min in $\langle m_\tau, s_{\tau+2}\rangle$. Now m_τ ends in some even rung $Q_{\tau+2}$ above $S_{\tau+2}$. We assumed $|S_\tau| \le |A|$. Now we can place S_τ into A for free on the left by (O1), and so according to Proposition 2.26, $|S_\tau| \ge |Q_{\tau+2}|$. This, in turn, implies $\overline{S_\tau} > \overline{Q_{\tau+2}}$, that is, if $|Q_{\tau+2}| \le |S_{\tau+2}|$, then applying (O2) on the left and (O3) on the right, we could place $Q_{\tau+2}$ into $S_{\tau+2}$ and obtain a better optimum. (If $|Q_{\tau+2}| = |S_{\tau+2}|$, we obtain larger sum of squares of finish times since $f(m_\tau)$ decreases and a loc-max increases.) Thus, $|S_\tau| \ge |Q_{\tau+2}| > |S_{\tau+2}|$.

Second, let m be a single inserted loc-min between m_τ and $s_{\tau+2}$. Let S be the (odd) inserted rung of $m-1$, and Q be the (even) rung above S, where m_τ ends. Now we can prove the same way that $|S| > |S_{\tau+2}|$, respectively that $|S_\tau| > |S|$. The argument carries over to any number of consecutive inserted loc-mins. □ *Claim 2.32*

Claim 2.33 *In case* (1), $|S_t| > |A|$. *In case* (2), *for all* $\tau \neq t-1 \quad |S_\tau| > |A|/2$, *and if* S *is an inserted rung, then* $|S| > |A|/2$.

Proof. Suppose w.l.o.g. that t is odd, so s_t is to the right of $\langle i, r(i)\rangle$.

In case (1), S_t is a high rung. Suppose that $|S_t| \leq |A|$. We place S_t into A : it is for free on the left by (O1), and costs $|S_t|$ on the right by (O3). If $f(s_t)$ decreased by more than $|S_t|$, we obtained a smaller sum of finish times; if $f(s_t)$ decreased by exactly $|S_t|$, now we can also reduce $f(m_{t-1})$ for free on the right, and so increase the sum of squares of finish times, a contradiction.

In case (2), S_t is a rung above a compact loc-min m_t. First we show $|S_t| > |A|/2$. Note that $x(m_t) > |A|$, since m_t has $|A|$ conflicting levels and at least one non-conflicting level, because m_{t-1} is non-compact. Suppose that $|S_t| \leq |A|/2$. Now we place S_t to the bottom levels – make s_t compact –, applying (O2) on the left. The operation costs at most $2|S_t|$ on the right by (O4), and we gain at least $|S_t| + |A|/2 + 1 > 2|S_t|$, a contradiction.

Second, we prove $|S_{t-3}| > |A|/2$. We claim that $x(s_{t-1}) > |A|/2$, by Proposition 2.30.

If there are no inserted rungs between s_{t-1} and m_{t-3}, then m_{t-3} ends in some odd rung U above S_{t-1}. If $\overline{U} \geq \overline{S_{t-3}}$, then $|S_{t-3}| \geq x(m_{t-3}) > |A|$, otherwise we could place S_{t-3} down. If $\overline{U} < \overline{S_{t-3}}$, then $|U| \geq x(s_{t-1}) > |A|/2$, otherwise we could place down U by (O2) on the right and (O3) on the left. Furthermore, Proposition 2.26 implies $|S_{t-3}| \geq |U|$.

Finally, if S is a (leftmost) inserted rung between s_{t-1} and m_{t-3}, the same proof yields $|S| > |A|/2$.

Now Claim 2.32 implies the claim for all other S_τ or inserted rungs. □ *Claim 2.33*

Let $f(i)$ end in W on the left. Notice that unless $t = 1$, W is a high odd rung above S_2 or above an inserted rung S between s_2 and i.

Claim 2.34 *If* $\overline{W} > \underline{S_t} + |A|$, *then* $x(i+1) > |A|$.

Proof. Suppose that $x(i+1) \leq |A|$. We claim that $\overline{S_0} > \overline{W}$: otherwise the complete S_0 would have conflicts on the left side, A has conflicts on the right, so unless $|A| < |S_0| < x(i+1)$, it would be worth placing S_0 into A using (O1) and (O4). Let $\overline{S_0} = \overline{W} + \mu$.

Let's place all the $x(i+1)$ black levels of $i+1$ into A. This costs 0 on the right. It costs $|S_0|$ due to increasing $f(i)$ and costs at most $x(i+1) - |S_0| + \mu \leq |A| - |S_0| + \mu$ on the left, by (O1). The total cost is at most $|A| + \mu$, but we decreased $f(i+1)$ by more than $\mu + |A|$, since there are more than $|A|$ levels between $\underline{S_t}$ and $\overline{W}$. So, the solution was not optimal. □ *Claim 2.34*

Claim 2.35 *In case* (1), *either* $(*)$ *or* $(**)$ *holds.*

Proof. In this case we have $|S_t| \geq |A|$. If $\overline{S_t} < f(i)$ then $(*)$ holds by Claim 2.34.

The only case when $\overline{S_t} \geq f(i)$ might occur, is when $t = 1$, meaning that $s_1 = e$ is a high stair. Now $|E| > |A|$ follows from Claim 2.33. □ *Claim 2.35*

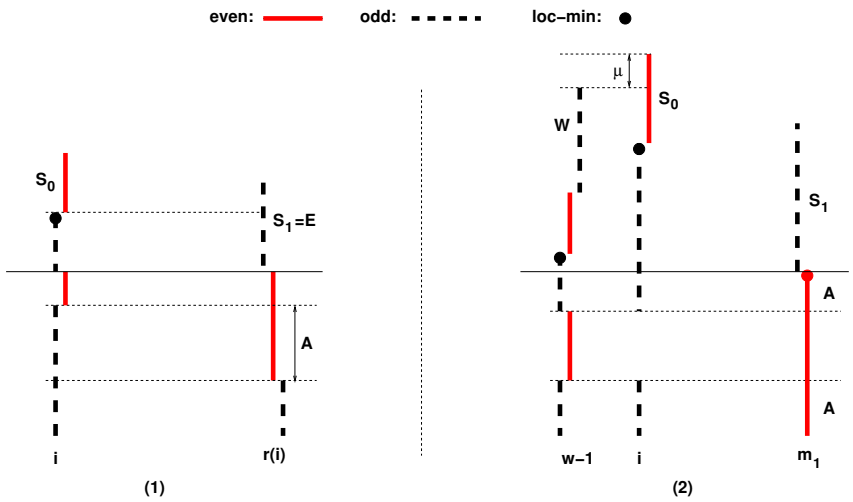

Figure 2.9: Illustrations to Claim 2.35 (1), and Claim 2.36 (2).

Claim 2.36 *In case* (2)*, either* $(*)$ *or* $(***)$ *holds.*

Proof. We claim that if W is a high rung, then $|W| \geq x(w-1)$. We saw in Claim 2.34 that the case $\overline{S_0} \leq \overline{W}$ can be excluded, so we assume $\overline{S_0} > \overline{W}$. If $|W| < x(w-1)$, then we could place W into the black levels of $w-1$ for free on the right by (O2), for $|W|$ on the left by (O3), and reducing $f(w)$ by more than $|W|$. Moreover, $x(w-1) > |A|/2$, since otherwise it would be worth making $w-1$ compact: it is for free on the right, it costs at most $2x(w-1)$ on the left, and $f(w-1)$ reduces by more than $|A|$. So, if W is a high rung then $|W| > |A|/2$.

If $t \geq 2$, then W is an odd rung above S_2 or above an inserted rung S. So we have $|W| > |A|/2$. Since $t \neq 1$, W and S_t are disjoint, and $S_t > |A|/2$ by Claim 2.33. So we have $\overline{W} \geq \underline{S_t} + |S_t| + |W| > \underline{S_t} + |A|$, and Claim 2.34 implies $(*)$.

Now let $t = 1$, implying that $m_1 = r(i)$ is a compact loc-min (see Figure 2.9 (2)). If the loc-min below W ends below $f(m_1)$ then it is compact by Corollary 2.11, and i is the root of a minimum-tree, so that $(***)$ holds. If the loc-min below W is non-compact, and it ends in S_1, then it has the same parity as w, so W is a high rung. Thus, $|W| > |A|/2$, and since $x(w-1) > |A|/2$, we obtain $\overline{W} > |A| + |A|/2 + |A|/2 = 2|A|$. Suppose that $(*)$ does not hold, that is $x(i+1) \leq |A|$. Now we make $i+1$ compact. This operation is for free on the right since m_1 is compact. If $\overline{S_0} = \overline{W} + \mu$, then it costs at most $x(i+1) + \mu$ on the left, but $f(i+1)$ decreases by more than $|A| + \mu$, and we obtained a better solution, a contradiction. □ *Claim 2.36*
□ *Lemma 2.31*

2.6 The uniqueness theorem

In this section we rely on the notation and definitions introduced in Section 2.3.1. Recall that there we fixed a pair of nodes $(i, r(i))$ and the partial instance $\mathcal{I}$ between these nodes. We defined the level Γ and the number α of conflicting black-black levels of $(i, r(i))$ as well as the solutions Φ_α.

Our major goal is to show that for all α, the solution Φ_α has the same minimum-tree (there might be different equivalent solutions if there is one loc-max in $\langle i, r(i)\rangle$).

In order to facilitate a global view on the main results, we collected the long and technical proofs in the next section. The results themselves and the algorithm can be understood even if one omits Section 2.7 completely.

Theorem 2.1 serves as an illustration of what we would like to show in general, and also as the base step for an induction proof of the general uniqueness theorem. In this basic case we will assume that for some a and some Δ, in both Φ_a and $\Phi_{a-\Delta}$ there is just one loc-max between i and $r(i)$. Recall that $\mathcal{F}(\alpha) = \sum_{k=i+1}^{r(i)-1} f_{\Phi_\alpha}(k)$ denotes the optimum sum of finish times for given α on $\langle i+1, r(i)-1\rangle$.

The proof of Theorem 2.1 uses the following proposition. Note that the proposition assumes a more general setting than the theorem, e.g., in that the roles of i and j are symmetric. Later, in the uniqueness theorem we will apply a similar assumption.

Proposition 2.37 *Let $i < j$ be nodes of different parity, and Φ'_a and $\Phi'_{a-\Delta}$ two solutions, such that in both of them at least one of i or j is a non-compact loc-min. Let a, resp. $a-\Delta$ denote the number of black-black conflicts of (i,j) in the two solutions. Suppose that a level Π exists s.t. in both solutions $\max(f(i), f(j)) < \Pi \leq \min\{f(k) | k \in \langle i+1, j-1\rangle\}$. If $\mathcal{F}(\alpha) = \sum_{k=i+1}^{j-1} f_{\Phi'_\alpha}(k)$, then $\mathcal{F}(a) - \Delta \leq \mathcal{F}(a-\Delta) \leq \mathcal{F}(a)$.*

Proof. First we prove $\mathcal{F}(a-\Delta) \leq \mathcal{F}(a)$. Consider the solution Φ'_a restricted to $\langle i, j\rangle$, and take arbitrary $\Delta \leq a$ levels where i and j have black-black conflicts. Since each conflicting level has white-white neighbors inside $\langle i, j\rangle$, these Δ levels can be exchanged for Δ levels where (i,j) have white-white conflicts, so that the schedule remains valid. Now the new schedule has $a-\Delta$ black-black conflicts, and we can permute the levels below Π according to the schedule $\Phi'_{a-\Delta}$, and still have $\mathcal{F}(a)$ as sum of finish times. Since $\Phi'_{a-\Delta}$ is (optimal) solution, $\mathcal{F}(a-\Delta) \leq \mathcal{F}(a)$ must hold.

Now we prove $\mathcal{F}(a) - \Delta \leq \mathcal{F}(a-\Delta)$. Assume that i is a loc-min in the solution $\Phi'_{a-\Delta}$, and consider this solution restricted to $\langle i, j\rangle$. If m is the closest loc-max to the right of i, we exchange arbitrary Δ non-conflicting black-white levels for Δ white-black levels on $\langle i, m\rangle$. We make the schedule valid, by increasing $f(m)$ by at most Δ, and so we obtain a schedule with a conflicts, and an optimum sum of at most $\mathcal{F}(a-\Delta) + \Delta$. We can permute the levels as they are in Φ'_a. Since Φ'_a was optimal, we have $\mathcal{F}(a) \leq \mathcal{F}(a-\Delta) + \Delta$. □

Theorem 2.1 *Let i, $r(i)$, Φ_α, etc. be as defined above. If $a, a-\Delta \in \mathcal{D}_\mathcal{F}$, and $pit(i, r(i)) = \emptyset$ in both Φ_a, and in $\Phi_{a-\Delta}$, then*
(I) *we may assume – modulo equivalent solutions – that $top(i, r(i))$ is the same node in both Φ_a and $\Phi_{a-\Delta}$, moreover*
(II) $\mathcal{F}(a-\Delta) = \mathcal{F}(a) - \Delta$.

Proof. In Φ_a, let m be the only loc-max, and let q denote the size of the topmost contiguous set of black levels of m. We claim that $q > \Delta$: By Corollary 2.10, either

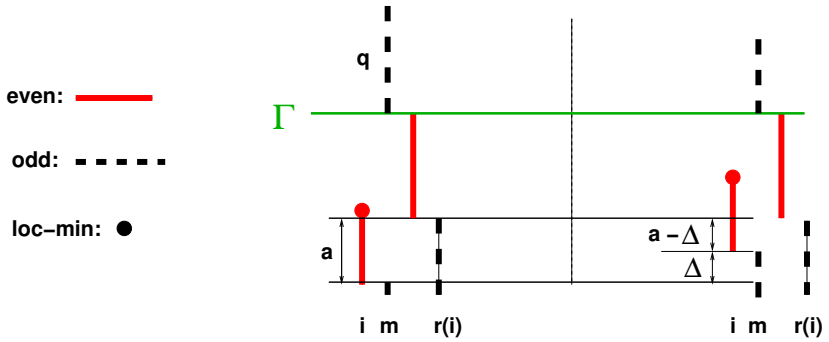

Figure 2.10: Illustration to Theorem 2.1.

$m-1 \neq i$ or $m+1 \neq r(i)$. We assume w.l.o.g. that $m+1 \neq r(i)$. Consequently, $m+1$ is not a loc-min, and $f(m+1) \geq \Gamma$. Note that $(i, r(i))$ has at least Δ black-black and at least Δ white-white conflicts strictly below Γ, otherwise the number of conflicting levels could not decrease to $a - \Delta$ while $f(i) < \Gamma$.

Suppose that $q \leq \Delta$, then – depending on the parity of m – we could place the q black levels of m below Γ, into the black-black or to the white-white conflicting levels for free on one side, and increasing $f(m+1)$ by q on the other side. Since m was white on the level $f(m+1) \geq \Gamma$, we decreased $f(m)$ by at least $q+1$, so we got a smaller sum of finish times, a contradiction.

Due to Proposition 2.37, $\mathcal{F}(a) - \Delta \leq \mathcal{F}(a - \Delta)$. Using $q > \Delta$, now we show $\mathcal{F}(a - \Delta) \leq \mathcal{F}(a) - \Delta$. If the number of conflicts in Φ_a decreases by Δ, we gain Δ black-white and Δ white-black levels of $(i, r(i))$, and can put the top Δ black levels of m to exactly one of these level-types (depending on parity), obtaining a sum of finish times at most $\mathcal{F}(a) - \Delta \leq \mathcal{F}(a - \Delta)$. Since $\mathcal{F}(a - \Delta)$ is the optimum, our new sum of finish times must equal $\mathcal{F}(a) - \Delta = \mathcal{F}(a - \Delta)$, and the new finish time of m must equal $f_{\Phi_a}(m) - \Delta$. So, $m = top(i, r(i))$ still holds, furthermore we can take this solution to be $\Phi_{a-\Delta}$. □

Figure 2.11 sketches two equivalent solutions with different $top(i, j)$ nodes.

Next, we formulate a **degenerate** variant of Theorem 2.1, in which i and $i+1$ change roles, and the optimum sum is considered only on $\langle i+2, j-1 \rangle$:

Theorem 2.2 *Assume that Φ'_a and $\Phi'_{a-\Delta}$ are two solutions on $\langle 1, n \rangle$, such that in both of them j is a loc-min or a stair-down; in one of them i is a loc-min and in the other $i+1$ is a loc-min and $f(i) = x(i) + x(i+1)$; furthermore, in both solutions there is exactly one loc-max in $\langle i, j \rangle$, and it is different from $i+1$. Let a, resp. $a - \Delta$ denote the number of conflicting black levels of i, under $x(i) + x(i+1)$ in Φ'_a and $\Phi'_{a-\Delta}$. If $\widetilde{\mathcal{F}}(\alpha) = \sum_{k=i+2}^{j-1} f_{\Phi'_\alpha}(k)$, then $\widetilde{\mathcal{F}}(a - \Delta) = \widetilde{\mathcal{F}}(a) - \Delta$.*

Proof. An analogous degenerate variant of Proposition 2.37 holds, and the proof of the theorem is based on this. The proof is basically the same as that of Theorem 2.1, the only difference is that if in Φ'_a the loc-max node is $j-1 = i+2$, then only $q \geq \Delta$ holds, instead of $q > \Delta$. However, even in this case, $j - 1 = i + 2$ is the loc-max node in both Φ'_a and $\Phi'_{a-\Delta}$, and using this, $\widetilde{\mathcal{F}}(a - \Delta) = \widetilde{\mathcal{F}}(a) - \Delta$ is obvious. □

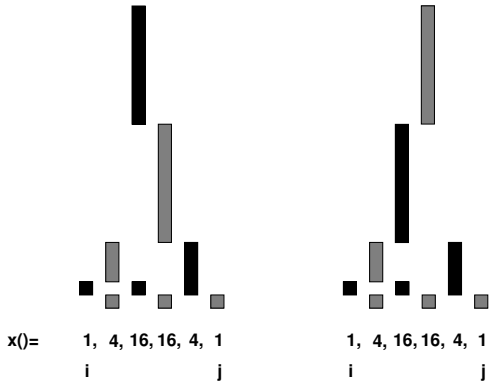

Figure 2.11: Equivalent solutions with different $top(i,j)$ nodes.

In the uniqueness theorem we will generalize Theorem 2.1 for those Φ_α with $pit(i,r(i)) \neq \emptyset$. We want to show that independently of α, the solutions on $\langle i, r(i)\rangle$ have the same minimum-tree. The reason for this is roughly as follows:

Within $\langle i, r(i)\rangle$ the loc-mins end in different (pairs of) rungs, and the sizes of these rungs are large compared to possible differences in $f(i)$. Consequently, the demands of nodes in $\langle i, r(i)\rangle$ differ too much to be alternative candidates for, e.g., $pit(i,r(i))$.

In order to handle the general case, we slightly modify our notation and assumptions: We fix two nodes of different parity $i < j$, and the partial instance $\mathcal{I}$ on the subpath $\langle i, j\rangle$. Suppose that two (partial) solutions, Ψ^* and Ψ exist, s.t. either in both of them $j = r(i)$, or in both of them $i = \ell(j)$; moreover, $pit_{\Psi^*}(i,j)$ has smaller finish time in Ψ^*, than $pit_\Psi(i,j)$ in Ψ. Let $k^* := pit_{\Psi^*}(i,j)$, $d := L_{\Psi^*}(k^*)$ and $c := R_{\Psi^*}(k^*)$ (see Figure 2.12). Clearly, we can assume that $d < k^* < c$ are all even. Moreover, we claim the following:

Proposition 2.38 *We may assume w.l.o.g., that in Ψ^* it holds that $\underline{C} \geq \underline{D}$, and C is a high rung.*

Proof. Note that since the roles of i and j are symmetric, we may swap left and right if $\underline{C} < \underline{D}$.

Now suppose $\underline{C} > \underline{D}$, that is $f(c+1) > f(d-1)$. If $c+1$ were a loc-min, then it would be identical to j, so it would be non-compact. Moreover, $L(c+1) = d$ would hold, but that is impossible, because $c+1$ is odd and d is even. Therefore, C is a high rung.

Finally, if $\underline{C} = \underline{D}$, that is $f(c+1) = f(d-1)$, then due to (P3) $c+1$ or $d-1$ is not a loc-min. Again, we can assume that it is $c+1$, otherwise we swap left and right. □

Notice that we had to impose symmetric roles on i and j, since the previous proposition introduced a new type of asymmetry. In the next paragraphs we introduce Γ', α', and $\Psi_{\alpha'}$, which will be more appropriate for our purposes than Γ, α, and Φ_α :

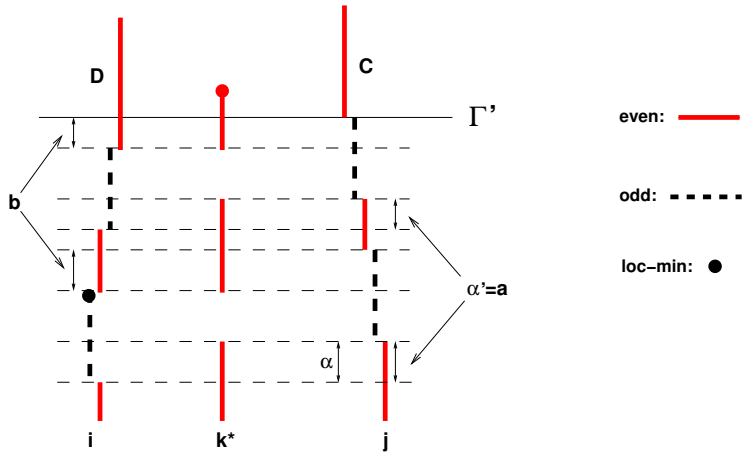

Figure 2.12: The solution Ψ^*; the numbers a and b of conflicting levels in case $\Psi^* = \Psi_a$.

Definition 2.39 *Let $\Gamma' := \underline{C}$, and let α' be the number of levels below Γ', where c is black and has conflicts with d, that is, the number of odd-even levels below Γ'.*

For technical reasons we will calculate with Γ' instead of $\Gamma = \min(x(i)+x(i+1), x(j)+x(j-1))$. Observe that $\Gamma' = \Gamma$, whenever $L(k^*) = i+1$ and $R(k^*) = j-2$, or if $R(k^*) = j-1$ and $L(k^*) = i+2$, which will happen to be the difficult cases.

Depending on the parity of i, the α' odd-even levels include the black-black or the white-white conflicts of (i, j). Clearly, there is a one-to-one correspondence between the old α values and α', (i.e., occurrences of different α in different solutions can be associated with occurrences of different α'), so that we have a solution Φ_α fixed for every occurring α'. Let $\Psi_{\alpha'}$ denote this fixed solution, that is, $\Psi_{\alpha'} = \Phi_\alpha$ whenever α' and α are corresponding values (see Fig. 2.12).

We defined above two different solutions Ψ^* and Ψ on $\langle i, j\rangle$. There exist numbers a and b and $0 < \Delta \le \min(a, b)$ so that the following hold: $\{\Psi^*, \Psi\} = \{\Psi_a, \Psi_{a-\Delta}\}$ (in Figure 2.12 we illustrate the case $\Psi^* = \Psi_a$). In Ψ_a there are a odd-even levels and b even-odd levels below Γ'; In $\Psi_{a-\Delta}$ there are $a-\Delta$ odd-even levels and $b-\Delta$ even-odd levels, and at least Δ clear even and at least Δ clear odd levels below $\Gamma' = \underline{C}$.

In the uniqueness theorem we want to prove that Ψ^* and Ψ are essentially the same solutions. Unfortunately, we could not exclude a few potential exceptions to this rule – although we did not find any example for such an exception. In the lemmas of the previous section we investigated the sizes of lowest rungs in $\langle i, r(i)\rangle$. The proof of Theorem 2.3 is based on the lower bounds on these rung sizes. However, the lower bounds may not hold on the first two floors of a minimum-tree of a block.

We will distinguish the (good) case, when

$$a < \max(x(d),\ \min(3x(d)/2,\ x(c+1),\ x(c) - x(c+2))) \qquad (2.a)$$

holds. It will be more convenient to refer to (2.a) as to the disjunction of properties (R1) and (R2) below.

Proposition 2.40 *Let $\langle i, j\rangle$, Ψ^*, Ψ, the nodes $d < c$, and the number of conflicts a be defined as above, furthermore* (R1) *and* (R2) *be defined as*
(R1) $x(d) > a$;
(R2) $d = i + 1$ *and* $j = r(i)$, *moreover* $a \geq x(d) > 2a/3$, $x(c + 1) > a$ *and* $|C| > a$.
Now (2.a) $\Leftrightarrow$ (R1) $\vee$ (R2).

Proof. Notice, that if $i + 1 < d$, then d is a high stair, and Lemma 2.29 implies $x(d) > |D| \geq a$, so both (R1) and (2.a) hold.

If $i+1 = d$, then (2.a) is a reformulation of ((R1) or (R2)), using the observation that $|C| = x(c) - x(c + 2)$ holds for any stair-down due to (P1). □

Proposition 2.41 *If (2.a) does <u>not</u> hold, then Ψ_a may occur as part of a solution on $\langle 1, n\rangle$ only if in this solution $j = r(i)$ (resp. $i = \ell(j)$), and i (resp. j) is on at most the 2nd floor of a minimum-tree.*

Recall Lemma 2.31, and note that (R1) corresponds to $(*)$, (R2) corresponds to $(**)$, whereas Proposition 2.41 roughly corresponds to $(***)$.

We prove the proposition in the next section. In Theorem 2.3 we will show that if (2.a) holds, then Ψ_a and $\Psi_{a-\Delta}$ have the same minimum-tree, i.e. they are essentially the same. Although we have to deal with different alternative pairs of solutions in the dynamic programming algorithm, fortunately they do not propagate: we can forget all but one solution as soon as we stepped down two floors in the minimum-tree in the dynamic process. Notice that for given (d, c) and a, it is easy to decide whether (2.a) is valid or not, and (2.a) will be a precondition for a solution to occur on higher floors of a minimum-tree.

Finally, similarly to Theorem 2.2, we also consider the case when i and $i + 1$ change roles, i.e. $i + 1$ becomes a loc-min (see Fig. 2.13). We do this by extending the definitions to this degenerate case, whenever $x(d) > a$. This will facilitate a handy presentation of the induction proof. After that we can state the uniqueness theorem.

Definition 2.42 *Let Ψ^*, and Ψ be two solutions, such that in one of them i is a loc-min, $i + 1$ is a stair-up (not loc-max), and $j = r(i)$ holds, whereas in the other solution the node $i+1$ is a loc-min, and $f(i) = x(i)+x(i+1) > f(i+1)$. Furthermore, assume that in both Ψ^* and Ψ the smallest loc-min in $\langle i + 2, j - 1\rangle$, has finish time above $x(i) + x(i + 1)$. The definitions of k^*, $d = L(k^*)$, $c = R(k^*)$, and $\Gamma' = \underline{C}$ is the same (analogous) as before.*

Let $\widetilde{\Gamma} := \max(\Gamma', x(i) + x(i + 1))$, and α be the number of odd-even conflicts below $\widetilde{\Gamma}$. Assume that $\{\Psi^*, \Psi\} = \{\Psi_a, \Psi_{a-\Delta}\}$, i.e., $\alpha = a$ in one of the solutions and $\alpha = a - \Delta$ in the other.

Now, if $x(d) > a$, then let $\widetilde{\mathcal{F}}(\alpha) = \sum_{k=i+2}^{j-1} f_{\Psi_\alpha}(k)$. The degenerate case when $j - 1$ becomes a loc-min is symmetric.

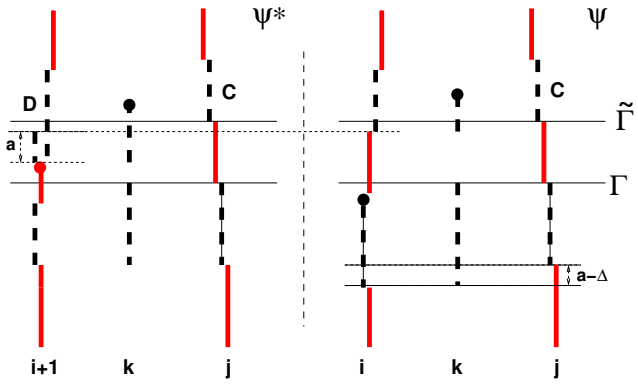

Figure 2.13: Definition 2.42.

Theorem 2.3 (uniqueness theorem) *Let* $\{\Psi^*, \Psi\} = \{\Psi_a, \Psi_{a-\Delta}\}$ *and* $d < k^* < c$ *be as defined above. If (2.a) holds, then*
(I) $k^* = pit_\Psi(i,j)$, $d = L_\Psi(k^*)$ *and* $c = R_\Psi(k^*)$ *also holds in* Ψ, *and*
(II) $\mathcal{F}(a-\Delta) = \mathcal{F}(a) - \Delta$.
(III) *In the degenerate case* $\widetilde{\mathcal{F}}(a-\Delta) = \widetilde{\mathcal{F}}(a) - \Delta$.

For $j-i > 3$ the proof of (I) is long and technical, and it is deferred to Section 2.7. Note that (I) implies recursively that all the solutions on $\langle i,j\rangle$ are essentially the same. Here we just show how (II) and (III) follow from (I).

Proof. The proof is by induction on the length of $\langle i,j\rangle$. Theorem 2.1 proves (I) and (II), and Theorem 2.2 proves (III) in the base case, when there is only one loc-max between i and j in two different solutions. Notice, that for $j = i+3$, this is the only possible case.

Now let $j - i > 3$. We prove (I) in the normal and in the degenerate case in Section 2.7. In that proof we will assume that (II) and (III) hold on *shorter* subpaths. Here we prove (II) and (III) on condition that (I) holds also on $\langle i,j\rangle$.

Proof of (II). We show that a decrease of a by Δ induces an increase of $f(k^*)$ by the same Δ. This is actually a simple corollary of (P2), (P4) and (I): below Γ' the number of black-black conflicts a is decreased by Δ, and the number of white-white conflicts is decreased by Δ. Originally, k^* is black on both of these level sets. Now – depending on the parity of k^* – on one of these level sets k^* becomes white, and it becomes black on the former $[f(k^*), f(k^*)+\Delta] \subset D \cap C$, thereby decreasing the number of black-black conflicts on $\langle \ell(k^*), k^*\rangle$ and on $\langle k^*, r(k^*)\rangle$. Consequently, by the induction hypothesis the optimum sum on $\langle d, k^*-1\rangle$ resp. on $\langle k^*+1, c\rangle$ each decrease by Δ. The finish times of the stairs $\langle i+1, d-1\rangle$ and $\langle c+1, j-1\rangle$ are constant, so the changes total to $\mathcal{F}(a-\Delta) = \mathcal{F}(a) + \Delta - \Delta - \Delta$.

Proof of (III). In the degenerate case in one of the two solutions $i+1$ is a stair-up (and not a loc-max). The proof is the same as for (II), except that for (II) we calculated with a finish time of $f(i+1) = x(i) + x(i+1)$ in both Ψ_a and $\Psi_{a-\Delta}$. Subtracting this constant from both sides of the equation (II), we obtain $\widetilde{\mathcal{F}}(a-\Delta) = \widetilde{\mathcal{F}}(a) - \Delta$. □

2.7 The proof of the uniqueness theorem

This section contains a collection of technical proofs which we omitted before. These proofs are not too difficult, since they only apply the tools presented in Section 2.4. Typically, some obvious way to prove this or that statement is easy to read out from an appropriate figure. However, to write up or read the proofs in detail is a lengthy and tiresome task.

We start with the proof of Proposition 2.41, claiming that 'bad' solutions, i.e., where inequality (2.a) does not hold, may occur only on the lowest floors of minimum-trees. After that, in the second subsection we state and prove two technical lemmas, that we later use in the uniqueness proof.

In Subsection 2.7.3 we turn to the proof of the uniqueness theorem. The statement of Theorem 2.3 (I) was that two arbitrary different solutions Ψ^* and Ψ always have the same $pit(i,j)$ node (given that (2.a) holds), moreover this node ends in the same pair of rungs. In the proof we assume the contrary, that either the two $pit(i,j)$ nodes are not the same, or they are the same but they end in different pairs of rungs. We subdivide the proof into five lemmas: Lemmas 2.50 and 2.52 exclude that the $pit(i,j)$ nodes are the same, resp. are of the same parity, but they end in different pairs of rungs; Lemma 2.51 excludes that in (only) one of the solutions there is a unique loc-max. These cases are relatively easy to handle. It requires much more elaboration to show that $pit_\Psi(i,j)$ and $pit_{\Psi^*}(i,j)$ cannot be different nodes (of the same parity), ending in the same rungs (Lemma 2.54); or nodes of different parity, ending in consecutive pairs of rungs (Lemma 2.56). The last case is the 'tightest' one, in particular this is the only lemma where we need to exploit the inequality (2.a). In other words, the last setting (different parity $pit(i,j)$ nodes) might occur if one of the solutions appears on the first or second floor of a minimum tree.

Having a close and exact view on all these cases, in the last subsection we can present procedure SELECT, which for given (i,j) pair of nodes, selects at most one even and at most one odd node inside $\langle i,j\rangle$, that might play the role of $pit(i,j)$ or $top(i,j)$, in solutions where $j = r(i)$ or $i = \ell(j)$.

The proofs in Subsections 2.7.1 and 2.7.3 are based on the notation introduced in the previous section.

2.7.1 The proof of Proposition 2.41

The statement of the proposition is the following:

If the inequality (2.a) does <u>not</u> hold, then Ψ_a may occur as part of a solution on $\langle 1,n\rangle$ only if in this solution $j = r(i)$ (resp. $i = \ell(j)$), and i (resp. j) is on at most 2nd floor of a minimum-tree.

Proof. According to Proposition 2.40, either of (R1) or (R2) implies (2.a), where
(R1) $x(d) > a$;
(R2) $d = i+1$ and $j = r(i)$, moreover $a \geq x(d) > 2a/3$, $x(c+1) > a$ and $|C| > a$.

Thus our goal is to show that either (R1) holds, or (R2) holds, or Ψ_a appears on a low floor of a minimum-tree, as stated in the proposition. In the first part of our

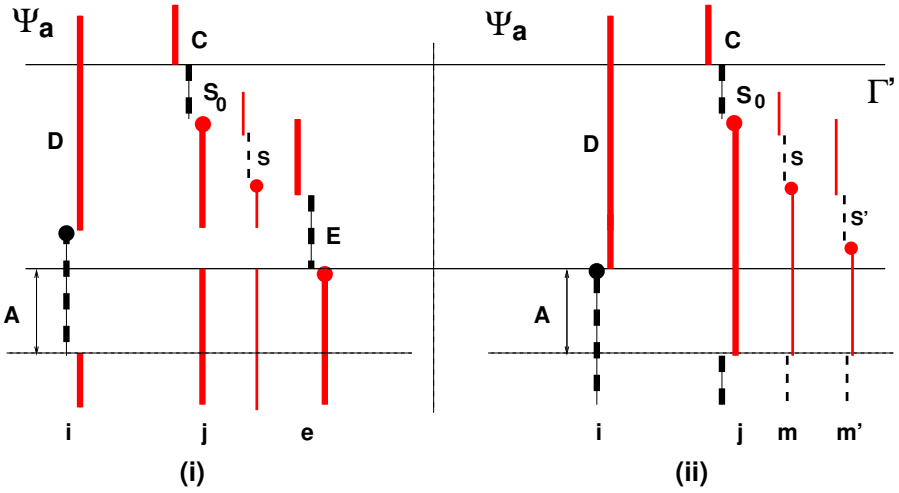

Figure 2.14: Proposition 2.41 (1): the solution Ψ_a if $i = \ell(j)$.

proof we consider the solution Ψ_a. Let A be the set of odd-even and B be the set of even-odd levels below Γ'. Now $|A| = a$ and $|B| = b$.

If D is a high rung, then Lemma 2.29 implies that $x(d) > |D| \geq |A| = a$, and (R1) holds.

If D is a second rung, then $d = i + 1$, and according to Lemma 2.31, either $(*)$ $x(d) > a$; or $(**)$ $e := R(i)$ is a high stair and $|E| > a$; or $(***)$ $r(i)$ is compact and i is the root of a minimum-tree. Since $(*)$ is the same as (R1), we need to analyze the cases $(**)$ and $(***)$.

(1) Assume that $i = \ell(j)$.

Recall that i is a loc-min (D is a second rung), moreover $i = \ell(j)$ implies that $j < r(i)$ (see Fig. 2.14). Suppose that i is non-compact. First we apply Lemma 2.31 to $\langle i, r(i)\rangle$. Observe that set A of the lemma stands for the same set where (i, j) have odd-even conflicts below Γ', so that $|A| = a$. In Lemma 2.31 we proved $(*)$ in all cases except when $t = 1$, where t was the length of the zigzag walk in that proof. In our case, $(*)$ may not hold only if e is a high stair, or it is a second stair above a compact loc-min *and* i is the root of a minimum-tree, as implied by the proof of Lemma 2.31.

We elaborate on these two cases, while now we apply Lemma 2.31 to $\langle i, j\rangle$, with $t = 2$. Note that now $L_{\Psi^*}(k^*) = d$ implies $S_0 \subset D$. Assume first, that e is a high stair. Now Claims 2.32 and 2.33 yield $|S_0| \geq |E| > |A|$, and so $|D| > |A|$, that is, (R1) holds.

Second, let e be a second stair above a compact loc-min, and i be the root of a minimum-tree (see Figure 2.14 (i)). If there are inserted loc-mins inside $\langle j, e\rangle$, and at least one inserted rung S, then S and S_0 are disjoint, $|A|/2 < |E| < |S| < |S_0|$ holds by Claims 2.32 and 2.33, and $S \cup S_0 \subset D$ proves $|D| > |A|$. If there are no inserted loc-mins, then j is on the 2nd floor of the minimum-tree.

Finally, suppose that i is compact. If there are at least 2 non-compact loc-mins m and m' and inserted rungs S and S' as drawn in Figure 2.14 (ii), then

$|A|/2 < x(s') < |S| < |S_0|$. If there is at most one such loc-min, then j is on at most 2nd floor of a minimum-tree.

(2) Assume that $j = r(i)$.

Now if $(***)$ holds for Ψ_a, then j is compact and i is the root in a minimum-tree. If $(**)$ holds, then $e = j - 1$ is a high stair in Ψ_a, and $|E| > |A|$. We show that in this case (2.a) holds. In this part we analyze the solution Ψ^* (which is either Ψ_a or $\Psi_{a-\Delta}$). Observe, that since E is a high rung in Ψ_a, it follows that in any partial solution where $j = r(i)$, the rung of e includes E as a subset. Therefore we assume w.l.o.g. that E is the rung of e in Ψ^* as well. Moreover, since we are in Ψ^*, we have $L(k^*) = d$ and $R(k^*) = c$.

If $c \neq e - 1$ (see Fig. 2.15 (ii)), then there exists an odd rung $Q \subset D$ and $a \leq |Q| < |D|$, otherwise we could apply (O3) to Q, so $|D| > a$ holds again.

If $c = e - 1$, then $x(c + 1) = |E| > a$. In the rest of the proof we show that the case $c = e - 1$ yields (R2). (see Fig. 2.15 (i)).

First we prove $|C| \geq |E|$, which implies $|C| > a$:

Let V be the topmost contiguous set in $\Psi^*(k^*)$, and W be the rung of $k^* + 1$. If $\overline{C} \leq \overline{W}$, then $|C| \geq |E|$ must hold, otherwise we could place C into E applying (O2) on the left, and gain at least 1.

If $\overline{C} > \overline{W}$, then $|W| \geq min(a, |V|)$, otherwise it would be worth placing W below Γ' : it is for free on the right, it costs at most $2|W|$ on the left and we gain $|W|+|V|$. Moreover, $|V| \geq |E|/2$, otherwise we would place it into the bottom of E, and get a better solution. Thus, $|W| + |V| \geq |E|/2 + |E|/2 = |E|$.

Let $G = [\overline{W} + 1, \overline{C}] \subset C$. If $|C| < |E|$, then placing C into E would cost $|C|$ on the right by (O3). Moreover, $|C| < |E| \leq |X| + |V|$ implies that G fits into $[f(i) + 1, \overline{E}]$. Therefore, applying (O1) and (O2) we pay 0 on the left, but gain more than $|C|$, a contradiction. So we proved $|C| \geq |E|$.

It remains to show that $x(d) > 2a/3$. Note that $x(d - 1) + x(d) > |E| + a > 2a$. Assume that $x(d) \leq 2a/3$. If we place all black levels of d into the bottom black levels of j, then $f(d)$ decreases by at least $4a/3$, but placing down is for free on the right by (O2), and costs at most $2x(d)$ on the left, a contradiction. □

2.7.2 Two lemmas

The next two technical lemmas are elementary building blocks of the proof of Theorem 2.3. They exclude the existence of a certain kind of solution on a subpath:

Definition 2.43 *If in some schedule, node k is a second stair above a (non-compact) loc-min l, and k has r black levels above $L(l) \cup R(l)$, we will call these r levels* **expensive levels**. *If k is not a second stair, then it has 0 expensive levels.*

Lemma 2.44 *Suppose that in a schedule Φ either $k - 1$ or $k + 1$ is not compact, and there exist some levels Λ and Π, such that $\Pi > \Lambda + 2\theta$, furthermore k has at most θ black levels above Λ, and at least θ white levels below Λ. If k has r expensive levels, and $f(k) \geq \Pi + r$, then Φ is not a solution.*

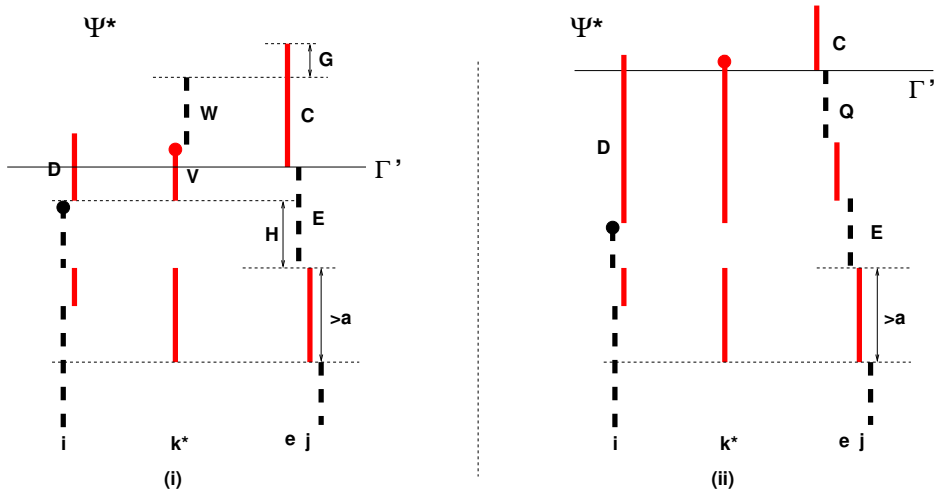

Figure 2.15: Proposition 2.41 (2): the solution Ψ^* if $j = r(i)$.

Proof. The proof follows immediately from (O1) – (O5) and Corollary 2.25. If k is not a second stair, then placing the θ black levels of k below Λ costs at most 2θ. If k is a second stair, then placing down costs at most $2\theta + r$, i.e., we pay 3 times only for the expensive levels. On the other hand, we reduce $f(k)$ by at least $\Pi - \Lambda + r > 2\theta + r$, so Φ was not optimal. □

Lemma 2.45 *Let Φ be a schedule on $\langle i, j\rangle$. If the following conditions hold on some subpath $\langle u, v\rangle \subset \langle i, j\rangle$, then Φ is <u>not</u> a solution (see Figure 2.17):*

– u and v have different parity (suppose w.l.o.g. that u is even and v is odd). Furthermore, either
(i) $v = R(u-1)$ or
(ii) $u = L(v+1)$,
and U and V are the rungs of u resp. of v, and $\Lambda := \min(\underline{U}, \underline{V}) - 1$.

– Below Λ, there is a set of odd-even levels G, a set of clear odd levels H, and a set of levels G', which is even on the right and $|G'| \geq |H|$ (G' and G might intersect). Moreover, $|U| > |G| + |H|$ and $|V| > |H|$ hold.

– An even node $u \leq k < v$ exists, s.t. one of the following hold: k is in the odd part $\langle u, m\rangle$ w.r.t. G, and k has at most $|G| + |H|$ black levels above Λ; or k is in the even part $\langle m+1, v\rangle$ w.r.t. G, and k has at most $|H|$ black levels above Λ.

Recall that the loc-max node m above exists by Corollary 2.16 (for simplicity, we added the node m to one of the two parts). In the next claim, the nodes u, v, and m as well as the set G are the same as in Lemma 2.45.

Claim 2.46 *There exist zero or more odd loc-mins $l_1 \dots l_\eta$, and zero or more even loc-mins $k_1 \dots k_\zeta$ with the following properties:*
– $u < l_1 < l_2 < \dots < l_\eta < k_\zeta < \dots < k_2 < k_1 < v$;
– $\forall \nu$, $f(l_\nu) < f(l_{\nu+1})$ and in $\langle l_\nu, l_{\nu+1}\rangle$ no loc-mins finish below $f(l_{\nu+1})$;

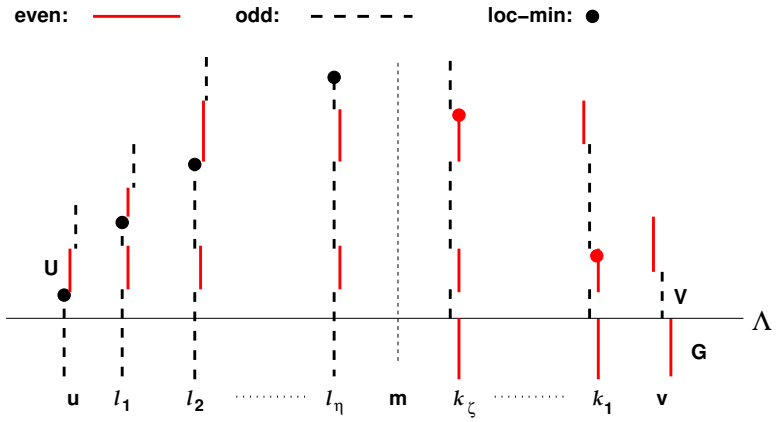

Figure 2.16: Illustration to Claim 2.46.

– $\forall \nu$, $f(k_{\nu+1}) > f(k_\nu)$ *and in* $\langle k_{\nu+1}, k_\nu \rangle$ *no loc-mins finish below* $f(k_{\nu+1})$;
– *there are no loc-mins in* $\langle l_\eta, k_\zeta \rangle$.

Moreover, if the loc-max m *separates the odd part and the even part w.r.t.* G, *then* $l_\eta < m < k_\zeta$.

Proof. The loc-mins can be found by induction (see Fig. 2.16): if $pit(u-1, v+1)$ is odd, then let it be l_1, if it is even, then let it be k_1. Suppose that so far we defined, e.g., k_1. Now if $pit(\ell(k_1), k_1)$ is odd, then let it be l_1; if it is even, then let it be k_2, and so on. The proof of $l_\eta < m < k_\zeta$ is straightforward based on (P1) – (P4). □ *Claim 2.46*

Proof of Lemma 2.45. Let $\xi := |G| + |H|$. We carry out a case-analysis based on the possible positions of k.

(1) Assume that k **is in the even part** $\langle m+1, v \rangle$.

Suppose that k is a stair-down above v, and let $s = v-1$. We show that $|S| > |H|$. Assume the contrary, that is $|S| \leq |H|$. Now we place S into H. It costs at most $|S|$ on the left by (O1) and $|S|$ on the right by (O3). But we got a better optimum because we decreased $f(s)$ by $|S| + |V| > |S| + |H| \geq |S| + |S|$. So we showed $|S| > |H|$, and therefore $k \neq s$ or any stair-down above s, because k has at most $|H|$ black levels above Λ.

In any other case $f(k_1) \leq f(k)$ holds. An even stair s' exists, s.t. $R(k_1) = s' \leq s = v-1$, where s' is not a second stair-up, and for all (odd and even) rungs $S', \ldots, S, V$ it can be shown like above, that $|S'| \geq \ldots \geq |S| \geq |V| > |H|$.

Observe, that according to Claim 2.46, except for the stair-downs above v, every finish time in the even part is at least $f(k_1)$. If $S' \neq S$, then every finish time is at least $f(k_1) > \overline{S}$. So, let $\Pi_1 := \overline{S}$. Now $\Pi_1 \geq \Lambda + |V| + |S| > \Lambda + 2|H|$, and Lemma 2.44 yields the proof.

If $f(k_1)$ ends in S on the right, then let z be one of the two neighbors of k_1 that has smaller finish time (see Figure 2.17 (ii)). Let $\tilde{T}$ and $\tilde{Z}$ denote the black levels of k_1, resp. of z *above* $\underline{V}$. We show that $|\tilde{T}| > |H|$. Assume the contrary $|\tilde{T}| \leq |H|$, then by Proposition 2.27 $|\tilde{T}| \geq |V| > |H|$ would hold. So we have $|\tilde{T}| > |H|$, and this implies $k \neq k_1$.

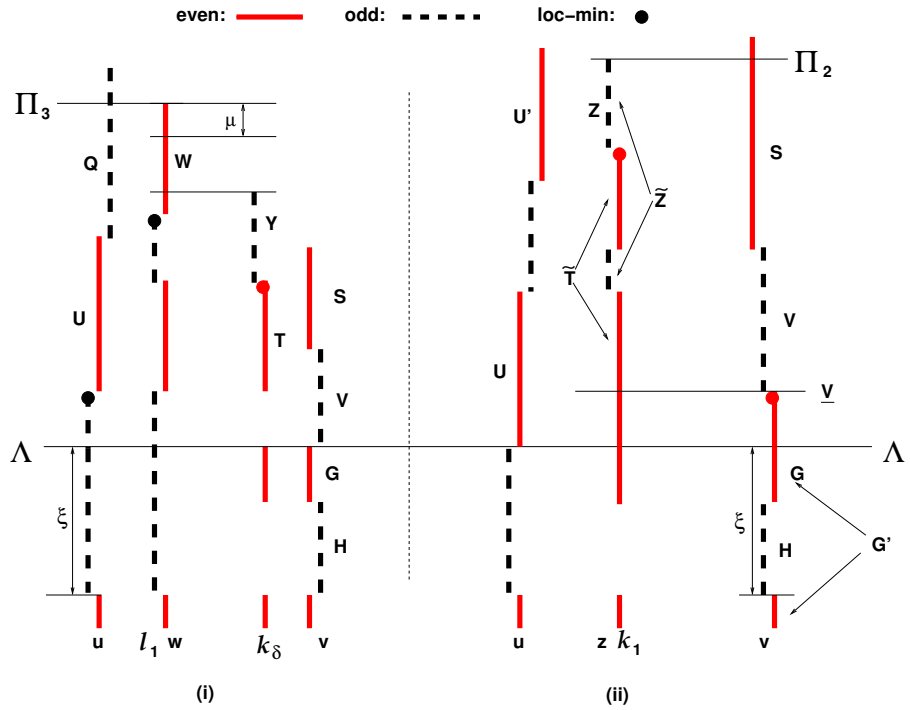

Figure 2.17: Illustration to Lemma 2.45 in cases (i) and (ii), respectively.

We set $\Pi_2 := \min(\overline{Z}, \overline{S})$. We claim that except for $f(k_1)$, by Claim 2.46 every finish time in the even part is at least Π_2 : the rungs Z and S are of different parity, so no loc-min ends between these rungs; moreover, since z is odd, k_2 ends above $\overline{Z}$.

We show $\Pi_2 > \Lambda + 2|H|$. This is obvious when $\Pi_2 = \overline{S}$, so let us assume that $\Pi_2 = \overline{Z} < \overline{S}$. We claim that $|\tilde{Z}| \geq |H|$, otherwise it would be worth placing $\tilde{Z}$ into G' : it would cost at most $2|\tilde{Z}|$, since z has 0 expensive levels. On the other hand, $f(z)$ would be reduced by at least $|\tilde{Z}| + |\tilde{T}| > |\tilde{Z}| + |H|$.

So, $|\tilde{T}| + |\tilde{Z}| > 2|H|$, and $\Pi_2 > \Lambda + 2|H|$. Due to Lemma 2.44, Φ is not a solution.

(2) Assume that k is in the odd part $\langle u, m \rangle$.

Notice that $k \neq u$, or any stair-up above u, since k has only at most $\xi < |U|$ black nodes.

Suppose that $k = m$ is the unique loc-max in $\langle u, v \rangle$, and m has less than $|U|$ black levels above Λ. Now $m \neq u$, and assume that m has μ black levels above $\overline{U}$. We could place these levels into U for free on the left and for μ on the right, and decrease $f(m)$ by more than μ, a contradiction.

In any other case $f(l_1) < f(k)$ holds. Suppose that w is one of the two neighbors of l_1, that has smaller finish time. Furthermore, let $f(l_1)$ end in Q on the left, i.e. Q is some odd rung above U.

Let $\Pi_3 = \min(\overline{W}, \overline{Q})$. Observe, that according to Claim 2.46, except for stairs above u, the finish time of any node in the odd part is at least Π_3. Moreover, even if $k = w$, the expensive levels of k are all above Π_3. If we prove $\Pi_3 > \Lambda + 2\xi$, then Lemma 2.44 implies that Φ is not a solution.

Claim 2.47 *In case* (2), $\Pi_3 = \min(\overline{W}, \overline{Q}) > \Lambda + 2\xi$.

Proof. If $\overline{W} > \overline{Q}$, then $|Q| \geq |U|$, otherwise placing Q into U provides a better optimum by (O2) on the right and by (O3) on the left. Thus, $\Pi_3 \geq \Lambda + |U| + |Q| \geq \Lambda + 2|U| > \Lambda + 2\xi$.

The case $\overline{W} \leq \overline{Q}$ is illustrated in Figure 2.17 (i). Now W has $|W|$ conflicts on the left. We show $|W| > \xi$. Assume the contrary, that $|W| \leq \xi$. We place W into $G \cup H$. Note that w is white on $G \cup H$ and has $|G|$ conflicts on the right. This implies that $|G|$ levels we can place there for free on the right (this remains true if some even loc-mins k_ν have finish time below $\overline{W}$).

Suppose that $f(l_1)$ ends in some odd rung Y on the right. If $\overline{W} \leq \overline{Y} + |G|$, then placing W into $G \cup H$ is for free on the left and free on the right, and it costs $|W|$ due to increasing $f(l_1)$; since we decrease $f(w)$ by more than $|W|$, this provides a better solution.

So, let $\overline{W} = \overline{Y} + |G| + \mu$. Now placing W into $G \cup H$ costs $|W|$ by increasing $f(l_1)$, and μ on the right; but we decrease $f(w)$ by at least $|Y| + |G| + \mu$. If we show $|Y| > |H|$, then $|Y| + |G| + \mu > |H| + |G| + \mu = \xi + \mu \geq |W| + \mu$ yields a contradiction, i.e., then we obtain $|W| > \xi$.

Note that either $y \leq v$ is an odd rung above v, or y is an odd rung above an even loc-min k_δ, that is $f(l_1) > f(k_\delta) \geq f(k_1)$. In the first case all (odd and even) rungs from V to Y have size larger than $|H|$: since none of these rungs is a second stair-up, we can show $|Y| \geq \ldots \geq |S| \geq |V| > |H|$ like in case (1).

Let us assume the second case, that Y is a rung above k_δ. If $f(k_\delta)$ ends in $U' \neq U$ on the left, then U is a set of clear even levels on $\langle u, k_\delta \rangle$ and since k has only at most $\xi < |U|$ black nodes, k cannot be in the odd part.

Thus, we can assume that $f(k_\delta)$ ends in U on the left. If T denotes the topmost contiguous black levels of k_δ, then in case (i) $|T| > |H|$ by Proposition 2.27; in case (ii) $V \subset U$, and $|T| > |V| > |H|$.

If Y is a high rung and $|Y| \leq |H|$, then we could place Y into G' for free on the left by (O2) and for at most $|Y|$ on the right by (O3), and decrease $f(y)$ by at least $|Y| + 1$. If, on the other hand Y is a second rung, and $|Y| \leq |H|$, then $y = k_\delta - 1$ and we place Y into G' : it is for free on the left by (O2), it costs at most $|Y|$ by increasing $f(k_\delta)$ and at most $|Y|$ on the right by (O1), while $f(y)$ is decreased by $|Y| + |T| > |Y| + |H|$. We could get a better solution in both cases, consequently $|Y| > |H|$ holds.

We showed $|W| > \xi$, and this implies $\Pi_3 \geq \Lambda + |U| + |W| > \Lambda + 2\xi$. □ *Claim 2.47*

□ *Lemma 2.45*

2.7.3 The proof of Theorem 2.3 (I)

Recall that $k^* = pit_{\Psi^*}(i, j)$, furthermore $d = L_{\Psi^*}(k^*)$ and $c = R_{\Psi^*}(k^*)$. We prove Theorem 2.3 (I) by contradiction, that is we assume that either $k^* \neq pit_\Psi(i, r(i))$ or $L_\Psi(k^*) \neq d$ or $R_\Psi(k^*) \neq c$. We will exclude the optimality of either Ψ^* or Ψ.

Proposition 2.48 $x(d) > a/2$ *and* $x(c+1) > a/2$, *furthermore* $|C| \geq b$.

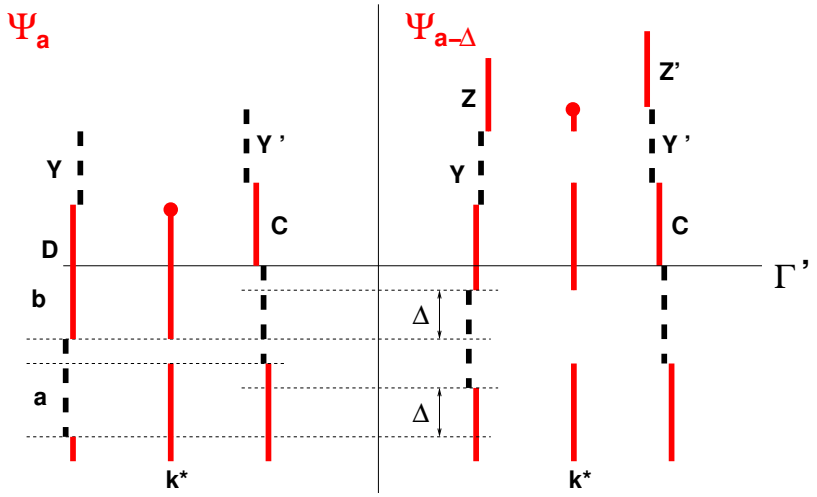

Figure 2.18: Illustration to Lemma 2.50.

Proof. We consider the solution Ψ_a.

If d is a high stair, then $x(d) > |D| \geq a$ by Lemma 2.29, since D is a high rung and $L(pit(i,j)) \geq d$. If d is a second stair, then $d = i+1$ or $d = j-1$ and $x(d) > a/2$ follows from Proposition 2.30. The proof of $x(c+1) > a/2$ is symmetric. Finally, since C is a high rung, $|C| \geq b$ follows from Lemma 2.29. □

From now on, we analyze the (hypothetical) solution Ψ. Lemmas 2.50 – 2.56 provide the proof of Theorem 2.3 (I) in the normal case, and given that $x(d) > a$, also in the degenerate case. The latter simply follows from the fact, that if $\widetilde{\Gamma} < f_{\Psi^*}(k^*)$ and $x(d) > a$, then none of the proofs uses any preassumption about the order of finish times below $\widetilde{\Gamma}$.

Proposition 2.49 *Suppose that $\Psi = \Psi_{a-\Delta}$ and $f(pit(i,j)) > \overline{C}$ or $pit(i,j) = \emptyset$ in Ψ. If S is a rung in the series of rungs above D or C, then $|S| > \Delta$.*

Proof. Suppose that $y := d+1$ is a stair-up. First we show that $|Y| > \Delta$. Assume the contrary, that $|Y| \leq \Delta$. In $\Psi_{a-\Delta}$ there are at least Δ clear even levels below Γ', and y is white on these levels. Suppose that Y has μ levels above $\overline{C}$. If we place Y down to the Δ white levels, it costs at most $|Y|$ on the left by (O3) and at most μ on the right by (O1); whereas $f(y)$ is decreased by at least $|C|+\mu \geq b+\mu \geq \Delta+\mu \geq |Y|+\mu$. If we did not decrease the sum of finish times, then $|C| = b = \Delta = |Y|$ must hold, and it is easy to verify, that placing Y below Γ' increases the sum of squares of finish times, so we obtained a better solution.

Now suppose that $y' := c-1$, is a stair-down, and $|Y'| \leq \Delta$. If we place Y' down to the Δ clear even levels below Γ', it costs at most $2|Y'|$ by (O1) and (O3); whereas $f(y')$ is decreased by at least $|C| + |Y'| \geq b + |Y'| \geq \Delta + |Y'| \geq 2|Y'|$. If we did not decrease the sum of finish times, then we obtained the same sum of finish times, and increased the sum of squares of finish times, a contradiction.

Finally, if $s = y+1$ is a stair-up, then $|S| > \Delta$, or $|S| \geq |Y|$: otherwise it would be worth placing S below Γ', applying (O1) and (O3). The same holds for

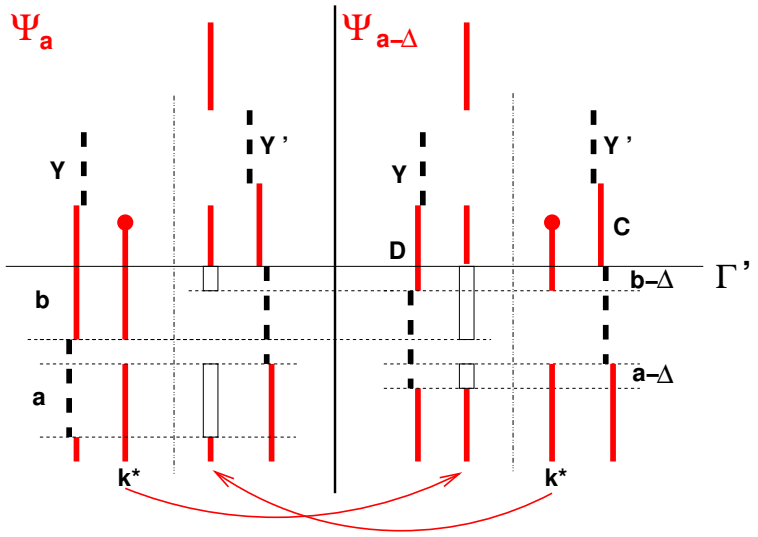

Figure 2.19: Illustration to Lemma 2.51, if $\Psi = \Psi_{a-\Delta}$, and if $\Psi = \Psi_a$. If k^* becomes the unique loc-max in Ψ then it has at most $a + b - \Delta$ black levels above Y.

$s' = y' - 1$. For rungs above S or S', the claim follows by induction. □

Lemma 2.50 *Suppose that $pit_\Psi(i,j) = k^*$, but either $d \neq L_\Psi(k^*)$ or $c \neq R_\Psi(k^*)$. Then Ψ is not an optimal schedule.*

Proof. We assumed $k^* = pit_{\Psi^*}(i,j) = pit_\Psi(i,j)$ and $f_{\Psi^*}(k^*) < f_\Psi(k^*)$. By (P1) – (P4) this is possible only if $\Psi^* = \Psi_a$ and $\Psi = \Psi_{a-\Delta}$.

Consider the schedule Ψ. Since the number of conflicts below Γ' decreased by Δ, the number of black levels of k^* above Γ' increased by Δ, so k^* has at most Δ black levels above $\min(\overline{D}, \overline{C})$ (see Figure 2.18).

Suppose that $L(k^*) = z \neq d$, and $R(k^*) = c$. Let $y := z - 1$. Now y is odd and $Y \subset C$, i.e., the complete Y has conflicts on the right, and therefore $|Y| \geq x(y-1) > b \geq \Delta$. Otherwise we could place Y below $\underline{Y} - 1$ for free on the right by (O1) and for $|Y|$ on the left by (O3), and reduce $f(y)$ by at least $|Y| + 1$. Since k^* has at most Δ black levels above $\underline{Y}$, and at least Δ white levels below $\underline{Y}$, Proposition 2.27 implies that it has at least $|Y| > \Delta$ levels above $\underline{Y}$, a contradiction.

Now suppose that $R(k^*) = z' \neq c$, and $y' := z' + 1$. Proposition 2.49 implies $|Y'| > \Delta$, and by Proposition 2.27, k^* has at least $|Y'|$ black levels above $\underline{Y'}$, a contradiction. □

Lemma 2.51 *Let $pit_\Psi(i,j) = \emptyset$, i.e. there is a unique loc-max in $\langle i,j \rangle$. Then Ψ is not an optimal schedule.*

Proof. From $k^* = pit_{\Psi^*}(i,j)$ it follows that $x(k^*) \leq x(d) + a$ and $x(k^*) \leq x(c) + b$.

Now we consider the schedule Ψ. Let $y = d + 1$ and $y' = c - 1$. If k^* is a stair-up above y, and Q denotes the black levels of k^* above $\overline{Y}$, then $|Q| \leq a$. If $\Psi = \Psi_a$, then there are a odd-even levels below Γ'; if $\Psi = \Psi_{a-\Delta}$, there are $a - \Delta$ odd-even levels and Δ clear odd levels below Γ'. We place Q into these levels: it costs at most $|Q|$

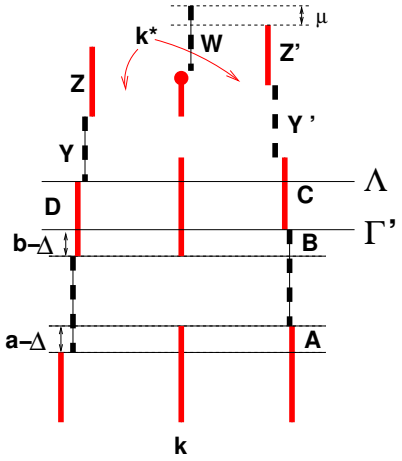

Figure 2.20: Illustration to Lemma 2.52.

on the left by (O3), and if $\Psi = \Psi_{a-\Delta}$, then it costs Δ on the right. If $\Psi = \Psi_{a-\Delta}$, then $|Y| > |\Delta|$ holds by Proposition 2.49. Therefore, $f(k^*)$ is reduced by at least $|Q|+1$, resp by $|Q|+|Y| > |Q|+\Delta$, so we obtained a better solution. The proof is symmetric in case k^* is a stair-down above y'.

The case when k^* is the unique loc-max in Ψ, is illustrated in Figure 2.19. If Q denotes the set of black levels of k^* above $\overline{Y}$, then $|Q| \leq a+b-\Delta$. Placing back Q below Γ' costs $|Q|$ if $\Psi = \Psi_a$, resp. it costs $|Q|+\Delta$ if $\Psi = \Psi_{a-\Delta}$. On the other hand, k^* is white on the levels of Y, so placing down reduces $f(k^*)$ by at least $|Q|+|Y|$. By Proposition 2.49, in $\Psi_{a-\Delta}$ this is more than $|Q|+\Delta$, so in either case we constructed a better solution, so Ψ was not optimal. □

Lemma 2.52 *Let $pit_\Psi(i,j) = k$, where $k \neq k^*$ is even. If either $L_\Psi(k) \neq d$ or $R_\Psi(k) \neq c$, then Ψ is not optimal.*

Proof. We analyze the solution Ψ, and prove that it is not optimal. We exclude any possible position of k^* in Ψ (see Figure 2.20). Let A denote the set of odd-even levels below Γ' in Ψ. Recall that either $|A| = a$, or $|A| = a-\Delta$.

The argument that k^* is not a stair above d or c with finish time below $f(k)$, is the same as in the proof of Lemma 2.51.

We will apply Lemma 2.45 with some $\Lambda \geq \Lambda' := \min(\overline{C}, \overline{D})$. The subpath $\langle i,k\rangle$ has $|A|$ odd-even levels below Γ'; moreover, if $\Psi = \Psi_{a-\Delta}$, then it has at least Δ clear odd levels. If k^* is in the odd part of $\langle i,k\rangle$, then it has at most a black levels above Λ'; if it is in the even part and $\Psi = \Psi_{a-\Delta}$, then it has at most Δ black levels above Λ'. A symmetric statement holds on $\langle k,j\rangle$, with a replaced by b.

Suppose first that $z' := R(k) \neq c$. Let $z := L(k)$. If $z \neq d$ then we show $|Z| > a$: Let Y be the odd rung right below Z. Assume that $|Z| \leq a$. Placing Z below Γ' costs $|Z|$ on the left by (O3); it costs additional Δ on the right if $\Psi = \Psi_{a-\Delta}$. On the other hand, we decrease $f(z)$ by $|Z|+|Y|$. In $\Psi_{a-\Delta}$, by Proposition 2.49 $|Y| > \Delta$, so we obtained a better solution, a contradiction.

If $z = d$, then $Y' \subset D$, Y' can be placed below Γ' for free on the left, so $|Y'| > a$. This implies that D has more than A levels above Λ'. In both cases $|Z'| > b$ can be

shown the same way as $|Z| > a$ was proven.

Let w be one of the two neighbors of k, having smaller demand. We show that if $\Psi = \Psi_{a-\Delta}$, then $|W| > \Delta$. In this case we have at least Δ clear odd and Δ clear even levels below Γ'. Assume that $|W| \leq \Delta$. If $\overline{W} < \overline{Z'}$ then by Proposition 2.26, $|W| \geq |Z| > a \geq \Delta$. If $\overline{W} = \overline{Z'} + \mu$, then by applying (O2) we get $|Z'| \geq x(z'+1) \geq b + |Y'| > 2\Delta$. Placing down W costs at most $2|W| + \mu \leq 2\Delta + \mu$, but $f(w)$ decreased by at least $|Z'| + \mu > 2\Delta + \mu$, a contradiction.

We apply Lemma 2.45 on $\langle z, k-1\rangle$ with $U := Z$, $V := W$, $G := A$ and either $H := \emptyset$ or H is a set of odd-odd levels s.t. $|H| = \Delta$. If $Z = D$, then let U be the levels of D above Λ' (Lemma 2.45 remains valid with this minor modification). On $\langle k+1, z'\rangle$, the lemma can be applied in he same way.

Finally, the case $L(k) \neq d$ (and $R(k) = c$) can be exluded by a symmetric argument. □

Proposition 2.53 *If in the solution Ψ_a some even node $\bar{k} = pit(i,j)$, moreover $L(\bar{k}) = d$, and $R(\bar{k}) = c$, then $|C| > a/2$.*

Proof. If $b > a/2$, then Proposition 2.48 implies $|C| \geq b > a/2$. Otherwise $\Delta \leq b \leq a/2$ holds.

For an illustration see Figure 2.21, where $\bar{k} = k_2$. Let w be one of the two neighbors of $\bar{k}$ having smaller demand.

If $\overline{W} \geq \overline{C}$, then $|C| \geq x(c+1)$: otherwise placing C down costs 0 on the left by (O2) and $|C|$ on the right by (O3). So, by Proposition 2.48 $|C| \geq x(c+1) > a/2$.

If $\overline{W} < \overline{C}$, then $|C| > |W| \geq a$, otherwise W could be placed into A for free on both sides, $f(\bar{k})$ would increase by $|W|$, and $f(w)$ would decrease by more than $|W|$, yielding a better solution. □

Lemma 2.54 *Suppose that $pit_\Psi(i,j) = k$, where $k \neq k^*$ is even, $L(k) = d$, and $R(k) = c$. Let $\{k_1, k_2\} := \{k^*, k\}$ s.t. $k_1 < k_2$, and $\{\Psi_1, \Psi_2\} := \{\Psi^*, \Psi\}$, s.t. $k_\xi = pit_{\Psi_\xi}(i,j)$ $(\xi = 1, 2)$.*
– If $x(k_1) \leq x(k_2) - a/2$, then Ψ_2 is not optimal;
– If $x(k_1) > x(k_2) - a/2$, then Ψ_1 is not optimal.

Proof. (1) $x(k_1) \leq x(k_2) - a/2$.

In case (1), A denotes the set of odd-even levels below Γ' in Ψ_2. We will analyze Ψ_2, and show that it is not optimal. Note that either $|A| = a$, or $|A| = a - \Delta$, depending on whether Ψ_2 is Ψ_a or $\Psi_{a-\Delta}$. On Figure 2.21 we illustrated the case $\Psi_2 = \Psi_a$. We show that $x(k_1)$ is too small to finish above $f(k_2)$.

The only levels under $f(k_2)$ where k_2 is black and k_1 may be white are the odd-even levels, that is the set A. Since $x(k_1) \leq x(k_2) - a/2$, there may be at most $|A| - a/2$ black levels of k_1 above $f(k_2)$.

Let W denote the rung of $k_2 - 1$.

(1.1) Assume that k_1 is a stair below $pit(d-1, k_2)$, or unique loc-max.

Clearly, k_1 is not a stair above k_2, since $x(k_1) < x(k_2)$; it is also not a stair or loc-max above D, because it would be worth placing the rung of k_1 into A.

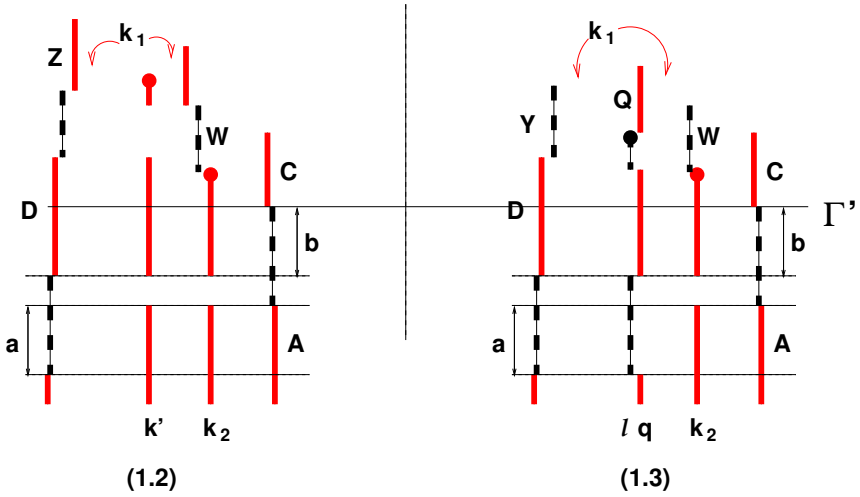

Figure 2.21: Illustration to Lemma 2.54 (1).

(1.2) Assume that $k' := pit(d-1, k_2)$ **is even, and** $f(k_1) \geq f(k')$.

For illustration see Figure 2.21 (1.2). Now $k' \leq k_1 < k_2$ is impossible, since there are $x(k_2)$ clear even levels below $f(k_2)$, and $x(k_1) < x(k_2)$. So, $d < k_1 \leq k'$ holds.

Suppose first, that $L(k') = d$. Then $\overline{D} > \overline{W}$, and placing W below Γ' is for free on the left. According to Propositions 2.26 and 2.53, $|W| \geq \min(a, |C|) > a/2$. Since W has more than $a/2$ levels, and the levels of W are clear even on $\langle d, k' \rangle$, it follows that $f(k_1) < f(k')$, a contradiction.

Second, suppose that $L(k') = z$, where $z \geq d+2$. Now $|Z| > |A|$, otherwise we would place Z into A, for free on the right, and using (O3) on the left.

Finally, we apply Lemma 2.45 with $(u, v) := (z, k'-1)$; $G := A$; and $H := \emptyset$. The conditions of the lemma hold, because $|Z| > |A|$, and k_1 has less than $|A|$ black levels above $\underline{Z}$ in the odd part and 0 levels above $\underline{Z}$ in the even part.

(1.3) Assume that $l := pit(d-1, k_2)$ **is odd, and** $f(k_1) \geq f(l)$.

For illustration see Figure 2.21 (1.3). Assume that $L(l) = y > d$ and $R(l) = w < k_2$. Let q be the neighbor of l having smaller demand. Since Q can be placed into A for free on the right, by Proposition 2.26 $|Q| \geq \min(|A|, |Y|)$. If $\overline{Y} > \overline{Q}$, then $|Y| > |Q| > |A|$.| If $\overline{Y} \leq \overline{Q}$, then placing Y into the black levels of d would cost 0 on the right by (O2) and $|Y|$ on the left by (O3). Thus, $|Y| \geq x(d)$, and Proposition 2.48 implies $x(d) > a/2$, so that $|Q| \geq |Y| > a/2$.

If $y < k_1 < l$, then we apply Lemma 2.45 on $\langle y, q \rangle$ so that $G := \emptyset$; H is $|A| - a/2$ levels of A; and G' is the black levels of d. Since $|G'| > a/2 \geq |H|$, and k_1 has at most $|A| - a/2$ levels above $\underline{Y}$, the conditions of the lemma hold.

If $l < k_1 < k_2$, then we apply Lemma 2.45 on $\langle q, w \rangle$ with $G := A$; and $H = \emptyset$.

(2) $x(k_2) = x(k_1) + \theta < x(k_1) + a/2$.

The case $x(k_2) \leq x(k_1)$ can be easily excluded following the lines of the argument below. Therefore, we will assume w.l.o.g. that $\theta > 0$. In (2) we regard the solution Ψ_1, and show that it is not optimal. Let A denote the set of odd-even, resp. B denote the set of even-odd levels below Γ' in Ψ_1. Figure 2.22 depicts the case when

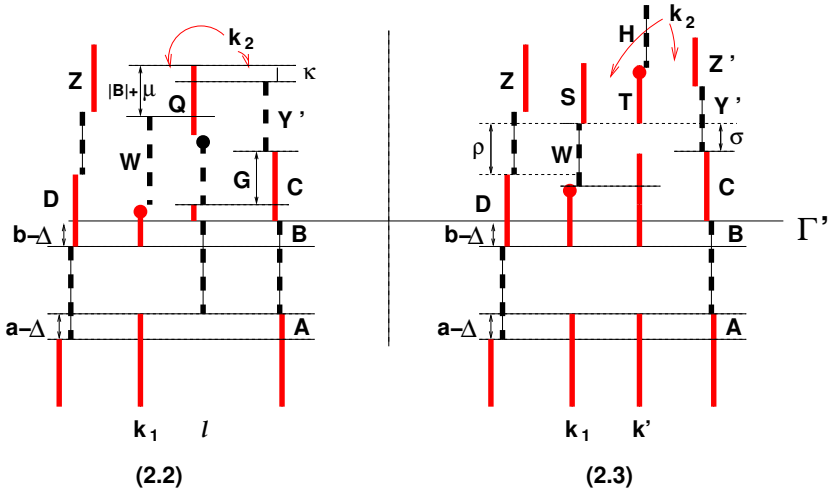

Figure 2.22: Illustration to Lemma 2.54 (2).

$|A| = a - \Delta$, and $|B| = b - \Delta$.

The only levels under $f(k_1)$, where k_1 is black and k_2 may be white are the levels of B. Therefore, there are at most $|B| + \theta < |B| + a/2$ black levels of k_2 above $f(k_1)$.

Claim 2.55 *In case* (2)*, if* W *denotes the rung of* $k_1 + 1$ *in* Ψ_1*, then* $|W| \geq a/2$.

Proof. We assume the contrary, that $|W| < a/2$. Since obviously w has at least a white levels below Γ', it is possible to place W below Γ'.

If $\overline{D} \geq \overline{W}$, then placing down is free on the left, and Propositions 2.26 and 2.48 imply $|W| \geq |C| > a/2$. Thus, we can assume that $\overline{W} = \overline{D} + \rho$; moreover $\rho > |A|$, otherwise placing down W would still be free on the left (see Figure 2.22 (2.3)). Consequently, $|A| < \rho \leq |W| \leq a/2$, implying $|A| = a - \Delta$ and $\Delta > a/2$.

We claim that in this case $|D| > a/2$: Note that we can place D into the black levels of $d-1$ for free on the right by (O2). If D is a high rung then by (O3) it costs $|D|$ on the left, and therefore $|D| \geq x(d-1) \geq a$. If D is a second rung, then there are $\Delta > a/2$ contiguous black levels of $d-1$ below $f(d-1)$, and placing D into the bottom $a/2$ black levels costs at most $2|D|$ on the left, and reduces $f(d)$ by more than $|D| + a/2$, a contradiction. So, we showed $|D| > a/2$.

Now, if $\overline{W} \leq \overline{C}$, then, by Proposition 2.26, $|W| \geq |D| > a/2$. So, let $\overline{W} = \overline{C} + \sigma$. Placing W below Γ' costs $\rho - (a - \Delta)$ on the left, σ on the right and $|W|$ at k_1. On the other hand, $f(w)$ is reduced by at least $|C| + \sigma \geq b + \sigma \geq \Delta + \sigma$. If the solution was optimal, then $\Delta + \sigma \leq \rho - (a - \Delta) + \sigma + |W|$ implying that $a \leq \rho + |W| \leq 2|W|$ and $a/2 \leq |W|$. □ *Claim 2.55*

(2.1) Assume that k_2 is a stair above W or C, or the only loc-max in $\langle k_1, c \rangle$.

If k_2 were a stair-up above W, then placing down θ levels of k_2 would cost at most 2θ, but it would reduce $f(k_2)$ by at least $|W| + \theta$, and $\theta < a/2 \leq |W|$, a contradiction.

Suppose that k_2 is a stair-down above C or unique loc-max. Let $y' := c - 1$. Now $|Y'| \geq |C| > a/2 > \theta$, otherwise it would be worth placing Y' below Γ'. If we place

the $|B|+\theta$ levels of k_2 below $\overline{C}$, that costs $|B|+\theta$ on the right and at most θ on the left. On the other hand, $f(k_2)$ is reduced by at least $|B|+\theta+|Y'|>|B|+2\theta$, a contradiction.

(2.2) Assume that $l := pit(k_1, c+1)$ is odd, and $f(k_2) > f(l)$.

For illustration see Figure 2.22 (2.2). Let $L(l) = w' \geq w$ and $R(l) = y' \leq c$. Let q be one of the two neighbors of l having smaller demand.

We prove that $|Q| > |B|+\theta$. Suppose that the contrary holds, i.e., $|Q| \leq |B|+\theta$, and therefore Q fits below $\overline{C}$, just like $x(k_2)$

Assume first that $\overline{Q} = \overline{Y}' + \kappa$ for $\kappa \geq 0$. Now placing down Y' into the black levels of c would be free on the left by (O2), and it would cost $|Y'|$ on the right by (O3), but $f(y')$ would reduce by more than $|Y'|$. Consequently, $|Y'| \geq x(c) \geq a+|C| \geq a+b$. If we place Q below $\overline{C}$, then it costs at most θ on the left, at most κ on the right and $|Q| \leq |B|+\theta$ at $f(l)$. On the other hand, we reduce $f(q)$ by at least $|Y'|+\kappa \geq a+b+\kappa > \theta+\theta+|B|+\kappa$.

Second, assume that $\overline{Q} < \overline{Y}'$. In this case we can place Q below $\overline{C}$ for free on the right. If it is also free on the left, then placing down costs only $|Q|$ at $f(l)$, but $f(q)$ is reduced by more than $|Q|$.

So, we assume $\overline{Q} = \overline{W}' + |B| + \mu$, and placing down Q costs altogether $|Q|+\mu < |B| + a/2 + \mu$. We show that $x(w') > a$. If $w < w'$, then obviously $|W'| > a$, since otherwise we could put W' below Γ' for $|W'|$ on the left by (O3), and for free on the right by (O2).

Suppose that $w = w'$. Note that $|A|$ levels of W can be placed into A for free. Thus, if $|W| \leq |A|$, then we could place down W for only $|W|$ at $f(k_1)$, and therefore $|W| > a$ follows again, a contradiction. Moreover, if $|A| = a - \Delta$, then there are at least Δ clear odd levels below Γ', so we obtain $x(w) \geq |W|(+\Delta) > a$. Finally, since $|W| \geq a/2$, this implies that placing down Q, reduces $f(q)$ by at least $a/2+|B|+\mu$ (i.e., Q fits below the top $a/2$ levels of W).

Now the proof of $|Q| > |B|+\theta$ is complete, and this implies that $k_2 \neq q$. Furthermore, $|Y'| > |B|+\theta$ also follows in both of the previous cases.

If $k_2 \in \langle l+1, y' \rangle$, the we apply Lemma 2.45 with $G = [f(k_1)+1, \overline{C}]$ and arbitrary H, s.t. H is clear odd, and $|H|+|G| = |B|+\theta$. Note also, that we can find an appropriate G', since $x(c) \geq a+b$.

If $k_2 \in \langle w'+1, l-1 \rangle$, the we apply Lemma 2.45 where $G := B$, $\Lambda := \overline{W}' - a/2$ and H is clear odd below Λ, s.t. $|H| = \theta$.

(2.3) Assume that $k' := pit(k_1, c+1)$ is even, and $f(k_2) \geq f(k')$.

See Figure 2.22 (2.3). Assume first, that $R(k') = z' \neq c$ for some even stair z'. Let $y' := c-1$, and T the set of black levels of k' above $\underline{Y}'$. We show that $|T| > \theta$. By Proposition 2.27, if T fits below $\underline{Y}'$, then $|T| \geq |Y'|$. On the other hand, $|Y'| \geq \min(a, |C|) \geq a/2 > \theta$, otherwise it would be worth placing Y' below Γ'. Thus, $x(k') > x(k_1) + \theta$.

Second, if $R(k') = c$, then $x(k') > x(k_1) + |W| \geq x(k_1) + a/2$. This proves that $k_2 \neq k'$ in any of these cases.

Let $s = L(k')$. If $k_2 \in \langle k_1, k' \rangle$, then k_2 has at most θ black levels above $\underline{S}$ in the odd part, and 0 black levels above $f(k')$ in the even part. Let H be the smaller rung

above $f(k')$. If $\overline{S} \leq \overline{H}$, then $|S| \geq |W| > a/2$, and k_2 has less than $|H|$ black levels above $f(k')$. Lemma 2.45 is easily applicable with $\Lambda := f(k')$. If $\overline{S} > \overline{H}$, then H can be put below Γ' for free on the left, and $|H| > a/2$ is straightforward if $R(k') = c$; if $R(k') = z'$, then either $|H| > a$, or $|H| \geq |Z'| > |Y'| > a/2$ can be shown. Finally, Lemma 2.45 is applicable with $\Lambda := \underline{S}$.

If $k_2 > k'$, then due to $x(k') > x(k_2)$, k_2 cannot be in the even part of $\langle k', c\rangle$; in the odd part it has at most $|B| + \theta$ black levels above $\overline{C}$, and these black levels fit below $\overline{C}$.

Suppose that $R(k') = z' \neq c$ and $k_2 \in \langle k', z'\rangle$. If $\overline{H} > \overline{Z}'$, then $|Z'| \geq b + |Y'| > b + a/2 > |B| + \theta$. Since $\overline{H} > \overline{Z}'$, k_2 has at most $|H|$ levels above $f(k')$, and we can apply Lemma 2.45 with $\Lambda := f(k')$.

Finally, let $R(k') = c$ and k_2 be in the odd part of $\langle k', c\rangle$. Now $x(k') > x(k_1)+a/2$, and so k_2 has less than $|B|$ black levels above $f(k')$. Obviously, k_2 is not a stair or a loc-max above c, since it would be worth placing it into B for free on the left. Consequently, there exists an odd loc-min l^*, s.t. $k' < l^* < c$. Let $R(l^*) = v'$, and U be the smaller even rung above $f(l^*)$. Now $f(k_2) \geq \min(\overline{U}, \overline{V'})$. If $\overline{U} \leq \overline{V'}$, then placing U into B is for free on both sides, and costs $|U|$ at $f(l^*)$, so $|V'| > |U| > |B|$. If $\overline{U} > \overline{V'}$, then placing V' into C would cost $|V'|$, so $|V'| > |C| \geq b$. Moreover, since placing U into B is free on the left, Proposition 2.26 implies $|U| \geq |V'| > b \geq |B|$. We can apply Lemma 2.45 on $\langle k', l^*\rangle$ with U, $V := \emptyset$, and $G := B$; respectively on $\langle l^*, y''\rangle$ with U, $V := V'$, $H := B$ and $G' := C$. □ *Lemma 2.54*

The proof of the following lemma is a bit more tricky than the previous proofs. Here we had to assume that the inequality (2.a) holds, in order to exclude the optimality of Ψ or Ψ^*.

Recall that (2.a) implies either
(R1) $x(d) > a$; or
(R2) $d = i + 1$ and $j = r(i)$, moreover $a \geq x(d) > 2a/3$, $x(c + 1) > a$ and $|C| > a$.

The above methods easily lead to applications of Lemma 2.45, if $x(d) > a$. Otherwise, i.e., in case (R2) we need to make a careful comparison of Ψ^* and Ψ. We do this by applying the induction hypothesis that Theorem 2.3 (II) and (III) hold on shorter subpaths than $\langle i, j\rangle$. Claims 2.60 and 2.63 discuss the two crucial cases when this induction step is necessary. For other parts of the proof only a high level argument will be presented.

Throughout the proof, we analyze the solution Ψ, and in all but one case we show that Ψ is not optimal. In particular, (R1) always excludes the optimality of Ψ. The only exception, i.e., when we exclude the optimality of Ψ^*, will occur in Claim 2.62. We will exploit this fact in procedure CHOOSE (see Section 2.7.4), which chooses one of the odd and even candidates to (possibly) be $pit(i, j)$.

Lemma 2.56 *Suppose that (2.a) holds. Let $pit_\Psi(i, j) = l$, where l is odd. Then either Ψ or Ψ^* is not an optimal schedule.*

Proof. We analyze Ψ (see Figure 2.23). Let A denote the set of odd-even levels, resp. B denote the set of even-odd levels below Γ' in Ψ. Like before, $|A| = a$ if

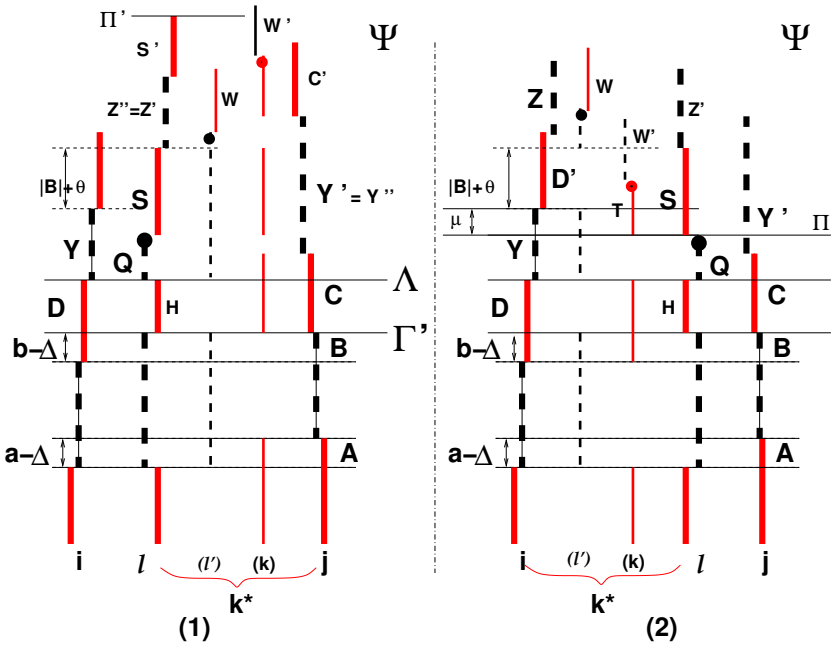

Figure 2.23: Illustration to Lemma 2.56 if $l < k^*$ (1), resp. if $k^* < l$ (2).

$\Psi = \Psi_a$, but $|A| = a - \Delta$ if $\Psi = \Psi_{a-\Delta}$. Let $\Lambda := \min(\overline{D}, \overline{C})$. Suppose that $f(l)$ ends in Y on the left and in Y' on the right for some odd stairs y and y'.

(1) Suppose that $l < k^*$.

Let $s := l + 1$.

(1.1) Assume that $|S| > a + b - \Delta$.

On the subpath $\langle l, j \rangle$, the levels of A are odd-even, and there are at least b clear odd levels. If k^* is in the odd part wrt. A, then it has at most $a+b-\Delta$ black levels above Λ. If k^* is in the even part, then it has at most b black levels above Λ. Notice that $|Y'| > |C| \geq b$ follows from (O2) if $\overline{S} > \overline{Y'}$, and $|Y'| > b$ is obvious if $\overline{S} \leq \overline{Y'}$. Consequently, Lemma 2.45 applied with $U := S$ and $V := Y'$ excludes this case, since $|S| > a+b-\Delta$ and $|Y'| > b$, and there are $|C| > b$ white levels of y' below $\underline{Y'}$.

(1.2) Assume that $|S| \leq a + b - \Delta$.

Let Q denote the topmost contiguous set of black levels of l.

Claim 2.57 *In case* (1)*, if* $|S| \leq a+b-\Delta$*, then* $\overline{Y} + |B| + |Q| \leq \overline{S} < \overline{Y'}$*. Moreover,* $y = d + 1$*,* $|Y| \leq a$ *and* (R2) *holds.*

Proof. Let $\overline{S} = \overline{Y} + |B| + \theta$. If $\overline{S} > \overline{Y'}$, then $|Y'| > a + |C| \geq a + b$ by (O2).

Suppose first that $\overline{S} = \overline{Y'} + a - \Delta + \mu$, then placing S below Γ' costs μ on the right and at most a on the left. We obtain $a + b + a - \Delta + \mu < |Y'| + a - \Delta + \mu \leq |S| + |Q| \leq a + b - \Delta + |Q|$ Therefore $|Q| > a + \mu$, and it is worth placing S below Γ', a contradiction.

Second, suppose that $\overline{S} = \overline{Y'} + \mu' \leq \overline{Y'} + a - \Delta$. Now S can be placed below Γ' for free on the right, and for $\max(\theta, \mu')$ on the left. Thus, placing down costs $|S| + \max(\theta, \mu')$.

On the one hand, we reduce $f(s)$ by $|Y'| + \mu' > a + b + \mu' \geq |S| + \mu'$. This excludes that $\mu' = \max(\theta, \mu')$, so the above placing down cost $|S| + \theta > |S| + \mu'$.

On the other hand, we reduce $f(s)$ by $|Y|+|B|+\theta$. If $|Y| > a$, then $|Y|+|B|+\theta > |S|+\theta$, a contradiction. Assume that $|Y| \leq a$. Now it is easy to show that $y = d+1$, moreover $x(d) \leq |Y| \leq a$ contradicting (R1), so that (R2) holds. Now comparing the levels on the two sides in Ψ we have $a + a + |S| \geq x(d) + |Y| + |B| + \theta > b + |C| + |Y'| \geq b + |C| + |C| + a \geq b + 3a$, so $|S| > a + b$, a contradiction.

Third, if $\overline{S} \leq \overline{Y'}$, then placing S below Γ' costs $|S| + \theta$, and we gain $|S| + |Q|$, so $\theta \geq |Q|$. On the other hand, we would gain at least $|Y| + |B| + \theta \leq |S| + \theta$, consequently $|Y| \leq a$, so $y = d + 1$, and (R2) hold. □ *Claim 2.57*

(1.2.1) Assume that $l' = pit(l, y' + 1)$ is odd, and $f(k^*) > f(l')$.

Let $L(l') = z''$ and $R(l') = y''$, and let w be the neighboring node of l' of smaller demand. Finally, let $H := [\underline{C}, \overline{D}]$. Observe first, that $x(s) > |S| + |H| \geq |B| + |Q| + |H| \geq |B| + |C| > a + b - \Delta$.

Now it is easy to show that $|W| > a + b - \Delta$:

If $\overline{W} < \overline{Y''} + a - \Delta$, then W is free to place down on the right, and $|W| > a+b-\Delta$, or $|W| \geq |Z''| \geq x(s) > a + b - \Delta$. If $\overline{W} = \overline{Y''} + a - \Delta + \rho$, then placing down costs at most $|W| + a + \rho$, but we gain $|Y''| + \rho + a - \Delta > a + b - \Delta + a + \rho$. Finally, $|Z''| > a + b - \Delta$ and $|Y''| > a + b - \Delta$ are easy to see, and Lemma 2.45 applies.

(1.2.2) Assume that $k = pit(l, y' + 1)$ is even, and $f(k^*) \geq f(k)$.

Let $z' := s + 1$. Now $|Z'| \geq |S| \geq |Y| > b$. We will apply Lemma 2.44. We have $\overline{S} - \Lambda \geq |Y| + |B| + |Q| > |H| + |Q| + |B| > |C| + |B| > a + |B|$. Furthermore, $|Z'| \geq |S| \geq |Y| > b$, and so $\overline{Z'} > 2b + a - \Delta + \Lambda$. If $k^* \geq k$, then k^* has at most b black levels above Λ, so Lemma 2.44 excludes this case.

Let $w' := k-1$; $s' := L(k)$ and $c' := R(k)$. If $s < k^* < k$, then k^* has its expensive levels above $\Pi' := \min(\overline{S'}, \overline{W'})$. If $\overline{S'} \leq \overline{W'}$, then $|S'| > x(l) > a$. If $\overline{S'} > \overline{W'}$, then $|W'| \geq |C'| \geq x(c + 1) > a$. The latter holds, because all the rungs above C can be shown by induction to have size at least a. Since $\Pi' > \Lambda + 2b + 2a - \Delta$, Lemma 2.44 can be applied.

(1.2.3) Assume that k^* is finished below $pit(l, y' + 1)$.

We will show in Claim 2.60 that $k^* \neq l + 1$. Here we will exclude that $k^* > l + 1$ holds. Note that k^* is not a stair-up above $l + 2$, because $|S| > |Y| > b$, and k^* would have at most $a - \Delta$ black levels above $\overline{S}$, so it would be worth placing these levels into A. Similarly, k^* is not a stair-down above $c - 1$, since it would cost Δ on the left to place it down but $|Y'| > \Delta$. Also, k^* is not a unique loc-max, since it would cost at most $a + b$ to place it down, and we would gain much more.

(2) Suppose that $k^* < l$.

If k^* is in the odd part wrt. B, then it has at most $a+b-\Delta$ black levels above Λ. If k^* is in the even part, then it has at most a black levels above Λ. Let $d' := y + 1$, $z := y + 2$, $s := l - 1$, and $z' := l - 2$. If $|S| \leq a + b - \Delta$, then the same proof as that of Claim 2.57 in case (1), yields:

Claim 2.58 *In case* (2)*, if* $|S| \leq a+b-\Delta$*, then* $\overline{Y} + |B| + |Q| \leq \overline{S} < \overline{Y'}$*. Moreover,* $y = d + 1$, $|Y| \leq a$ *and* (R2) *holds.* □

(2.1) Assume that $|S| \leq a + b - \Delta$.

(2.1.1) Assume that $l' = pit(d, l)$ **is odd, and** $f(k^*) > f(l')$.

Suppose w.l.o.g. that $z = L(l')$, and $z' = R(l')$, and let w be the neighbor of l' of smaller demand. We will apply Lemma 2.44 again. Like in case (1.2.2), $\min(\overline{S}, \overline{D'}) - \Lambda > a + |B|$. Furthermore, $x(d') \geq |D| + |B| + |Q| > |B| + a$, and $x(s) > |B| + |Q| + |H| > |B| + a$. Therefore, $\min(|Z|, |W|) > a + b - \Delta$, as well as $\min(|Z'|, |W|) > a + b - \Delta$, and Lemma 2.44 can be applied.

(2.1.2) Assume that $k = pit(d, l)$ **is even, and** $f(k^*) \geq f(k)$.

Claim 2.59 *In case* (2)*, if* $|S| \leq a+b-\Delta$*, and* $k = pit(d, l)$ *is even, then* $R(k) \neq s$.

Proof. Let w' be the one of $k - 1$ and $k + 1$, having smaller demand. Suppose that $R(k) = s$ and $L(k) = d'$ (the case $L(k) > d'$ is easy to exclude). Now $\overline{S} > \overline{W'} + |B|$, otherwise we could place S below Γ' for free. If T denotes the topmost set of contiguous black levels of k, we have $|T| \geq |Y| \geq x(d) > 2a/3$, consequently $|W'| \leq a/3$, since $|S| \leq a + |B|$. However, in this case it would be worth placing W' below Λ, a contradiction. □ *Claim 2.59*

Finally, either because $D' \supset Z'$, or due to Proposition 2.28, $|Z| > |B| + a$ or $|Z'| > |B| + a$ holds, and we can apply Lemma 2.44 to show that Ψ is not optimal.

(2.1.3) Assume that k^* **is finished below** $pit(d, l)$.

The very same argument as in the previous sentence excludes k^* being a stair or a unique loc-max above Z or Z'. We also claim that if $d' \neq s$, then $|D'| > a$, and therefore $d' \neq k^*$:

Assume that $|D'| \leq a$. Now we place both D' and S below Γ'. This operation costs $|D'|$ on the left, and at most $|S| + b$ on the right. On the other hand, $f(d')$ is reduced by $|D'| + |Y|$, and $f(s)$ is reduced by more than $|S|$. Since $|Y| \geq x(d) > b$, we obtained a better solution, a contradiction.

The next claim proves that $k^* \neq s$. The claim completes the proof of cases (2.1) and (1.2.3).

Claim 2.60 *If* $k^* = l - 1$ *or* $k^* = l + 1$*, then* Ψ *is not an optimal solution.*

Proof. We prove the claim by contradiction. The cases $k^* = l - 1$ and $k^* = l + 1$ will be discussed simultaneously. Figure 2.24 illustrates the case $\Psi^* = \Psi_a$. Let $\widetilde{\Gamma} = x(k^*) + x(l)$.

Let S denote the rung of k^* in Ψ. We start the proof with a couple of simple observations. In particular, we argue that Figure 2.24 reflects the order of finish times correctly, and at the same time we elaborate on degenerate cases, when k^* or l is a loc-max.

The property (R2) implies $\overline{D} \leq \overline{C}$, and it is trivial to show that $y' = c - 1$. Moreover $f(d') > \widetilde{\Gamma}$, otherwise neither Ψ^* nor Ψ would be optimal.

Since $f(d') > \widetilde{\Gamma}$, the node $l = k^* - 1$ cannot be a loc-max in Ψ^*. Otherwise placing all black levels of l below $\widetilde{\Gamma}$ would yield a better solution. Similarly,

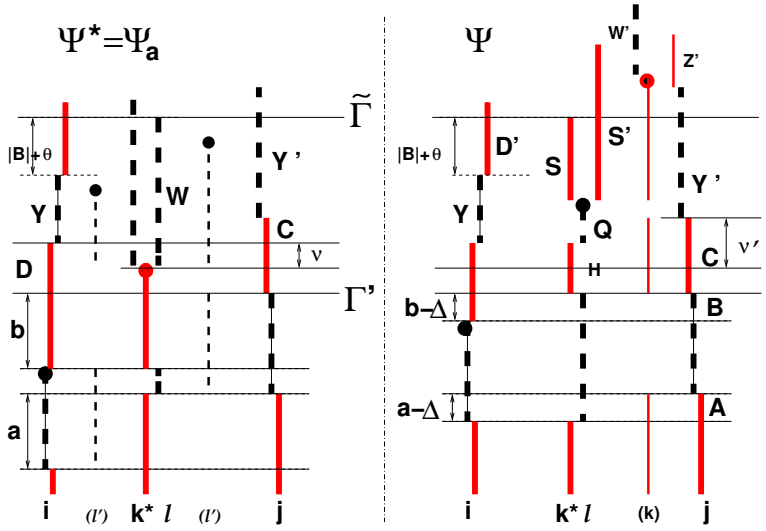

Figure 2.24: Illustration to Claim 2.60.

note that $l \neq R(l) = y'$, and by the same argument, $l = k^* + 1$ cannot be a loc-max in Ψ^*.

Assume that $l < k^* < y'$ are consecutive nodes, and k^* is a loc-max in Ψ. Then $|Y'| > a + |C|$, would hold, and it would be worth placing S below Γ'. We do not yet exclude the 'mirrored' case, i.e., when $d' = k^*$. In all other cases $l + 2$, resp. $l - 2$ is a stair or a loc-max above k^* in Ψ, and by trivial calculation it has larger demand than l.

Let $s' := l + 1 = k^* + 2$, resp. $s' := l - 1 = k^* - 2$. We claim that $|S'| > a + b - \Delta$ in Ψ : otherwise placing both S and S' below Γ' costs at most $2|S'|$, but we gain $|S| + |S'| + |Q|$. Thus, $|S'| \geq |S| + |Q| \geq |Y| + |B| + |Q| \geq x(d) + |Q| + |B| > |C| + |B|$.

In the following we will exclude that any loc-min except for k^* or l has a finish time below $\widetilde{\Gamma}$ in either of the solutions.

By Claim 2.59, in Ψ no even loc-min ends between D' and S.

In Ψ^*, let W denote the rung of l, and assume that an odd loc-min l' ends between Y and W. It is easy to see that $x(l') < x(l) + a - |B| - |Q|$, and so l' has at most $a - |Q| < |H|$ black levels above $\underline{S}$ in Ψ. Since in Ψ any stair, or loc-min above $\underline{S}$ obviously has more than $|H|$ black levels, if such an l' exists, then Ψ is not optimal.

Finally, assume that an odd loc-min l' ends between W and Y' in Ψ^*. We can exclude any possible schedule of l' in Ψ, in case Ψ is optimal. Let $\nu = \overline{D} - f_{\Psi^*}(k^*)$. We observe that $|S| = a + b - \Delta - \nu > |B| + |Q|$, so $a > \nu + |Q| > \nu'$.

Suppose that $l'' = pit(l, c)$ is odd in Ψ. If $l' < l''$, then l' has at most b black levels above $\widetilde{\Gamma}$ in the even part and 0 black levels in the odd part. If $l'' \leq l'$, then it has at most $\nu' < a \leq x(c + 1) \leq |C|$ black levels in the even part and 0 in the odd part. Both cases are straightforward to exclude by Lemma 2.45.

Suppose that $k = pit(l, c)$ is even in Ψ. If $k < l'$, then above $\widetilde{\Gamma}$, l' has ν' black levels in the odd part and $|S| + \nu'$ black levels in the even part; if $l' < k$, then it has at most b black levels in the odd part and $|S| + \nu'$ black levels in the even

part. Let w' be the neighbor of k having smaller demand. We need to show that $|W'| > |S| + \nu' = a + b - \Delta + \nu' - \nu$, and based on this it is straightforward to apply Lemma 2.45. Let $z' = R(k)$.

The only nontrivial case is when $L(k) = s'$ (see Fig. 2.24). Suppose that $|W'| \leq |S| + \nu'$. If $\overline{S'} > \overline{W'}$, then $|W'| \geq |Z'| > |Y'| + b \geq c + a + b$, a contradiction. If $\overline{S'} \leq \overline{W'}$, then $|S'| > x(l)$, by (O5), moreover $x(l) > 2a$ is a corollary of $x(d) < a$ and $\overline{C} > 3a$. Thus, $|S'| > 2a > 2b$. If $\overline{W'} \leq \overline{Z'}$, then it is worth placing down W' below $\overline{S'} - b$. If $\overline{W'} = \overline{Z'} + \mu$, then placing down costs at most $|W'| + b + \mu$, but $f(w')$ is reduced by $|Z'| + \mu \geq |Y'| + x(c+1) + \mu > |S| + \nu' + \mu$, so Ψ was not optimal.

We have shown, that on $\langle d', y' \rangle$, every node has finish time at least $\widetilde{\Gamma}$, except for k^* in Ψ^*, resp. for l in Ψ. We assume w.l.o.g that $k^* = l - 1$, and we apply Theorem 2.3 (III) on $\langle d, l \rangle$ and on $\langle k^*, c+1 \rangle$. We claim that the conditions of (III), as given in Definition 2.42, are fulfilled: the next smallest loc-mins $pit(d, l)$, and $pit(k^*, c+1)$ have finish time above $\widetilde{\Gamma}$; furthermore, both k^* and l have at most $a = \max(a, b)$ conflicts on either side, whereas $x(k^*) > a$, and $x(l) > a$.

Let α denote the number of odd-even conflicts on $\langle i, l \rangle$, and α' the number of odd-even conflicts on $\langle k^*, c+1 \rangle$ below $\widetilde{\Gamma}$.

If $\Psi^* = \Psi_a$ and $\Psi = \Psi_{a-\Delta}$, then $\alpha_{\Psi^*} - \alpha_{\Psi} = a - (|Y| - |Q|)$; and $\alpha'_{\Psi^*} - \alpha'_{\Psi} = \nu - (a - \Delta)$. By Theorem 2.3 (III) these numbers represent the total reducement in the optimum on $\langle y, k^* - 1 \rangle$ resp. on $\langle l+1, c \rangle$, in favour of Ψ. Since the sum of finish times of k^* and l grows by $\nu + |Q|$, on $\langle i+1, j-1 \rangle$ we obtain

$$\mathcal{F}(a) - \mathcal{F}(a - \Delta) = a - |Y| + |Q| + \nu - a + \Delta - \nu - |Q| = \Delta - |Y| < \Delta - b \leq 0.$$

The degenerate case when $k^* = d'$ yields exactly the same result: in Ψ^* the node y is a loc-max with a extra black levels, whereas in Ψ the node k^* is a loc-max with extra $|Y| - |Q|$ levels above $\widetilde{\Gamma}$. This corresponds to the improvement of $a - (|Y| - |Q|)$ on $\langle i, l \rangle$.

The case $\Psi^* = \Psi_{a-\Delta}$ and $\Psi = \Psi_a$ is depicted in Figure 2.25. Here the improvement from Ψ_a to $\Psi_{a-\Delta}$ in terms of odd-even conflicts sums to

$$\mathcal{F}(a) - \mathcal{F}(a - \Delta) = [|Y| - |Q| - (a - \Delta)] + [|Q| + \nu] + [a - \nu] = |Y| + \Delta > \Delta.$$

Like in the previous case, if $k^* = d'$, we end up with the same result.

In the two cases we obtained $\mathcal{F}(a) < \mathcal{F}(a - \Delta)$, resp. $\mathcal{F}(a) > \mathcal{F}(a - \Delta) + \Delta$, both contradicting to Proposition 2.37. Finally, observe that $\Psi_{a-\Delta}$ is not optimal in the first case, respectively Ψ_a is not optimal in the second case, that is in both cases we obtained the non-optimality of Ψ. □ *Claim 2.60*

(2.2) Suppose that $|S| > a + b - \Delta$.

We continue the analysis of schedule Ψ. See Figure 2.23 (2) for illustration. If $x(d) > a$, then $|Y| > a$, and we can apply Lemma 2.45 like in case (1.1).

In the rest of the proof we assume that $x(d) \leq a$, and (R2) is valid. Observe, that $x(y) > 2a$ and $x(l) > 2a$, since $\overline{C} > 3a$ and $x(d) \leq a$.

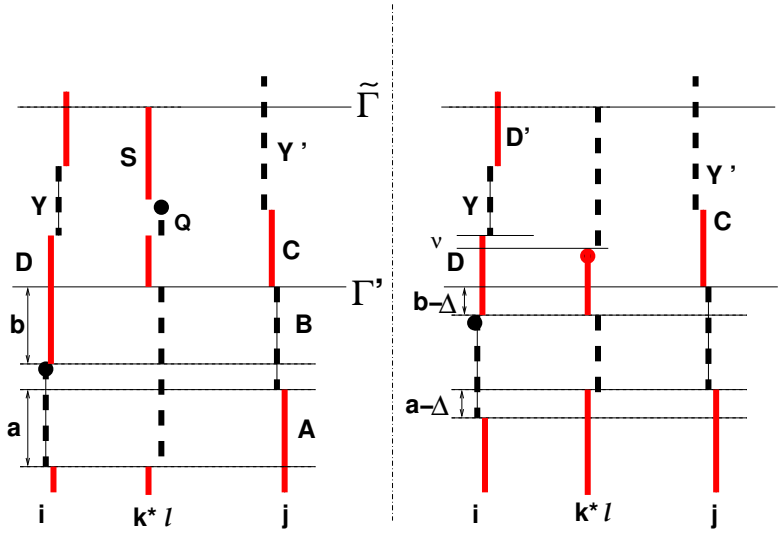

Figure 2.25: Illustration to Claim 2.60 if $\Psi^* = \Psi_{a-\Delta}$.

(2.2.1) Assume that $l' = pit(d, l)$ is odd, and $f(k^*) > f(l')$.

Suppose w.l.o.g. that $z = L(l')$, and $z' = R(l')$, and let w be the neighbor of l' of smaller demand. Observe, that $x(d') = |D'| + x(d) \geq \min(|Y|, a) + x(d) \geq |Y| + |H| + b > a + b$. Let μ be the number of levels where S has conflicts on the left; note that k^* has at most μ black levels above $\overline{S}$. Now it is straightforward to show that $|W| > \min(a + b - \Delta, \mu)$, and Lemma 2.45 can be applied to show that Ψ is not a solution.

(2.2.2) Assume that $k = pit(d, l)$ is even, and $f(k^*) \geq f(k)$.

If $R(k) < s$, then $k^* < k$, and it has at most $\mu < a$ levels above $\overline{S}$. Since either $D' \supset Z'$, or $\min(|Z||D'|) > \mu$, Lemma 2.44 excludes this case.

Suppose that $R(k) = s$. The only nontrivial case is when $L(k) = d'$. In this case it is easy to show that $|D'| > a + b - \Delta$, so $d' \neq k^*$. Let W' denote the smaller rung next to k, and let $\Pi := f_\Psi(l)$. According to Lemma 2.44, the following claim excludes any other case but $k = k^*$.

Finally, Claim 2.63 will complete the proof of (2.2.2):

Claim 2.61 *In case* (2.2.2), $\min(\overline{D'}, \overline{S}, \overline{W'}) \geq \Pi + 2a$.

Proof. Let T denote the topmost set of black levels of k. Note that $|T| > \min(a, y)$, and $x(k) \geq x(d) + |T| \geq x(d) + \min(a, y)$. If $\overline{W'} \leq \min(\overline{D'}, \overline{S})$, then $|W'| > x(k)$, and $|W'| + |T| \geq x(d) + 2|Y| \geq 3x(d) > 3 \cdot 2a/3 = 2a$.

If $\overline{D'} < \overline{W'}$, then $|D'| \geq x(y) > 2a$.

If $\overline{S} < \overline{W'}$, then placing S below $\overline{C}$ would be free on the left; therefore, either $|S| \geq \overline{C} - x(d) > 2a$, or $|S| \geq |Y'| > c + a > 2a$. □ *Claim 2.61*

(2.2.3) Assume that k^* is finished below $pit(d, l)$.

Since $|S| > a + b - \Delta$, the node k^* is trivially not a stair above S or a unique loc-max. Similarly, since $|Z| > \Delta$, k^* is not a stair above d' with at most a black nodes. We show in Claim 2.63 that $k^* = d'$ is also impossible.

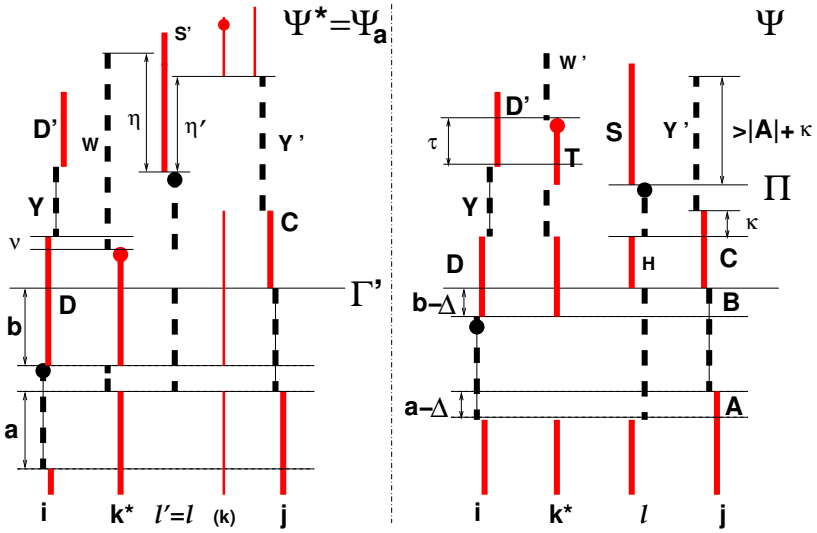

Figure 2.26: Illustration to Claims 2.62 and 2.63.

In order to finish case (2.2), it remains to exclude that $k^* = pit_\Psi(d, l)$ or $k^* = d'$. These two cases are very similar. If $k^* = pit_\Psi(d, l)$, then $|Y| \leq |T| \leq a$; if $k^* = d'$, then $|Y| \leq |D'| \leq a$, and no loc-min ends between D' and S. Let $w := w'$ in the first case; resp. $w := d' + 1$ in the second case. The definitions of κ, η, τ, etc. are easy to read out from Figure 2.26. A simple calculation yields that $\overline{Y'} - \Pi > |A| + \kappa$, otherwise Ψ is not a solution; furthermore $\overline{W} \geq \Pi + 2a$ follows from Claim 2.61, resp. it follows from $x(y) > 2a$ and $x(l) > 2a$, if $d' = k^*$. In the next claim we show that $l = pit_{\Psi^*}(k^*, c + 1)$ is the only possible position of l in Ψ^*. Finally, Claim 2.63 completes the proof of cases (2.2.2) and (2.2.3).

Claim 2.62 *In case* (2.2), *if* $\overline{W} - \Pi \geq 2a$, *and* $\overline{Y'} - \Pi > |A| + \kappa$, *then* $l = pit_{\Psi^*}(k^*, c + 1)$.

Proof. We consider the schedule Ψ^*, and exclude any other position of l, but $l = pit(k^*, c + 1)$. Assume first, that $l' = pit_{\Psi^*}(k^*, c + 1)$ is odd. We have $\overline{W} \geq \Pi + 2a > \Pi + a + b - \Delta$, and $\overline{Y'} - \Pi > |A| + \kappa$. Also, it is straightforward to show that $|S'| > 2a$, where S' is the smaller rung next to l'. If $l < l'$, then l has at most $a + b - \Delta$ black levels above Π, so at most η black levels above $\min(\overline{S'}, \overline{W})$; if $l' < l$, then l has at most $|A| + \kappa$ black levels above Π, so at most η' black levels above $\min(\overline{S'}, \overline{Y'})$. Now Lemma 2.45 yields that Ψ^* is not optimal, unless $l' = l$.

Since $f(w) > f(l)$ and $f(y') > f(l)$, l cannot be a stair. Finally, assume that $k = pit_{\Psi^*}(k^*, c + 1)$ is even. Now l has at most b black levels above $\overline{Y'}$ if $k < l$; respectively, l has less than b black levels above $\overline{W}$ if $l < k$. Now it is straightforward to apply Lemma 2.44 above $\overline{Y'}$ or above $\overline{W}$. □ *Claim 2.62*

Claim 2.63 *In case* (2.2), *if* $k^* = d'$ *or* $k^* = pit_\Psi(d, l)$, *and* $l = pit_{\Psi^*}(k^*, c + 1)$, *then* Ψ *is not an optimal solution.*

Proof. Assume first, that $\Psi^* = \Psi_a$. We compare the sum of finish times in Ψ^* and in Ψ on three subsections:

On $\langle d, k^* \rangle$ Ψ^* has smaller sum by at least $|Y|$, since placing down T reduces $f(k^*)$ by $\tau + |Y| + \nu$, and costs at most $\tau + \nu$.

On $\langle k^* + 1, l - 1 \rangle$ Ψ^* has larger sum by at most Δ. This follows from Proposition 2.37, since in Ψ^* the nodes (k^*, l) have more conflicts by Δ.

Both numbers – $|Y|$ and Δ – are the same in case $k^* = d'$.

On $\langle l, c \rangle$ the optimum is unchanged according to Theorem (II), since the number of conflicts is less by $a - \Delta - \nu$, but $f(l)$ is increased by $a - \Delta - \nu$.

Overall, we obtain $\mathcal{F}(a - \Delta) - \mathcal{F}(a) \geq |Y| - \Delta > b - \Delta \geq 0$, contradicting to Proposition 2.37, and yielding that $\Psi_{a-\Delta} = \Psi$ was not optimal.

Second, let $\Psi^* = \Psi_{a-\Delta}$. On $\langle d, k^* \rangle$, Ψ^* has smaller sum by at least $|Y|$. By Proposition 2.37, on $\langle k^* + 1, l - 1 \rangle$ Ψ^* has not larger sum than Ψ, since Ψ^* has less conflicts. Finally, on $\langle l, c \rangle$ the optimum is unchanged. The case $k^* = d'$ yields the same numbers. The difference totals to at least $\mathcal{F}(a) - \mathcal{F}(a - \Delta) \geq |Y| > \Delta$, contradicting to Proposition 2.37, yielding that $\Psi_a = \Psi$ was not optimal. □ *Claim 2.63*

□ *Lemma 2.56*

2.7.4 The procedure SELECT

Theorem 2.3 assures that modulo the two lowest levels of a minimum-tree, there is at most one possible $pit(i, r(i))$ or $top(i, r(i))$ node for fixed $(i, r(i))$. However, it is not trivial to find such a candidate node fast. We end the section with the description of procedure SELECT, which chooses at most one even and one odd such candidate node in linear time. More precisely, as we will see in Section 2.9, SELECT runs in time $\mathcal{O}(\min(\log p, n))$, this being an upper bound on the number of stairs finishing below $f(pit(i, j))$. If inequality (2.a) holds (or is required to hold), then one of these candidates can be dropped by procedure CHOOSE, which runs in constant time. It is important to note that in this form SELECT may find infeasible solutions, i.e., it will keep $\langle i, j \rangle$ sections that actually never occur as $\langle i, r(i) \rangle$ or $\langle \ell(j), j \rangle$ in any schedule. These, in principle, will be filtered out in the dynamic algorithm by finding the optimal alternatives (as well). From a practical point of view, of course one would strive to do this filtering as early as possible, by additional checking methods, as may be suggested by the proof in the next section.

We assume that i is odd and j is even. We did not include trivial checks like, e.g., $x(i) \leq x(j) + x(j - 1)$ in this pseudo code. Recall that $\mathcal{D}_{\mathcal{F}}$ is the possible domain of the number of conflicts α below Γ – and not below Γ'. The functions $f_{a_{\max}}(k)$ and $\widehat{f}_{a_{\max}}(m)$ are computed by procedures f-TIME and $\widehat{f}$-TIME of Section 2.3.1. The subroutine UPDATE is presented at the end of SELECT. This subroutine subtracts a value Δ from the maximum possible number of odd-even conflicts (below Γ) $a_{\max}$ and the same Δ from the number of even-odd conflicts $b_{\max}$.

Procedure SELECT

On input $\langle i, j\rangle$, and $x(i), ..., x(j)$, SELECT outputs the results according to one of (S1) – (S3):

(S1): $pit(i,j) = \emptyset$ and m is the only possible $top(i,j)$ node, and $\mathcal{D}_{\mathcal{F}} \subseteq [a_{\min}, a_{\max}]$.

(S2): $k = pit(i,j)$, $d = L(k)$, $c = R(k)$, and $\mathcal{D}_{\mathcal{F}} \subseteq [a_{\min}, a_{\max}]$; or
$l = pit(i,j)$, $d+1 = L(l)$, $c-1 = R(l)$, and $\mathcal{D}_{\mathcal{F}} \subseteq [a_{\min}, a'_{\max}]$.

(S3): $j = r(i)$ is impossible in any solution.

PART 1. (D and C are the lowest rungs above Γ; we check their size and adjust $a_{\max}$)

$\Gamma := \min(x(i) + x(i+1), x(j) + x(j-1))$;
$a_{\max} := \min(x(i) - 1, x(j))$; (maximum number of odd-even levels below Γ)
$b_{\max} := \Gamma - \max(x(i), x(j) + 1)$; (the same for even-odd levels)
$a_{\min} := \max(x(i) + x(j) - \Gamma, 0)$;
$b_{\min} := \max(\Gamma - (x(i) + x(j)), 0)$;
if $\min(x(i+1), x(j-1)) \le a_{\max}/2$ **then**
  $\Delta := a_{\max} - 2 \cdot \min(x(i+1), x(j-1))$;
  UPDATE$(\Delta + 1)$; (check of Proposition 2.30)
end if
if $\Gamma = x(i) + x(i+1)$ **then**
  $d := i + 2$; $c := j - 1$;
  if $c = d$ **then**
    go to PART 5.
  end if
  if $|D| < b_{\max}$ **then**
    $\Delta := b_{\max} - |D|$; UPDATE(Δ); (check of Lemma 2.29)
  end if
end if
if $\Gamma = x(j) + x(j-1)$ **then**
  $d := i + 1$; $c := j - 2$;
  if $c = d$ **then**
    go to PART 5.
  end if
  if $|C| < b_{\max}$ **then**
    $\Delta := b_{\max} - |C|$; UPDATE(Δ); (check of Lemma 2.29)
  end if
end if
$a_0 := 0$; $b_0 := 0$; (number of odd-even resp. even-odd levels above Γ)

PART 2.

(Now d and c have the same parity. W.l.o.g., here we assume $\underline{D} \le \underline{C}$, and C is high (otherwise we mirror the instance), and $d < c$ are both even (otherwise a is replaced by b, and even k by odd l, etc.). We search for a k, s.t. $f(k) \in C \cap D$.)

if $\nexists k \in \langle d+2, c-2\rangle$, s.t. k is even, and $f_{a_{\max}}(k) \le \min(\overline{D}, \overline{C})$ **then**

go to PART 3.
end if (if no $f(k)$ ends below $D \cap C$, we step on)
$k := \emptyset$;
$l := \emptyset$;
if $|C| \leq \frac{a_{\max}+a_0}{2}$ **then**
$\Delta := a_{\max} - (2|C| - a_0)$; (check of Proposition 2.53)
UPDATE$(\Delta + 1)$;
end if

(new test with the decreased $a_{\max}$; if now the $f_{a_{\max}}(k) > \min(\overline{D}, \overline{C})$, for all even k, then there is no even solution; we search for odd $pit(i,j)$ in PART 4.)

if $\nexists k \in \langle d+2, c-2\rangle$, s.t. k is even and $f_{a_{\max}}(k) \leq \min(\overline{D}, \overline{C})$ **then**
go to PART 4.
end if
$a := a_{\max} + a_0$ (we select at most one even k by Lemma 2.54)
if $k_1, k_2, \ldots, k_\xi$ all fulfil $f_{a_{\max}}(k) \leq \min(\overline{D}, \overline{C})$ **then**
let $k \in \{k_1, \ldots k_\xi\}$ s.t. $x(k)$ is minimum.
if $\exists k_\mu > k$, s.t. $x(k_\mu) < x(k) + a/2$ **then**
$k := \max\{k_\mu > k \mid x(k_\mu) < x(k) + a/2\}$;
if $\exists k_\nu > k$, s.t. $x(k_\nu) < x(k) + a/2$ **then**
$k := \emptyset$
go to PART 4.
end if
end if
if $x(d) > a$ **then**
output (S2). ((2.a) holds and k is the only possible $pit(i,j)$)
else
go to PART 4.
end if
end if

PART 3. (we step to the next rung on the left or on the right)

if $\overline{D} \leq \overline{C}$ **then**
$d := d + 1$;
if $c = d$ **then**
go to PART 5.
end if
if d is even **then**
UPDATE $a_{\max}$ so that $|D| \geq a_{\max} + a_0$ holds;
else
UPDATE $b_{\max}$ so that $|D| \geq b_{\max} + b_0$; (check of Lemma 2.29)
end if
else
$c := c - 1$;
if $c = d$ **then**

go to PART 5.
end if
check Lemma 2.29 for $|C|$;
end if
$\delta := \min(\overline{D}, \overline{C}) - \max(\underline{D}, \underline{C})$
if (d, c) is odd-even **then**
$a_0 := a_0 + \delta$; go to PART 3.
end if (number of odd-even conflicts above Γ increased, we step to the next rung)
if (d, c) is even-odd **then**
$b_0 := b_0 + \delta$; go to PART 3.
end if (number of even-odd conflicts above Γ increased)
go to PART 2. (d and c are of the same parity)

PART 4. (assuming that k was even in PART 2. we search for an odd l to be $pit(i, j)$)
$d' := d + 1$; $c' := c - 1$;
if $D' \cap C' = \emptyset$; **then**
output (S2) if k was selected, or (S3) otherwise. (there is no odd candidate l)
end if
(We assume that (d', c') is odd-odd, and $\underline{D'} \leq \underline{C'}$)
$a'_{\max} := a_{\max}$; (we record separate $a_{\max}$ and $b_{\max}$ values for $l = pit(i, j)$)
$b'_{\max} := b_{\max}$;
$a'_0 := a_0 + (\underline{C'} - \underline{D'})$;
$b' := b'_{\max} + b_0$;
$a' := a'_{\max} + a'_0$;
check Lemma 2.29 for $|D'|$ and $|C'|$ by UPDATE-ing $a'_{\max}$ and $b'_{\max}$ if needed;
select at most one odd $l \in \langle d' + 2, c' - 2 \rangle$, s.t. $f_{a'_{\max}}(l) \leq \min(\overline{D'}, \overline{C'})$ analogously to PART 2.;
if the search fails, set $l := \emptyset$;
if $k \neq \emptyset$ and $l \neq \emptyset$ **then**
output (S2); (run CHOOSE(k, l);)
else
if $k \neq \emptyset$ or $l \neq \emptyset$ **then**
output (S2).
else
output (S3).
end if
end if

PART 5. (now $pit(i, j) = \emptyset$; we select one possible $m = top(i, j)$)
$\mathcal{F} := \infty$;
for $\bar{m} := d - 3$ to $d + 3$ **do**
$\mathcal{F}_{\bar{m}} := \sum_{h:=i+1}^{\bar{m}-1}(x(h-1) + x(h)) + \widehat{f}_{a_{\max}}(\bar{m}) + \sum_{h:=j-1}^{\bar{m}+1}(x(h+1) + x(h))$;
if $\mathcal{F}_{\bar{m}} < \mathcal{F}$ **then**
$m := \bar{m}$; $\mathcal{F} := \mathcal{F}_{\bar{m}}$;

end if
end for
output (S1);

Procedure UPDATE(Δ) (subtracts Δ from $a_{\max}$ and $b_{\max}$ in parallel.)

if $a_{\max} - \Delta \geq a_{\min}$ and $b_{\max} - \Delta \geq b_{\min}$ **then**
let $a_{\max} := a_{\max} - \Delta$;
let $b_{\max} := b_{\max} - \Delta$;
else
if UPDATE(Δ) was called from PART 2. **then**
go to PART 4.
else
if UPDATE(Δ) was called from PART 4. and $k \neq \emptyset$ **then**
$l := \emptyset$;
output (S2).
else
output (S3).
end if
end if
end if

We complete SELECT by procedure CHOOSE below, which sets the value $a_{\max}$ so that (R1) or (R2) holds, meaning that i or j can be on any high floor of the minimum-tree. For this restricted domain CHOOSE chooses one of k or l, if both were output by SELECT. We can assume that the nodes $pit(d-1,k)$, $pit(k,c+1)$, $pit(d,l)$, $pit(l,c)$, etc. were found by SELECT earlier by the dynamic programming algorithm.

The subroutine ADJUST trims the domain $[a_{\min}, a_{\max}]$ according to the domains of, e.g., $f(pit(d-1,k))$ and $f(pit(k,d+1))$, and so that, e.g., $f(k)$ is sure to fall in $C \cap D$. The pseudocode of ADJUST is given in Section 2.9.

The output of CHOOSE(k,l) is the interval domain $[a_{\min}, a_{\max}]$ for the possible number of odd-even levels below Γ, and exactly one of $\{k, l, \emptyset\}$ to be the node $pit(i,j)$.

Procedure CHOOSE(k,l)
PART 1. (check if $k = pit(i,j)$ is possible)
$a_{orig} := a_{\max}$; (we will set $a_{\max}$ to obey (2.a); we save the original values)
$b_{orig} := b_{\max}$;
$R_2 := \min(3x(d)/2,\ x(c+1),\ x(c) - x(c+2)))$;
$a := a_{\max} + a_0$;
(reduce $a_{\max}$ so that (2.a) holds)
if $d \neq i+1$ and $a \geq x(d)$ **then**
$\Delta := a - x(d)$;
UPDATE($\Delta + 1$);

end if
if $d = i + 1$ and $a \geq \max(x(d), R_2)$ **then**
$\Delta := a - \max(x(d), R_2)$;
UPDATE$(\Delta + 1)$;
end if
$a'_{\min} := a_{\min}$; $a'_{\max} := a_{\max}$; (values for the case $l = pit(i, j)$)
if $f_{a_{\max}}(k) > \min(\overline{D}, \overline{C})$ **then**
go to PART 2. (with reduced $a_{\max}$, $k = pit(i, j)$ is impossible)
end if
if $a < x(d)$ **then**
output k and $[a_{\min}, a_{\max}]$. ((R1) excludes $l = pit(i, j)$)
end if

(we check the solutions on both sides of k; for simplicity, we write $pit(d-1, k)$ instead of ($pit(d-1, k)$ or $top(d-1, k)$), etc.)

ADJUST$(a_{\min}, a_{\max})$ according to $pit(d-1, k)$ and $pit(k, c+1)$;
if $pit(d-1, k)$ and $pit(k, c+1)$ exist, and $a_{\min} \leq a_{\max}$ **then**
if $k = l - 1$ or $k = l + 1$ **then**
go to PART 3.
else
output k and $[a_{\min}, a_{\max}]$.
(k and l are not neighbors and k is a possible solution, so l is excluded)
end if
else
go to PART 2.
end if

PART 2. ($k = pit(i, j)$ is impossible; we check if $l = pit(i, j)$ is possible)
ADJUST$(a'_{\min}, a'_{\max})$ according to $pit(d, l)$ and $pit(l, c)$;
if $pit(d, l)$ and $pit(l, c)$ exist, and $a'_{\min} \leq a'_{\max}$ **then**
output l and $[a'_{\min}, a'_{\max}]$.
else
output $\emptyset$.
end if

PART 3. ($k = l - 1$ or $k = l + 1$)

(if there are different solutions for k and l on either side of k, then the non-optimal solution is rejected like in part 2 of SELECT)

ADJUST$(a'_{\min}, a'_{\max})$ according to $pit(d, l)$ and $pit(l, c)$;
if $pit(d, l)$ and $pit(l, c)$ exist, and $a'_{\min} \leq a'_{\max}$ **then**
if $pit(d, l)$ ends in lower rungs than $pit(d-1, k)$ **then**
output l and $[a'_{\min}, a'_{\max}]$.
end if (the lower $pit(d, l)$ is better since (R1) holds on $\langle d, l \rangle$)
if $pit(d, l) \neq pit(d-1, k)$ end in the same rungs **then**

choose one of $pit(d, l)$ or $pit(d-1, k)$ according to Lemma 2.54.
if $pit(d, l)$ was chosen **then**
output l;
end if
if $pit(d-1, k)$ was chosen **then**
output k;
end if
end if
do the same with $pit(l, c)$ and $pit(k, c+1)$
end if
output k and $[a_{\min}, a_{\max}]$. (if the solutions on the two sides are the same for k and l, then k is the proper candidate by Claim 2.60)

Theorem 2.3 implies the following corollary. Here we provide a brief verification, referring to the proof in Section 2.7.3.

Corollary 2.64 *For given $\langle i, j\rangle$, and $x(i), ..., x(j)$, procedure* SELECT *finds all potential $pit(i, j)$ or $top(i, j)$ node that might occur in an optimal schedule where $j = r(i)$. Furthermore,* CHOOSE *chooses one of these candidates correctly, given that i (resp. j) is on at least the 3rd floor of the minimum-tree.*

Proof. Suppose that in PART 2, SELECT found the nodes $d < k < c$, so that $f_{a_{\max}}(k) \leq \min(\overline{D}, \overline{C})$. We assume w.l.o.g. that d, k, and c are even. First we argue that there is no other even node k' for which $k' = pit(i, j)$ is possible.

We claim that at the end of PART 2, the inequalities of Proposition 2.48 hold for the nodes c, d, and for the maximum possible number of conflicts $a = a_{\max} + a_0$ resp. $b = b_{\max} + b_0$. This follows from the fact, that as SELECT proceeds upwards along the stairs on both sides, in PARTS 1 – 3 Proposition 2.30 is checked for the lowest rungs, and Lemma 2.29 is checked for each consecutive rung. Moreover, we do not lose candidates due to this check, since the proposition and the lemma must hold for $\langle i, j\rangle$ as well. Now by Lemmas 2.50, 2.51, and 2.52 there is no even $pit(i, j)$ with finish time above $\min(\overline{D}, \overline{C})$, or a unique loc-max. Observe that the lemmas excluded such a solution purely based on $f_{a_{\max}}(k) \leq \min(\overline{D}, \overline{C})$, and Proposition 2.48, whereas the actual existence of Ψ^* was not needed in these proofs.

After this, the procedure selects at most one even candidate k that could end in $D \cap C$. If such a $pit(i, j)$ node exists, then $|C| > a/2$ must hold by Proposition 2.53, so the procedure reduces $a_{\max}$ to obey this. If there remain even nodes that would finish in $D \cap C$, then SELECT chooses one of them, according to Lemma 2.54.

In the proof of Lemma 2.56, we could exclude an odd $l = pit(i, j)$ only using $f_{a_{\max}}(k) \leq \min(\overline{D}, \overline{C})$, if either $x(d) > a$, or $L(l) \neq d + 1$ or $R(l) \neq c - 1$. So, in these cases we output only k. Otherwise in PART 4, at most one odd l is selected, as suggested by Lemma 2.54.

We have to record both of these candidate nodes k and l for the case when i or j is on first or second floor of a minimum-tree. However, on higher floors (R1) or

(R2) must hold, and by Lemma 2.56, we need to keep only one of k or l. Procedure CHOOSE further reduces $a_{\max}$ to make sure that (R1) or (R2) is valid.

Except for the Claims 2.60 and 2.63, the proof of Lemma 2.56 exploits only $f_{a_{\max}}(k) \leq \min(\overline{D}, \overline{C})$, and excludes $l = pit(i,j)$. When the setting corresponds to one of the two claims, the choice between k or l becomes less obvious:

Let first $k = l-1$ or $k = l+1$. In the proof of Claim 2.60 we use that on $\langle d, k\rangle$ a solution exists and it is the same as the solution on $\langle d, l\rangle$. The same holds for $\langle k, c\rangle$ and $\langle l, c\rangle$. Notice that if in PART 3 of CHOOSE, k or l is rejected, then either on one side a solution does not exist, or is not optimal, because we found a better solution; or the solutions are the same on both sides, and Claim 2.60 implies that $l = pit(i,j)$ cannot be optimal.

Second, assume that $k = d'$ or $k = pit(d,l)$. Whenever $f_{a_{\max}}(k) \in D \cap C$, the candidate nodes $pit(d-1,k)$ (or $top(d-1,k)$) and $pit(k,c+1)$ (or $top(k,c+1)$) exist, and the adjusted $[a_{\min}, a_{\max}]$ domain is nonempty, then node $k = pit(i,j)$ is chosen. Obviously, if any of these conditions fails, then $k = pit(i,j)$ can be excluded.

We show that otherwise l can be rejected. If the premises of Claim 2.62 do not hold, then $l = pit(i,j)$ can be excluded by Lemma 2.56. If the premises hold, then $f_{a_{\max}}(l) \in W \cap Y'$, and therefore either $l = pit(k,c+1)$ or some other $l' = pit(k,c+1)$ is selected on $\langle k, c+1\rangle$, s. t. $f(l') \in W' \cap Y'$. If $l = pit(k,c+1)$, then l is excluded by Claim 2.63. If $l' = pit(k,c+1)$, then similarly to Claim 2.62, it can be shown that such an l' has too small demand to be scheduled in an optimal Ψ, so Ψ is not a solution, and $l = pit(i,j)$ is impossible again.

Finally, we turn to PART 5 of SELECT. If $i < i+1 < \ldots < m < \ldots j-1 < j$, and m is a unique loc-max, then $x(m-1) > x(m-3) > x(m-5)\ldots$; resp. $x(m-2) > x(m-4) > \ldots$; etc. If we end up with $c = d$, then depending on the order of demands, at most 4 potential nodes m inside $\langle d-3, d+3\rangle$ remain. For all these nodes, it is trivial to calculate the finish times of stairs, and $\widehat{f_{a_0}}(m)$. We select m yielding the smallest sum of finish times. □

2.8 The increase of finish times

Theorem 2.3 allows us to design a dynamic programming algorithm that runs in polynomial – $\mathcal{O}(n^4)$ – time: We could apply the SELECT procedure for every (i,j) pair. Within SELECT this would involve testing for every $k \in \langle i, j\rangle$, each test taking $\mathcal{O}(n)$ time, that is, the number of stairs finishing below $f(k)$. Using dynamic programming, for the triple $i < k < j$ we can calculate the optimum function $\mathcal{F}(\alpha)$ on $\langle i, j\rangle$ from the optimum functions on $\langle i, k\rangle$ and on $\langle k, j\rangle$, in constant time.

Based on the results of this section, we can further reduce the time bound, and obtain an algorithm with $\mathcal{O}(\min(n^2, n\log p))$ running time, where $p = \max_i x(i)$. The lemmas below are quite intuitive: they reflect the fact that within the minimum-tree of any block the finish times exhibit exponential growth.

As a consequence, for a given node i, there are only a constant number of j as possible $r(i)$ nodes. Our current bound for this number is not small (2721); however, this fact is not intrinsic to the problem, but is due to our wish to possibly provide a

short and simple proof rather than trying to optimize the constant. We conjecture that the 'actual' constant is a small number, but seemingly the better bound we aim at, the more lengthy and tedious proof it requires.

For the same reason, for given i and j, there are only a constant number of potential k (resp. l), found by SELECT.

Another consequence is that there are $\mathcal{O}(\min(n, \log p))$ preemptions in the schedule of any node, in other words, the schedule of any node can be described using $\mathcal{O}(\min(n, \log p))$ numbers.

The next proposition follows directly from Corollary 2.25. Recall from Corollary 2.25, that if k has loc-min neighbors on both sides, then $\langle k-1, k+1\rangle$ is a trivial block.

Proposition 2.65 *Let Φ be a solution on $\langle 1, n\rangle$, and $1 \leq k \leq n$. Unless k has loc-min neighbors on both sides, $f(k) \leq 4x(k)$. Moreover, if k is not a second stair, then even $f(k) \leq 3x(k)$ holds.*

Definition 2.66 *If i is a non-compact loc-min in a solution, then let $\widetilde{f}(i) := \min(f(L(i)), f(i-1), f(i+1), f(R(i)))$.*

Lemma 2.67 *If the loc-min i_0 is a ξth descendant of loc-min i_ξ in a minimum-tree, then $\widetilde{f}(i_0) \geq (\frac{5}{4})^{[\xi/2]}\widetilde{f}(i_\xi)$.*

Proof. We prove the lemma by induction on ξ. The inequality trivially holds if $\xi = 0$. For $\xi = 1$, we have to observe that due to parity constraints, $\widetilde{f}(i_0) \geq f(i_0) > \widetilde{f}(i_1)$, whenever i_1 is the parent of i_0.

Suppose that $\xi \geq 2$, let i_1 be the parent and i_2 be the grandparent of i_0 in the minimum-tree, and $f_0 := \widetilde{f}(i_0)$, $f_1 := \widetilde{f}(i_1)$, resp. $f_2 := \widetilde{f}(i_2)$. We show that $f_0 \geq (\frac{5}{4})f_2$, and this will prove the lemma. Let $w := L(i_0)$, and $w' := R(i_0)$. We assume w.l.o.g. that $f(w) \leq f(w')$. Let $y := i-1$ if $x(i-1) \leq x(i+1)$, and let $y := i+1$ otherwise. We distinguish four cases as shown in Figure 2.27. The lower bound on the demand $x()$ is implied by Proposition 2.65.

(1) Assume that $f_0 = f(y)$.

Now by (O1) and (O4), $|Y| \geq x(i_0)$, otherwise it would be worth placing Y below $f(i_0)$. Since $x(i_0) \geq f(i_0)/4$, we obtain $f_0 = f(i_0) + |Y| \geq (5/4)f(i_0) > (5/4)f_1 > (5/4)f_2$.

(2) Assume that $f_0 = f(w)$, where w is a high stair; moreover, $L(i_1) \neq w$.

Now using (O2) we get $|W| \geq x(w-1) \geq f(w-1)/4$. If $i_1 < i_0$, then $i_1 < w-1$, so $f_1 \leq f(i_1+1) \leq f(w-1)$; if $i_0 < i_1$, then by assumption $L(i_1) \leq w-1$, so $f_1 \leq f(L(i_1)) \leq f(w-1)$. We obtained $f_0 = f(w) \geq (5/4)f(w-1) \geq (5/4)f_1$.

If $f(w'+2) > f(w-1)$, then we can also argue with $f(w'+1) \geq (5/4)f(w'+2) > (5/4)f(w-1)$. We will use this observation in Lemma 2.72 and in Theorem 2.5.

(3) Assume that $f_0 = f(w)$, and w is a second stair; moreover, $L(i_1) \neq w$.

Note that $L(i_1) \neq w$ implies that $i_1 = w-1$. Let $v := L(i_1)$ and $v' := R(i_1)$. If $W \subset V$ or $f(w) \geq f(w-2)$, then $|W| \geq f(i_1)/4$ by (O1), (O2), and (O5). Otherwise

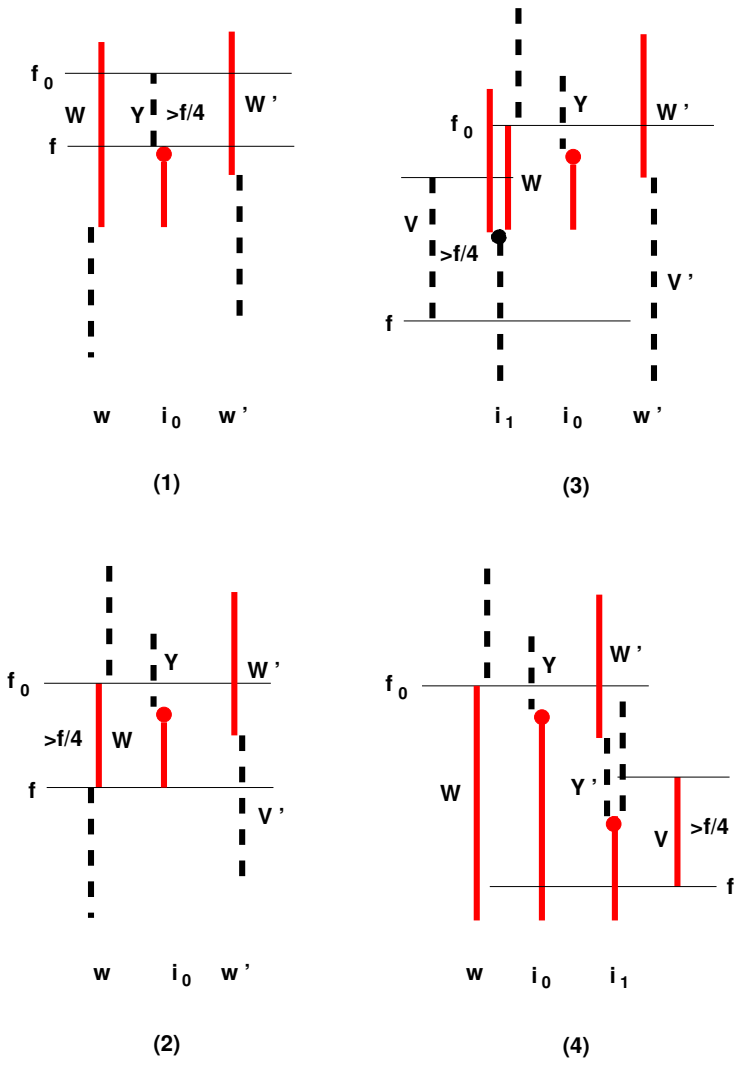

Figure 2.27: Illustration to Lemma 2.67 and Theorem 2.5.

$f(w-2) > f(w) > f(v)$. Assume that $\underline{V} \geq \underline{V'}$. In this case V is a high rung, and either $i_2 \leq v-2$ or $L(i_2) \leq v-1$. Like for W in case (2), we obtain $|V| \geq f(v-1)/4$, and $f(v-1) \geq f_2$, consequently, $f_0 = f(w) > f(v) = f(v-1) + |V| \geq (5/4)f_2$. In case $\underline{V} \leq \underline{V'}$, the proof is symmetric.

(4) Assume that $f_0 = f(w)$, and $L(i_1) = w$.

Let $y' := i_1 - 1$ and $v := R(i_1)$. If $Y' \subset V$ or $f(y') \geq f(y'+2)$, then $|Y'| \geq f(i_1)/4$ by (O1), (O2), and (O5). Otherwise $f(y'+2) > f(y') > f(v)$. Assume first that $\underline{V} \geq \underline{W}$. Now V is a high rung, and either $i_2 \geq v+2$ or $R(i_2) \geq v+1$. We obtain $|V| \geq f(v+1)/4$, and $f(v+1) \geq f_2$, and therefore $f(v) \geq (5/4)f_2$. Second, assume that $\underline{V} < \underline{W}$. Now W is a high rung, and either $i_2 \leq w-2$, or $L(i_2) \leq w-1$. This implies $|W| \geq f(w-1)/4$, and $f(w-1) \geq f_2$, so $f(w) \geq (5/4)f_2$. □

Lemma 2.68 *If U, V, and W are consecutive high rungs of stair-ups, s.t. no loc-min ends in any of these rungs, then $f(w) \geq (5/4)f(u-1)$. A symmetric statement*

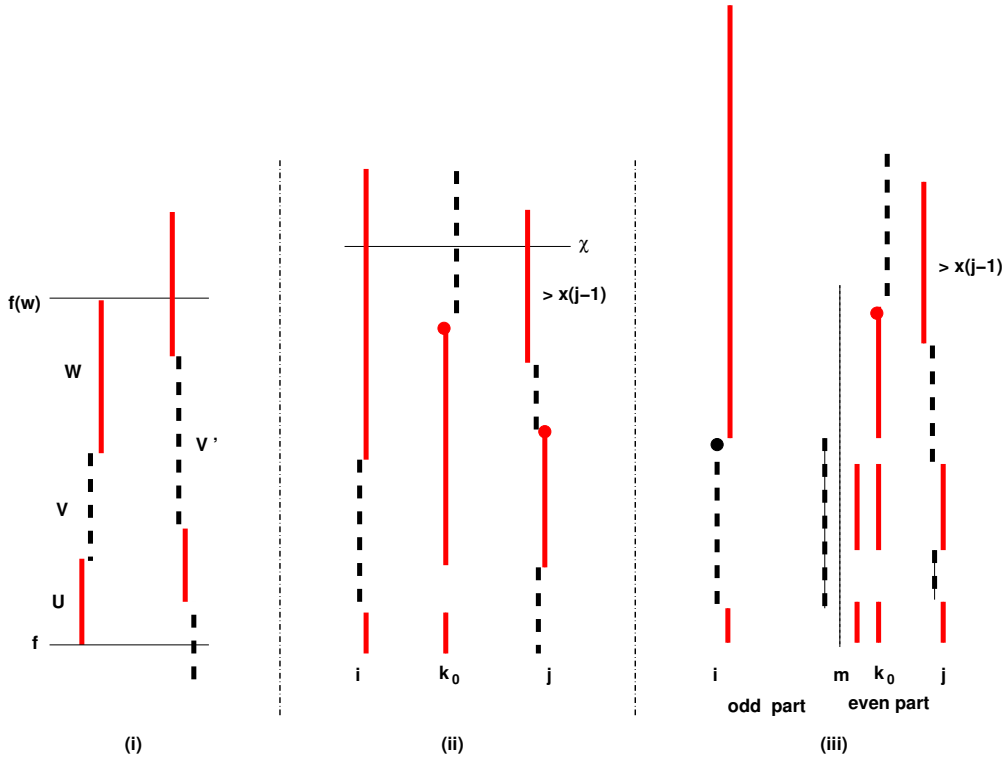

Figure 2.28: (i) Lemma 2.68; (ii) Lemma 2.70; and (iii) Lemma 2.71.

holds for stair-downs.

Proof. Let $f = f(u-1)$, and assume that v is odd (see Fig 2.28 (i)). Consider the first loc-min to the right of w, that has finish time below $f(w)$. Since no loc-mins end in any of the rungs U, V, and W, the finish time of this loc-min is less than $f = f(u-1)$. Moreover, some stair-downs above this loc-min must have rungs intersecting the level set $[f, f(w)]$. Concerning these rungs we distinguish the following cases:

If W is the subset of an odd rung, then, by (O1), $|W| \geq x(v) \geq f/4$.

Similarly, if U is the subset of an odd rung, then $|U| \geq x(u-1) \geq f/4$; resp. if V is the subset of an even rung, then $|V| \geq x(u) \geq f/4$.

If an even rung $Q \subset V$, then $|V| \geq |Q| \geq x(q+1) \geq f/4$.

In the remaining case, there is one and only one odd rung V' intersecting V. Moreover, $V' \subset U \cup V \cup W$, so that the conditions of Proposition 2.28 hold, and we have $\max(|V|, |V'|) \geq \min(x(u), x(v'+1)) \geq f/4$. In any of the cases above we obtain $f(w) \geq f + f/4$. □

The next lemmas are useful when the algorithm searches for potential $(i,j) = (i, r(i))$ or $(i,j) = (\ell(j), j)$ pairs. Assume that the node i is fixed, and we search for a possible $j > i$; the case $j < i$ is symmetric. Lemmas 2.70 – 2.72 imply that for every i, it suffices to test for a constant number of j. We summarize our results in Theorem 2.4. Procedure PAIRS selects the (i,j) pairs as described in the proof of the theorem. For illustration see Figure 2.28.

Proposition 2.69 *If i is a non-compact loc-min, then $x(r(i)) \leq 2x(i)$ and $x(\ell(i)) \leq 2x(i)$.*

Proof. Recall that i and $r(i)$ are of opposite parity. If $x(r(i)) > 2x(i)$, then obviously $f(i) > f(r(i)) \geq x(r(i)) > 2x(i)$. Suppose that $f(i) = 2x(i) + q$. Then i has at most $q-1$ black levels, where it is *not* conflicting with $r(i)$, i.e., where $(i, r(i))$ is black-white. Therefore, if we make i compact, that costs at most $q-1$ on the right and at most $x(i)$ on the left, whereas $f(i)$ is decreased by $x(i)+q$, so we obtain a better solution, a contradiction. □

Lemma 2.70 *Assume that i is odd, j is even, and either $j = r(i)$ or $i = \ell(j)$ in some solution Φ. If $x(j)+x(j-1) \leq x(i)+x(i+1)$, then there exist at most 2^9+1 even nodes $k \in \langle i, j\rangle$ so that $x(k)+x(k-1) \leq x(i)+x(i+1)$ and $x(k)+x(k-1) \leq (4/3)\cdot(x(j)+x(j-1))$.*

Proof. Let $\chi := \min(x(i)+x(i+1), (4/3)\cdot(x(j)+x(j-1)))$ (see Fig 2.28 (ii)). Consider an even k, such that $x(k)+x(k-1) \leq \chi$. If $pit(i,j)$ is odd, then $f(k) > \chi$ is straightforward. Let $k_0 := pit(i,j)$ be even. Using (O1)–(O5), it is easy to show that $\widetilde{f}(k_0) > \chi$, and except for k_0, all nodes inside $\langle i, j-1\rangle$ have finish time above χ. Assume that $k \neq k_0$. Then $f(k) > \chi$ implying that $(k-1, k)$ is white-white on some levels below $f(k)$. Therefore, either k or $k-1$ is a loc-max node by Corollary 2.16.

Suppose that $x(k) \leq x(k-1)$ (the proof is analogous if $x(k-1) \leq x(k)$). Now $x(k) \leq \chi/2$, and $f(k) \leq 4x(k) \leq 2\chi$ by Proposition 2.65. Let T' denote the connected subgraph of the minimum-tree – a tree –, rooted at k_0, and including only loc-mins of finish time at most 2χ. Since $\widetilde{f}(k_0) > \chi$, and $((\frac{5}{4})^{\frac{1}{2}})^8 \cdot \widetilde{f}(k_0) > 2\chi$, Lemma 2.67 implies that T' has at most 2^8 leaves. Recall that k or $k-1$ is a loc-max, so k can be associated with exactly one such leaf (i.e., with a branch of the minimum-tree leading to $k-1$ or k). Moreover, $f(k) \leq 2\chi$ (or $f(k-1) \leq 2\chi$) implies that we associated at most two such pairs $(k-1, k)$ with each leaf of the subgraph. As a consequence, together with k_0 there are at most $2 \cdot 2^8 + 1$ nodes k, so that $x(k)+x(k-1) \leq \chi$. □

Lemma 2.71 *Assume again that i is odd, j is even, in Φ either $j = r(i)$ or $i = \ell(j)$, moreover $x(j) \leq 2x(i)$, and $x(j)+x(j-1) \leq x(i)+x(i+1)$. Let $\langle i, m\rangle$ be the odd part and $\langle m, j\rangle$ be the even part of $\langle i, j\rangle$. If $x(i+1) > 3.2 \cdot x(i)$, then in the odd part there are at most 2^7 even nodes $k \in \langle i, j\rangle$ so that $x(k) \leq 2x(i)$ and $x(k)+x(k-1) \leq x(i)+x(i+1)$; in the even part there are at most 2 even nodes $k \in \langle i, j\rangle$ for which $x(k) \leq \frac{10}{9}x(j)$.*

Proof. Assume first that $k \in \langle m, j\rangle$ is in the even part, k is even, and $x(k) \leq \frac{10}{9}\cdot x(j)$. Notice that k has $x(j)$ black levels below $f(j)$, and at most $x(j)/9$ black levels above $f(j)$ (see also Figure 2.16). Let $k_0 := pit(i,j)$ (even or odd). It is easy to show – using (O1)–(O5) – that $\widetilde{f}(k_0) \geq f(j) + x(j)/3$. Furthermore, if $R(k_0) < j-2$, then $f(j-3) \geq f(j) + x(j)/3$. Unless $k = k_0$ or $k = j-2$, it would be worth placing the black levels of k below $f(j-1)$. This proves the second statement of the lemma.

In order to prove the first statement, we need to observe that all nodes in the odd part $\langle i, m\rangle$ have finish time above $f(i{+}1) = x(i){+}x(i{+}1) > x(i){+}3.2{\cdot}x(i) = 4.2{\cdot}x(i)$. Since $x(k) \leq 2x(i)$, we have $f(k) \leq 8 \cdot x(i)$. The rest of the proof follows the same lines as the previous proof:

Let l_1 be the first loc-min in the odd part as shown in Figure 2.16. We consider the subgraph T' of the minimum-tree rooted at l_1, having loc-mins with finish time at most $8x(i)$. Since $((\frac{5}{4})^{\frac{1}{2}})^6 > \frac{8}{4.2}$, there are at most 2^6 leaves in this subgraph. Again, $x(k-1) + x(k) \leq x(i) + x(i+1) \leq f(l_1)$, so that one of $k-1$ or k is a loc-max. We obtain that there are not more than $2 \cdot 2^6$ such k. □

Lemma 2.72 *Assume that i is odd, j is even, and $j = r(i)$ in some solution. There are less than 155 even nodes $k \in \langle i, j\rangle$, so that $x(k) \leq x(i)/2$.*

Proof. Let $k \in \langle i, j\rangle$ be even, and $x(k) \leq x(i)/2$. According to Proposition 2.65, $f(k) \leq 4x(k) \leq 2x(i)$. Thus we need to upper bound the number of all even nodes having finish time in the interval $[\widetilde{f}(i), 2x(i)]$. If $k_0 = pit(i, j)$ (even or odd), then $\widetilde{f}(k_0) \geq \frac{4}{3}x(i)$ can be easily proven using (O1)–(O5). Since $(\frac{5}{4})^2 > 2/\frac{4}{3} = 3/2$, the minimum-tree has at most 4 levels between $\frac{4}{3}x(i)$ and $2x(i)$ by Lemma 2.67.

Consider the subtree T_i rooted at i, of the minimum-tree. We will modify T_i, and associate each potential node k with *at least* one vertex of the modified tree. For simplicity, we do not treat loc-max nodes explicitly, since they can be regarded as (leaf) loc-mins in the trees.

1. Omit all vertices of T_i that represent loc-mins to the left of i.

2. Omit all vertices of T_i that represent loc-mins with finish time more than $2x(i)$; let T_i' denote the resulting tree.

3. For every loc-min $h \in T_i'$ associate the following nodes with the vertex of h in T_i' : $L(h)-2$, $L(h)-1$, $L(h)$, $h-1$, $h, h+1, R(h)$, $R(h)+1$, $R(h)+2$. There are at most 5 even nodes among these nodes.

4. The remaining (not yet associated) nodes are high stairs. We insert new pairs of vertices into T_i', that correspond to consecutive triples of high stairs as follows. Let h_0 be the child of h_1 in T_i', and suppose that $h_0 < h_1$. If $\kappa = \max(|L(h_0) - L(h_1)|, |R(h_0) - (h_1 - 1)|)$, then subdivide the edge (h_0, h_1) with $2 \cdot \lfloor(\kappa - 1)/3\rfloor$ new vertices. We obtain the tree T_i''. Now we can associate all the not-yet-associated high stair-ups $L(h_1) + 1, \ldots, L(h_0) - 3$, respectively the high stair-downs $R(h_0) + 3, \ldots, h_1 - 1$ with one of the new vertices so that at most 2 even stairs are associated with each vertex.

Observe that T_i'', is still a binary tree, and at least each pair of descendant vertices accounts for a growth factor of at least 5/4 in the finish times, by Lemmas 2.67 and 2.68. Since finish times of loc-mins are in the interval $[\frac{4}{3}x(i), 2x(i)]$, the paths leading from k_0 to a leaf of T_i'' have length at most 4. Disregarding i, the modified tree has at most $2^5 - 1 = 31$ vertices, and not more than 5 even nodes in $\langle i, j\rangle$ were assigned to each vertex. We obtain that at most $5 \cdot 31 = 155$ even nodes have finish time in the given interval. □

Theorem 2.4 *For any fixed odd node $i \in \langle 1, n\rangle$, it suffices to test for at most* 2721 *even nodes $j > i$ in order to find all potential $(i, j) = (i, r(i))$ and $(i, j) = (\ell(j), j)$ pairs for that either $x(j) + x(j-1) \le x(i) + x(i+1)$ or $x(j) \le x(i)/2$ holds. A symmetric statement is true for $j < i$ nodes.*

Proof. If we assume that $x(j) < x(i)/2$, then only the case $(i, j) = (i, r(i))$ is possible by Proposition 2.69. Lemma 2.72 implies that we need to test for the first $155 + 1 = 156$ even nodes $j > i$ of demand $x(j) < x(i)/2$.

After this, we only consider nodes $j > i$, for which the following hold:

$$x(j) + x(j-1) \le x(i) + x(i+1) \qquad \text{and} \qquad x(i)/2 \le x(j) \le 2x(i). \tag{2.b}$$

If $x(i+1) > 3.2 \cdot x(i)$, according to Lemma 2.71 we have to test for the first $2 \cdot 2^6$ even nodes $j > i$ for which (2.b) holds, *and* after the 2^7th tested node, independently for each $\nu = 1, 2, \ldots, 14$, the following 3 nodes j' of demand $x(j') \in [(\frac{10}{9})^{(\nu-1)} \cdot \frac{x(i)}{2}, (\frac{10}{9})^{\nu} \cdot \frac{x(i)}{2}]$ again with the property (2.b). (For $\nu = 15$ we would already get $x(j') > (\frac{10}{9})^{14} \cdot x(i) > 2x(i)$.) This amounts to at most $2^7 + 3 \cdot 14 = 170$ tests.

Finally, if $x(i+1) \le 3.2 \cdot x(i)$, then by Lemma 2.70 it is enough to test independently for each $\mu = 1, 2, \ldots, 5$, for the first $2^9 + 1$ even nodes $j > i$ for which $x(j) + x(j-1) \in [(\frac{4}{3})^{(\mu-1)} \cdot x(i), (\frac{4}{3})^{\mu} \cdot x(i)]$ and also (2.b) holds. (For $\mu = 6$ we would get $x(j) + x(j-1) > (\frac{4}{3})^5 \cdot x(i) > 4.21x(i) > x(i) + x(i+1)$.) The number of these tests is at most $5 \cdot (2^9 + 1) = 2565$. The total number of tests for such an i is at most $2565 + 156 = 2721$.

It is clear that the tested groups can overlap. Notice also, that as soon as we encounter a node $j_0 > i$ s.t. $x(j_0) \le x(i)/4$, we don't need to test for any node to the right of $j_0 + 1$, since in this case $f(j_0) \le x(i)$ by Proposition 2.65. □

For a given (i, j) pair, the Procedure PAIRS below finds j when i is fixed, in PART 1 if $x(j) < x(i)/2$; resp. in PARTS 2-3 if $x(i)/2 \le x(j) \le 2x(i)$ and $x(j)+x(j-1) \le x(i)+x(i+1)$. PAIRS finds i when j is fixed in PARTS 2-3 if $x(i)/2 \le x(j) \le 2x(i)$ and $x(j) + x(j-1) \ge x(i) + x(i+1)$; resp. in PART 1 when $2x(i) < x(j)$.

Procedure PAIRS

for $i := 1$ to n **do**

PART 1. (we test for 156 nodes j of demand $x(j) \le x(i)/2$ to the right of i)
$j := i + 3$;
$\lambda := 1$; (set counter)
repeat
if $x(j-1) \le x(i)/4$ **then**
goto PART 2. ($r(i) \le j$ by Proposition 2.65)
end if
if $x(j) \le x(i)/4$ **then**
output (i, j); goto PART 2. ($r(i) \le j$ by Proposition 2.65)
end if
if $x(j) \le x(i)/2$ **then**

```
      output (i, j);     λ := λ + 1;
    end if
    j := j + 2;
until j > n or λ > 156
```

PART 2. (we treat the case $x(i+1) > 3.2x(i)$)

```
j := i + 3;    λ := 1;
if x(i + 1) ≤ 3.2x(i) then
  goto PART 3.
end if
repeat
  if x(j − 1) ≤ x(i)/4 or x(j) ≤ x(i)/4 then
    HALT;                                   (see Proposition 2.65)
  end if
  if x(j) ≤ 2x(i) and x(j) + x(j − 1) ≤ x(i) + x(i + 1) then
    λ := λ + 1;
    if x(i)/2 < x(j) then
      output (i, j);
    end if
  end if
  j := j + 2;
until j > n or λ > 128
if j ≤ n then
  j_0 := j;
  for ν := 1 to 14 do
    j := j_0;    λ := 1;
    repeat
      if x(j) ≤ (10/9)^ν · x(i)/2 and x(j) + x(j − 1) ≤ x(i) + x(i + 1) then
        λ := λ + 1;
        if (10/9)^(ν−1) · x(i)/2 < x(j) then
          output (i, j);
        end if
      end if
      j := j + 2;
    until j > n or λ > 3
  end for
end if
HALT;
```

PART 3. (we treat the case $x(i+1) \le 3.2x(i)$)

```
for μ := 1 to 5 do
  if (4/3)^(μ−1) · x(i) < x(i) + x(i + 1) then
    j := i + 3;    λ := 1;
    repeat
      if x(j − 1) ≤ x(i)/4 or x(j) ≤ x(i)/4 then
        HALT;                               (see Proposition 2.65)
```

```
        end if
        if x(j) + x(j − 1) ≤ (4/3)^μ · x(i) then
          λ := λ + 1;
          if (4/3)^(μ−1) · x(i) < x(j) + x(j − 1) and x(j) ≤ 2x(i) then
            output (i, j);
          end if
        end if
        j := j + 2;
      until j > n or λ > 513
    end if
  end for
  run symmetric variants of PART 1. 2. and 3. for j < i nodes;
end for
```

Definition 2.73 *Let Φ be a solution, and $k \in \langle 1, n\rangle$. The job k has a* **preemption** *on the level $a < f(k)$, if either $a \in \Phi(k)$, and $a+1 \notin \Phi(k)$; or $a \notin \Phi(k)$, and $a+1 \in \Phi(k)$.*

Theorem 2.5 *Let $k \in \langle 1, n\rangle$ be an arbitrary node, and Φ be a solution. If k is a high stair or loc-max, then it has at most $\frac{5}{\log 5-2} \log f_\Phi(k) < 16 \log f_\Phi(k)$ preemptions. If k is a loc-min, then $k-1$, k, and $k+1$ each has at most $\frac{5}{\log 5-2} \log \widetilde{f}_\Phi(k)$ preemptions.*

Proof. The intuition behind the proof is the same as in case of Lemma 2.72. Here we use that argument in a more precise form.

If k is a loc-min or a loc-max on the ξth floor of the minimum-tree, then let $\xi(k) := \xi$. If k is a stair above a loc-min i, then let $\xi(k) := \xi(i)$. We prove the theorem by induction on $\xi(k)$. For compact loc-mins the statement of the theorem trivially holds.

First, let $k = s = i + \nu$ be a high stair-up above a loc-min i. Clearly, if i has θ preemptions, then k has $\theta + \nu$ preemptions. Let $s < h_0 < h_1 < \ldots < h_\mu$ be the loc-mins following i that end in the stairs $s, s-1, \ldots, i+2$ on the left (see Figure 2.29 (i)). Now in the minimum-tree h_0 is the child of h_1; h_1 is the child of h_2, etc.

We partition the rungs of the stairs $i+2, i+3, \ldots, s$ into groups of at most 4 consecutive rungs, so that with each group the finish time increases by a factor of at least 5/4.

First we deal with those rungs, where the loc-mins $h_0 < h_1 < \ldots < h_\mu$ end. Concerning $h_0 < h_1 < \ldots < h_\mu$, we define the same cases (1) – (4) as in Lemma 2.67. First of all, we fix a level f in all the four cases as shown in Fig. 2.27: in (1) let $f := f(i_0)$; in (2) let $f := \max(f(w-1), f(w'+2))$; in (3) $f := \max(f(v-1), f(v'+1))$; and finally in (4) $f := \max(f(w-1), f(v+1))$.

Next, we proceed from child to parent along $h_0, \ldots, h_\tau, \ldots, h_\mu$, and always associate one or two rungs (of the stairs $i+2, i+3, \ldots, s$) to one loc-min in cases (1) and (2), resp. to a pair of loc-mins in cases (3) and (4).

If the loc-min h_τ, is in case (1) (i.e., like the node i_0 in case (1)), we associate to h_τ, the rung W (on one side), respectively W' (on the other side); if h_τ, is in case

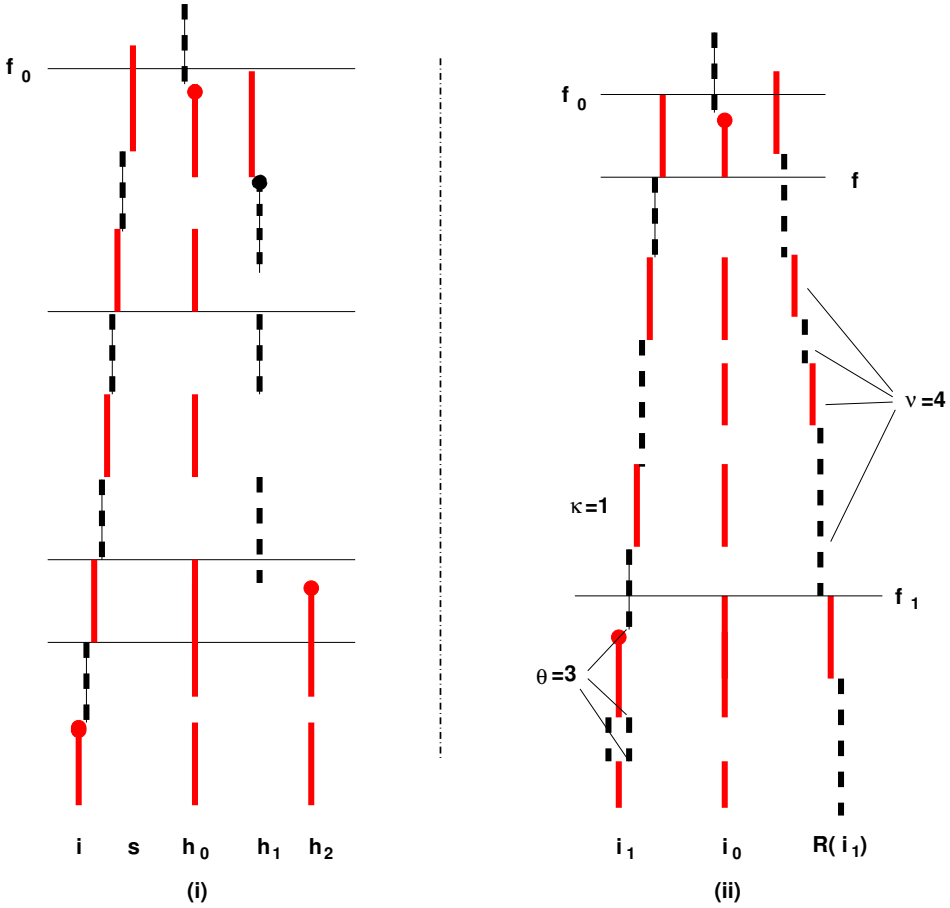

Figure 2.29: Illustration to Theorem 2.5: (i) shows rungs of high stairs above a loc-min i; here the rungs are grouped into 3 groups, he finish time within each group increases by a factor of 5/4. (ii) depicts a loc-min i_0 and its parent i_1; observe that i_0 has $\theta + \nu + 2\kappa + 1$ preemptions.

(2), we associate either W, or $V' \cup W'$ to h_τ; in case (3) we associate the rungs V' and W' to h_τ and $h_{\tau+1}$; in case (4) the rung W to h_τ and $h_{\tau+1}$ (we account for W' with the preemptions of loc-min i_1). Observe that in all cases $[f, f_0]$ is part of the assigned one or two rungs – which now form a so called *little group* –, furthermore in Lemma 2.67 we showed that $f_0 > (5/4)f$.

After that, we divide the rest of the rungs into groups of three, so that the remaining 1 or 2 rungs we always attach to a little group defined above. According to Lemma 2.68, each such triple increases the finish time by a factor of 5/4. We obtain

$$f(s) \geq (5/4)^{[\nu/4]} f(i+1) \geq (5/4)^{[\nu/5]} \widetilde{f}(i)$$

$$\log f(s) \geq \frac{\nu}{5}(\log 5 - 2) + \log \widetilde{f}(i)$$

$$\frac{5}{\log 5 - 2} \log f(s) \geq \nu + \frac{5}{\log 5 - 2} \log \widetilde{f}(i) \geq \nu + \theta.$$

This proves the theorem for high stairs.

Second, suppose that $k = i_0$ is a loc-min, and the theorem is proved for loc-mins on floors lower than $\xi(k)$.

Let i_1 be the parent of i_0 in the minimum-tree (for simplicity of presentation, here we allow that i_1 is compact). We will use the notation $f_0 := \widetilde{f}(i_0)$ and $f_1 := \widetilde{f}(i_1)$; for compact i_1, let $f_1 := f(i_1)$.

Consider again Figure 2.27: If i_0 is in case (4), then it has not more preemptions than i_1, (resp. $i_0 - 1$ has not more preemptions than $i_1 - 1$, etc.). If i_0 is in case (3), then by the properties (P1)–(P4), below the level $f(i_1)$, it has the same preemption levels as w'. Therefore, i_0 has as many preemptions as w', and $i_0 - 1$ (resp. $i_0 + 1$) has one more preemption. However, W' is a high rung, and by forming groups of 1 to 4 rungs we have shown above that $f_0 \geq (5/4)^{[\nu/4]}\widetilde{f}(i)$, where i is the loc-min below w'. We can add the rung Y of $i_0 - 1$ to the topmost group, and the inequalities remain valid with $\nu/5$ in the exponent.

Suppose that i_0 is in case (1) or (2). Let f be the level defined above for cases (1) and (2). We assume w.l.o.g. that $i_1 < i_0$. If $i_1 \equiv i_0 \ (mod\ 2)$, then the preemption levels of i_0 and i_1 are the same below $f(i_1)$. In this case let ν denote the number of stairs above $R(i_1)$ (i.e., preceding $R(i_1)$) and having finish time at most f. If $i_1 \not\equiv i_0 \ (mod\ 2)$, then the preemption levels of i_0 and $R(i_1)$ are the same below $f(i_1)$. In this case let ν denote the number of stairs following $i_1 + 1$, and finishing below f.

We assume that i_0 and i_1 are both even; the proof is similar when they are of opposite parity. Let $i_1 + 1$ have θ preemptions. The induction hypothesis implies $\theta \leq \frac{5}{\log 5 - 2} \log f_1$. Above the level f_1, let κ denote the number of those even rungs on the left, that are subsets of odd rungs on the right (see Figure 2.29 (ii)). Now i_0 has at most $\theta + \nu + 2\kappa + 1$ preemptions, and $i_0 - 1$ has at most $\theta + \nu + 2\kappa + 2$ preemptions.

Suppose first that $\kappa = 0$. According to the Lemmas 2.67 and 2.68, $f_0 \geq (5/4)f$, whereas $f \geq (5/4)^{[\nu/3]} f_1 \geq (5/4)^{(\nu-2)/3} f_1 \geq (5/4)^{(\nu-2)/5} f_1$. Thus,

$$f_0 \geq (5/4)^{(\nu-2)/5+1} f_1$$

$$\log f_0 \geq (\frac{\nu - 2}{5} + 1)(\log 5 - 2) + \log f_1$$

$$\frac{5}{\log 5 - 2} \log f_0 \geq \nu - 2 + 5 + \frac{5}{\log 5 - 2} \cdot \log f_1$$

$$\frac{5}{\log 5 - 2} \log f_0 \geq \nu + 3 + \theta \geq \theta + \nu + 2,$$

which proves the theorem if $\kappa = 0$.

Now let $\kappa \neq 0$, and Q be even rung on the left, being subset of an odd rung on the right. Let Q' be the rung above Q. Now by (O1) and (O3), $|Q'| \geq |Q| \geq f(q+1)/4$, and observe that we did not calculate with the increase of either $|Q|$ or $|Q'|$ in Lemma 2.68. Therefore, we get sufficient additional increase for the 2κ additional preemptions.

Finally, assume that $k = m$ is an even loc-max. If i_1 is the parent of m in the minimum-tree, then depending on parity, m has the same preemptions below f_1, as

one of i_1, or $R(i_1)$ (resp. $L(i_1)$). Let θ denote the number of these preemptions. Above $f(i_1)$, m has $\nu + 2\kappa$ preemptions, where ν is the number of preemptions of $m+1$, and κ is the number of odd rungs of $m-1$ that are subsets of even rungs of $m+1$. This yields an increase factor of at least $f(m) \geq (5/4)^{[\nu/3]+\kappa} f_1$. We obtain $\frac{5}{\log 5-2} \log f(m) \geq (\nu + 2\kappa) + \frac{5}{\log 5-2} \log f_1 \geq \nu + 2\kappa + \theta$. □

2.9 The algorithm

Let $x(1), x(2), ..., x(n)$ be an instance of the pSMC problem on the path $1, 2, ..., n$. In this section we present the $\mathcal{O}(\min(n^2, n \log p))$ algorithm that computes an exact optimal solution. We try to give a comprehensible and intuitive description, without giving up accuracy. We will refer to the procedures f-TIME, SELECT and PAIRS, the pseudocodes of which can be found in Sections 2.3.1, 2.7.4 and 2.8, respectively.

We treat the case of cycle conflict graphs in Section 2.9.3. There we show that, just like in the case of paths, a solution on cycles consists of one or more blocks, moreover it is easy to find a constant number of candidate nodes to be a 'starting node', i.e., one that is certainly compact in some optimal schedule of the cycle. After such a compact node is fixed, the algorithm essentially boils down to the same as on path conflict graphs.

The algorithm has two phases. In Phase 1 the blocks, and the minimum-tree of each block are determined. In Phase 2 the schedule is determined.

2.9.1 Phase 1

Finding the (i, j) pairs.

We group the nodes into $\lceil \log p \rceil$ groups: for $t = 1, \ldots, \lceil \log p \rceil$, nodes of demand in $[2^{t-1}, 2^t - 1]$ belong to group $\mathcal{G}_t$. First we traverse the path from right to left, and from a node k in $\mathcal{G}_t$ we set pointers to the nearest nodes of each of $\mathcal{G}_{t+2}, \mathcal{G}_{t+1}, \mathcal{G}_t, ..., \mathcal{G}_1$ to the right of k. After that, we do the same (symmetrically) from left to right. This preprocessing takes $\mathcal{O}(\min(n^2, n \log p))$ time.

Next, we search for potential $(i, j) = (i, r(i))$ and $(i, j) = (\ell(j), j)$ pairs of nodes. Procedure PAIRS in Section 2.8 finds all such pairs. However, due to the preprocessing, e.g., for fixed i we do not need to visit the nodes to the right of i, one by one. Recall, that PAIRS searches for nodes $j > i$, s.t. $x(j) \leq 2x(i)$; moreover, it needs not test nodes beyond the first node of demand $x(i)/4$. Consequently, if $\mathcal{G}_t$ is the group of i, we need to check (constant number of) nodes in the linked lists of $\mathcal{G}_{t+1}, \mathcal{G}_t, \mathcal{G}_{t-1}, \mathcal{G}_{t-2}$, restricting checks to nodes not further than the next node in $\mathcal{G}_{t-3} \cup ... \cup \mathcal{G}_1$. The procedure yields $\mathcal{O}(n)$ selected pairs.

Finally, we partition all the candidate (i, j) pairs into sets depending on their distances. In the following testing process the algorithm determines the optimum function $\mathcal{F}(\alpha)$ for each (i, j) dynamically, by proceeding from pairs of short distances to pairs of longer distances. During this, many (i, j) pairs get sorted out, that certainly cannot occur as $(i, r(i))$ or $(\ell(j), j)$.

The optimum function on $\langle i, r(i)\rangle$, for non-compact i.

Suppose we have i and j fixed. We assume (test for the possibility that) i is a non-compact loc-min and $j = r(i)$, or the symmetric case. The next part is completely based on the definitions and results of Section 2.6. For (i, j), procedure SELECT finds at most one even and one odd candidate node to be $pit(i, j)$, (or $top(i, j)$). We will discuss the running time of this procedure, and sketch how to determine the function $\mathcal{F}(\alpha) = \alpha + \mathcal{C}$ and the domain $\mathcal{D}_{\mathcal{F}}$ of α values in case of $m = top(i, j)$ or for both the potential even $k = pit(i, j)$, and odd $l = pit(i, j)$. Remember, that α is the number of black-black conflicts of (i, j) below $\Gamma = \min(x(i)+x(i+1), x(j)+x(j-1))$.

We claim that the running time of SELECT is $\mathcal{O}(\min(n, \log p))$. Recall, that this procedure proceeds upwards along rungs of stairs above i and j, and for two opposite rungs D and C of the same parity – say even –, it searches for even nodes $k \in \langle i, j\rangle$ such that $f_{a_{\max}}(k) \in D \cap C$, where $a_{\max}$ is a current upper bound on α. First, we observe that we can run the procedures SELECT and f-TIME (and $\hat{f}$-TIME) 'at once', and for given d and c determine a value $x_{d,c}$, so that $x(k) \leq x_{d,c} \Leftrightarrow f_{a_{\max}}(k) \leq \min(\overline{D}, \overline{C})$, for any even k. Suppose that t is minimum such that $\exists k_0 \in \mathcal{G}_t \cap \langle d+2, c-2\rangle$, and $x(k_0) \leq x_{d,c}$. Now only a constant number of even nodes from $\mathcal{G}_t$ and $\mathcal{G}_{t+1}$ can be between d and c, and these are the only even candidates. Since SELECT needs to visit only $\mathcal{O}(\min(n, \log p))$ rungs by Theorem 2.5, we obtain this as a bound on the running time.

Next, we turn to determining the function $\mathcal{F}(\alpha)$. Besides the candidate nodes k and l (or m), SELECT also determines the enclosing interval $[a_{\min}, a_{\max}]$ of $\mathcal{D}_{\mathcal{F}}$. Procedure ADJUST, which we sketch below, further cuts this interval as required by the connecting step of the dynamic algorithm. When we define the domain $\mathcal{D}_{\mathcal{F}}$ of the possible α values, we must take care that each α and $\mathcal{F}(\alpha)$ can be realized on $\langle i, j\rangle$, and that all realizable α are considered. (On the other hand, we don't need to bother, whether each α really occurs in some solution on $\langle 1, n\rangle$.)

First suppose that SELECT outputs a single $m = top(i, r(i))$ node. Such $(i, r(i))$ (resp. $(\ell(j), j)$) pairs correspond to the starting computations in the dynamic process. In this case, $\mathcal{F}(a_{\max}) = \mathcal{F}$, where $\mathcal{F}$ is obtained in part 5 of SELECT, as the sum of finish times for the best candidate m. Thus, $\mathcal{F}(a_{\max}) = a_{\max} + (\mathcal{F} - a_{\max})$, consequently $\mathcal{C} = \mathcal{F} - a_{\max}$ in the function $\mathcal{F}(\alpha) = \alpha + \mathcal{C}$. Moreover, we claim that $\mathcal{D}_{\mathcal{F}} = [a_{\min}, a_{\max}]$, that is, all α in this interval can occur. In particular, if m had less than $a_{\max} - a_{\min}$ black levels above $\max(f(m-1), f(m+1))$, meaning that $a_{\min}$ would not yield m as potential $top(i, j)$ node, then this is not an optimal solution for $a_{\max}$ either, (the algorithm would have found a better solution, or $j = r(i)$ is impossible). We saw the same argument in the proof of Theorem 2.1.

Second, suppose that SELECT output $k = top(i, j)$, $d = L(k)$, and $c = R(k)$. Furthermore, assume that the optimum function on $\langle d-1, k\rangle$ – i.e., the sum of finish times on $\langle d, k-1\rangle$ – has the form $\mathcal{F}'(\beta) = \beta + \mathcal{C}'$; resp. the optimum function on $\langle k, c+1\rangle$ has the form $\mathcal{F}''(\gamma) = \gamma + \mathcal{C}''$. Here β and γ are the numbers of black-black conflicts of $(d-1, k)$ resp. of $(k, c+1)$ below $f(k)$, and both domains $\mathcal{D}_{\mathcal{F}'}$ and $\mathcal{D}_{\mathcal{F}''}$ are intervals. These functions were computed earlier.

Procedure ADJUST($a_{\min}, a_{\max}$)

(We want that $\alpha \in [a_{\min}, a_{\max}] \Rightarrow f_\alpha(k) \in D \cap C$. We assume that k is even.)

if $f_{a_{\max}}(k) < \max(\underline{D}, \underline{C})$ **then**
  decrease $a_{\max}$ (and $b_{\max}$) so that $f_{a_{\max}}(k) = \max(\underline{D}, \underline{C})$;
end if
if $f_{a_{\min}}(k) > \min(\overline{D}, \overline{C})$ **then**
  increase $a_{\min}$ (and $b_{\min}$) so that $f_{a_{\min}}(k) = \min(\overline{D}, \overline{C})$;
end if

(Recall that there are a_0 odd-even conflicts between Γ and $\max(\underline{D}, \underline{C})$. The possible number of odd-even conflicts of $(d-1, k)$ ranges from $a_{\min}+a_0$ to $a_{\max}+a_0$; the number of even-odd conflicts ranges from $b_{\min}+b_0$ to $b_{\max}+b_0$. We restrict these intervals to their intersection with $\mathcal{D}_{\mathcal{F}'}$ resp. with $\mathcal{D}_{\mathcal{F}''}$.)

if $a_{\min} + a_0 < \min \mathcal{D}_{\mathcal{F}'}$ **then**
  $a_{\min} := \min \mathcal{D}_{\mathcal{F}'} - a_0$; increase $b_{\min}$ accordingly;
end if
if $a_{\max} + a_0 > \max \mathcal{D}_{\mathcal{F}'}$ **then**
  $a_{\max} := \max \mathcal{D}_{\mathcal{F}'} - a_0$; decrease $b_{\max}$ accordingly;
end if
do the same so that $[b_{\min} + b_0, b_{\max} + b_0] \subseteq \mathcal{D}_{\mathcal{F}''}$ holds;
if $a_{\min} > a_{\max}$ or $b_{\min} > b_{\max}$ **then**
  output $k \neq pit(i, j)$;
else
  output $[a_{\min}, a_{\max}] = \mathcal{D}_{\mathcal{F}}$;
end if

Since in the above procedure we always reduce $[a_{\min}, a_{\max}]$ to its intersection with some other interval, we obtain the following:

(1) the domain $\mathcal{D}_{\mathcal{F}}$ is an interval $[a_{\min}, a_{\min} + \Delta]$.
(2) the corresponding finish times of k range from $f_{\min}(k)$ to $f_{\min}(k) - \Delta$.
(3) β ranges from $\beta_{\min}$, to $\beta_{\min} + \Delta$ and γ ranges from $\gamma_{\min}$, to $\gamma_{\min} + \Delta$.

The stairs $\langle i+1, d-1\rangle$ and $\langle c+1, j-1\rangle$ have constant finish times, that total to some constant $\mathcal{S}$. Recall that we have the functions $\mathcal{F}'$ and $\mathcal{F}''$ for the sum of finish times on $\langle d, k-1\rangle$, resp. on $\langle k+1, c\rangle$. Thus, we obtain

$$\mathcal{F}(a_{\min} + \delta) = \mathcal{F}'(\beta_{\min} + \delta) + f_{\min}(k) - \delta + \mathcal{F}''(\gamma_{\min} + \delta) + \mathcal{S} =$$

$$= \beta_{\min} + \delta + \mathcal{C}' + f_{\min}(k) - \delta + \gamma_{\min} + \delta + \mathcal{C}'' + \mathcal{S} = \delta + \widetilde{\mathcal{C}},$$

where $\widetilde{\mathcal{C}}$ is a constant. That is, $\mathcal{F}(a_{\min} + \delta) = a_{\min} + \delta + (\widetilde{\mathcal{C}} - a_{\min})$, in other terms $\mathcal{F}(\alpha) = \alpha + \mathcal{C}$, where $\mathcal{C} = \widetilde{\mathcal{C}} - a_{\min}$, and $\alpha \in [a_{\min}, a_{\max}]$. Notice that above we provided again a formal proof of Theorem 2.3 (II).

In the previous argument we assumed that SELECT outputs only one $k = pit(i, j)$ candidate, moreover, the optimum functions $\mathcal{F}'$ and $\mathcal{F}''$ are linear. In general, we

might have another $l = pit(i, j)$ candidate, ending in the rungs of $d+1$ and $c-1$. On the other hand, recall that if i is on higher than second floor of a minimum-tree, then the inequality (2.a) of Section 2.6 must hold and one of k or l cannot be $pit(i, j)$. Assume that procedure CHOOSE picks k and drops l, if (2.a) holds.

Suppose that the even k, and the odd l candidates for $pit(i, j)$ yield the optimum functions $\mathcal{F}_k(\alpha)$ and $\mathcal{F}_l(\alpha)$ (on different domains). We have to take the minimum function $\mathcal{F} = \min(\mathcal{F}_k, \mathcal{F}_l)$, which will not be of the form $\mathcal{F}(\alpha) = \alpha+\mathcal{C}$ due to breaks, or holes in the domain. Moreover, not even $\mathcal{F}_k$ and $\mathcal{F}_l$, were of this form, for the same reason, so actually we might have several breaks and holes in $\mathcal{F}$. However, we can forget the solutions provided by $\mathcal{F}_l$, as soon as we are two steps further in the dynamic process, i.e., we stepped down at least two floors in the minimum-tree:

Say we paint pink the part of the domain – i.e. the large α –, where (2.a) does not hold for α, d, and c. After that, we paint red the part of the domain of $\mathcal{F}$ that corresponds to a pink part of either $\mathcal{F}'$ or $\mathcal{F}''$. Now on the remaining part of the domain $\mathcal{F}(\alpha) = \alpha + \mathcal{C}$ for some $\mathcal{C}$, by Theorem 2.3. In the next dynamic step, the red part can be excluded. This implies, that we will never have to deal with more than 4^2 breaks in the optimum function. However, we have to consider red and pink parts when i (or j) turns out to be on the first or second floor of the minimum-tree.

The optimum value on $\langle g, r(g)\rangle$, for compact g.

For an illustration see Figure 2.30. The last steps of the dynamic process are calculating the optimum on $\langle g, j\rangle = \langle g, r(g)\rangle$, (resp. on $\langle \ell(g), g\rangle$) assuming that g is compact. We also include here the case when $\langle g, j\rangle$ is a block and $x(g) = x(j)$. If a node g is fixed, then node $j = r(g)$ (in this broader sense) may be a node of arbitrary parity, s.t. $x(j) \le x(g)$.

When searching for nodes g and j of opposite parity, it suffices to consider all (g, j) pairs output by the procedure PAIRS. If $x(j) \le x(g)$, then we assume $j = r(g)$, and vice versa. We can restrict the tests to these pairs, since Lemmas 2.70 - 2.72 hold even for compact loc-mins.

If we search for g and j of the same parity, for fixed g we need to test for at most 15 nodes j, as stated in the corollary below:

Lemma 2.74 *Suppose that g and j are both even, g is compact, and either $j = r(g)$, or j is compact and $x(g) = x(j)$. Then there are no even nodes $k \in \langle g, j\rangle$, s.t. $x(k) \le \frac{10}{9} \cdot x(j)$.*

Proof. Suppose that $k \in \langle g, j\rangle$, and $x(k) \le \frac{10}{9} \cdot x(j)$. Below $f(j)$, the node k has the same black levels as j, and above $f(j)$, k has at most $\frac{1}{9} \cdot x(j)$ black levels. It is easy to see that k can not be an even stair above $g + 1$ or $j - 1$. Let $k_0 = pit(g, j)$. If k_0 is even, then $f(k) \ge f(k_0) > f(g + 1) > \frac{4}{3} \cdot f(g) > \frac{4}{3} \cdot f(j)$. If k_0 is odd, then $f(k) \ge \widetilde{f}(k_0) \ge \frac{4}{3} f(j)$. Therefore, by Lemma 2.44 it is worth placing the black levels of k below $f(j) + \frac{1}{9} \cdot x(j)$, a contradiction. □

Corollary 2.75 *For fixed even g, it is enough to test for 15 even nodes j in order to find all such (g, j) pairs.*

Proof. Assume that g is a fixed even node, and $\mathcal{G}_t$ is the group of g. We need to test for the nearest node of $\mathcal{G}_{t-3} \cup \mathcal{G}_{t-4} \cup \ldots \cup \mathcal{G}_1$, and at most 14 other $j > g$ not further than this node: for each of $\nu = 1, 2, \ldots, 14$, the nearest node to g of demand $x(j) \in [(\frac{10}{9})^{\nu-1} \cdot \frac{x(g)}{4}, (\frac{10}{9})^{\nu} \cdot \frac{x(g)}{4}]$. Since $(\frac{10}{9})^{14} \cdot \frac{x(g)}{4} > x(g)$, we don't have to consider $\nu > 14$ values. □

For fixed g and j, the candidate nodes for $pit(g, j)$ or $top(g, j)$ are found analogously to procedure SELECT. Note however, that for compact g we do not have a domain $[a_{\min}, a_{\max}]$ of the possible number of conflicts of (g, j) below Γ, but one fixed value: $a_{\text{fix}} = x(j)$ if $g \not\equiv j \pmod 2$, and $a_{\text{fix}} = x(g) - x(j)$ if $g \equiv j \pmod 2$.

On the other hand, if g is compact, then (P4) does not apply to $k = pit(g, j)$. Consequently, the number of conflicts a_{fix} does not determine $f(k)$, nor does it determine β or γ, which denote the number of conflicts of $(\ell(k), k)$ and $(k, r(k))$, respectively.

Next, we calculate the possible range of $f(k)$ values, assuming hat k is odd. Let $f_{a_{\text{fix}}}(k)$ be the finish time of k as computed by f-TIME. The optimal finish time $f_{\text{opt}}(k)$ has to be in rungs of some odd nodes $L(k)$ and $R(k)$ on both sides, so that $x(k) < f_{\text{opt}}(k) \leq f_{a_{\text{fix}}}(k)$. It is easy to show that there are not more than 3 different pairs of nodes (to be $L(k)$ and $R(k)$), whose rungs intersect $[x(k),\ f_{a_{\text{fix}}}(k)]$.

Let us fix $L(k)$ and $R(k)$ as well; this restricts the range of possible $f(k)$ values to an interval I. We have the optimum functions $\mathcal{F}'(\beta)$ on $\langle L(k), k-1\rangle$ and $\mathcal{F}''(\gamma)$ on $\langle k+1, R(k)\rangle$. The domains of these two functions directly correspond to two ranges $\mathcal{R}'$ and $\mathcal{R}''$ of possible $f(k)$ values (see also Figure 2.30). We take the intersection of $\mathcal{R}'$, $\mathcal{R}''$ and I. The maximum possible $f(k)$ value $f_{\text{opt}}(k) := \max \mathcal{R}' \cap \mathcal{R}'' \cap I$ will provide the optimum sum of finish times on $\langle g+1, j-1\rangle$ for this $k, L(k)$ and $R(k)$.

In case $m = top(g, j)$, then $\widehat{f}_{a_{\text{fix}}}$ is the finish time of m, and the sum of finish times on $\langle g+1, j-1\rangle$ is trivial to calculate.

Since we have a small constant number of candidates for $pit(g, j)$ or $top(g, j)$, we find the optimal solution for (g, j) in time proportional to selecting these candidates, i.e., in $\mathcal{O}(\min(n, \log p))$ time.

The optimum on a block $\langle g, h\rangle$.

We conclude Phase 1 with searching for (potential) blocks $\langle g, h\rangle$ and $\langle h, g\rangle$, assuming that $f(g) \geq f(h)$ holds.

Suppose that g is fixed, and we search h to the right of g. Since we allowed $x(r(g)) = x(g)$ in the previous paragraph, now we may assume that $r(g) \leq h$. The next claim is straightforward to prove using the operations (O1) and (O3) (see Figure 2.30).

Claim 2.76 *Let (g, h) be a block in a solution Φ, s.t. $x(g) > x(h)$. If $r(g) = s_\eta < \ldots < s_2 < s_1 < h$ are the consecutive stairs leading from $r(g)$ to h, then $x(s_{\tau+2}) \geq 2x(s_\tau)$, $\forall \tau \in [1, \eta-2]$.* □

The claim implies that for a fixed $(g, r(g))$ pair, any possible h is among the first $2 \log p$ nodes to the right of $r(g)$. Therefore, altogether there are at most $\mathcal{O}(\min(n^2, n \log p))$ potential (g, h) pairs.

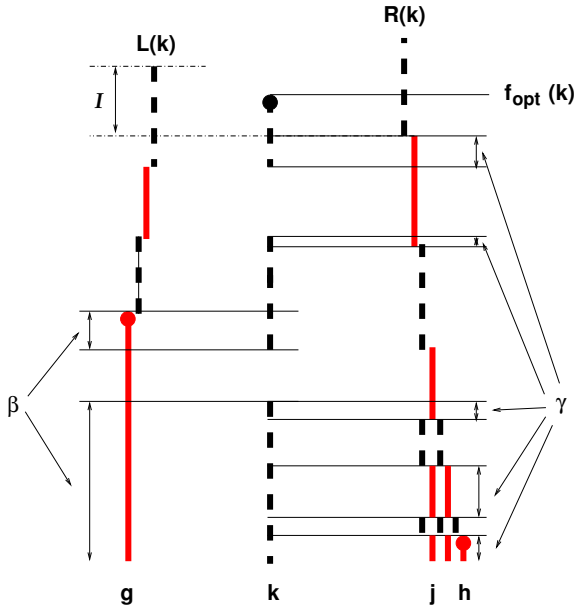

Figure 2.30: Sketch of an optimal schedule on a block $\langle g, h\rangle$. The nodes g and h are compact; $j = r(g)$ and $k = pit(g, h)$. For fixed $L(k)$ and $R(k)$ nodes, the algorithm has found the highest possible finish time of k, this is $f_{\text{opt}}(k)$. The number of conflicts of $(\ell(k), k)$ and of $(k, r(k))$ are denoted by β and γ, respectively. There is a trivial correspondence between the possible values β (or γ) and $f(k)$.

Let $\mathcal{F}(g, h)$ denote the optimum sum on $\langle g + 1, h\rangle$. For given $g < h$ there is obviously just one possible $r(g) \leq h$ node: either $r(g) = h$, or $r(g)$ is the highest stair above h (i.e., preceding h), such that $f(r(g)) < x(g)$ holds. Now $\mathcal{F}(g, h)$ equals the optimum on $\langle g + 1, r(g) - 1\rangle$ plus the finish times of stairs $r(g), \ldots, h$.

For $h < g$, the optimum sum on $\langle h + 1, g\rangle$, i.e., $\mathcal{F}(h, g)$ equals the sum of finish times of the stairs $h + 1, \ldots, \ell(g)$, plus the optimum on $\langle \ell(g) + 1, g - 1\rangle$, plus $x(g)$.

The optimum on the path.

We define a new graph $\mathcal{H} = (\mathcal{V}, \mathcal{E}')$ on the nodes $\mathcal{V} = \{0, 1, ..., n, n+1\}$, with weights on the edges: an edge connects g and h if $\langle g, h\rangle$ is a potential block (notice that $\langle 0, 1\rangle$ and $\langle n, n + 1\rangle$ may be blocks as well); the edge (g, h) is weighted by $\mathcal{F}(g, h)$. The graph has $\mathcal{O}(\min(n^2, n \log p))$ edges. Finally, we determine the optimum and the blocks on $\langle 0, n + 1\rangle$. Let $\mathcal{F}(g)$, denote the optimum on $\langle 0, g\rangle$ for compact g. For every node g, the algorithm determines $\mathcal{F}(g)$ dynamically:

$$\mathcal{F}(g) = \min\{\mathcal{F}(u) + \mathcal{F}(u, g) \mid 0 \leq u < g;\ (u, g) \in \mathcal{E}'\}.$$

The running time is proportional to the number of edges in $\mathcal{H}$.

2.9.2 Phase 2

In Phase 1, the algorithm computed the blocks, and the minimum-tree for each block in an optimal solution. By the latter we mean that for each potential $\langle i, j\rangle$ subpath

the algorithm stores the $m = top(i,j)$ node, or the $k = pit(i,j)$ and $l = pit(i,j)$ nodes that were output by SELECT, as well as the optimum function $\mathcal{F}(\alpha)$ and $\mathcal{D}_{\mathcal{F}}$. Recall that $\mathcal{F}(\alpha)$ consists of a few linear parts.

For $\langle g, r(g)\rangle$ with compact g, it stores one node $k = pit(g, r(g))$ (or $m = top(g, r(g))$) that yields the optimum sum of finish times. Moreover, in this case we need to store the finish time $f_{\text{opt}}(k) \in [x(k), a_{\text{fix}}(k)]$ that provides the optimum. The same holds for $\langle \ell(g), g\rangle$.

Let (g, h) be a block of τ nodes and $x(g) \geq x(h)$. First, we schedule the stairs $\langle r(g), h\rangle$. Observe, that we can schedule the stairs and compute $r(g)$ concurrently. Second, we schedule $k = pit(g, r(g))$ and the stairs $\langle g, \ell(k)\rangle$ and $\langle r(k), r(g)\rangle$. Notice that $\ell(k)$ and $r(k)$ follow directly from the stored optimal $f_{\text{opt}}(k)$ value. Since k is the root of the minimum-tree, it has black levels conflicting with both sides of the block (the number of these black levels follows again from $f_{\text{opt}}(k)$). These black levels are arbitrary, e.g., we can assume that these are the lowest possible such levels.

Next, suppose that $k_1 = pit(\ell(k), k)$ and $k_2 = pit(k, r(k))$. We schedule the stairs $\langle \ell(k)+1, \ell(k_1)\rangle$ and $\langle r(k_1), k-1\rangle$, together with scheduling k_1 itself. We do the same on $\langle k, r(k)\rangle$, and so on. If on the first levels there are other candidate nodes, like e.g. $l_1 \neq k_1$ to be $pit(\ell(k), k)$, one of them is chosen based on the number β of conflicts of $(\ell(k), k)$, which is a value in the domain of the optimum function for $(\ell(k), k)$. Obviously, every value in the domain is bound to (at least) one of the candidates k_1 or l_1.

The scheduling procedure for a loc-min or loc-max node is basically the same as f-TIME or $\widehat{f}$-TIME: the preemption levels in $\Phi(k)$ or $\Phi(m)$ are obtained by way of merging the finish-times of stairs on the two sides. Theorem 2.5 implies that a node has $\mathcal{O}(\min(\tau, \log p))$ preemption levels. Scheduling the whole block takes $\mathcal{O}(\min(\tau^2, \tau \log p))$ time.

2.9.3 Cycles

The algorithm is basically the same for cycle conflict graphs. Let $\{1, 2, \ldots, n\}$ be the consecutive nodes of a cycle so that $(i,\ i+1 \pmod n)$ is an edge for every $i \in [1, n]$. Like in the case of paths, we search for schedules having minimum value of $\sum_{i=1}^{n} f(i)$, and among all these schedules, having maximum value of $\sum_{i=1}^{n} f^2(i)$. We use the term **(optimal) solution** for such schedules.

We show in Proposition 2.77 that on cycles there is always at least one compact node g, so that the overlapping path $g, g+1, \ldots, g+n = g \pmod n$ consists of one or more blocks. In any solution Φ, every node belongs to one of the blocks, just like for path conflict graphs. The only additional difficulty is that we have to select a node g_0, which we assume to be compact in the solution we are about to compute. Having guessed a compact node g_0, this g_0 can become the starting node for the original path algorithm.

In Theorem 2.6 we state that there are at most 25 candidates to be such a compact g_0. By running the original algorithm with each of these candidates as compact nodes one by one, we achieve the same asymptotic running time bound $\mathcal{O}(\min(n^2, n \log p))$ as for paths, as summarized by Corollary 2.78.

Proposition 2.77 *If Φ is a solution to an arbitrary instance of the* pSMC *problem on a cycle conflict graph, then Φ has at least one compact node.*

Proof. Let g be the node of minimum finish time in Φ. Suppose that g is non-compact, that is, it is white on some level $\psi < f(g)$. We exchange the levels ψ and $f(g)$ on the whole cycle. Since every node has finish time at least $f(g)$, this operation does not increase the finish time of any node. On the other hand, it decreases the finish time of g by at least 1, so Φ was not optimal, a contradiction. Consequently, g was compact in Φ. □

Theorem 2.6 *In every instance of the* pSMC *problem on a cycle conflict graph, a set of at most* 25 *nodes exists, so that in any optimal solution at least one of these nodes is compact.*

Proof. Let Φ be a fixed optimal solution. Throughout the proof we measure the node distances modulo n.

Let $x(g)$ be minimum over all demands $x(1), x(2), \ldots, x(n)$. Our first candidate for a compact node is g. Second, if $x(g-1) \leq 4x(g)$, and $x(g+1) \leq 4x(g)$, then it is possible that $g-1$ *and* $g+1$ are compact nodes.

From now on we assume that g is even, non-compact, and that $\langle g-1, g+1\rangle$ is not a trivial block. Suppose that g is inside some block $\langle g_1, g_2\rangle$, where we do not exclude $g_1 = g_2$. Such a block exists by Proposition 2.77.

(1) Assume first, that $g_1 \not\equiv g_2 \pmod 2$.

Suppose that g_1 is odd, and g_2 is even. By the properties (P1)–(P4), in such blocks a loc-max m exists, s. t. $\langle g_1, m-1\rangle$ is the odd part and $\langle m+1, g_2\rangle$ is the even part below the level $\min(x(g_1), x(g_2))$ (see also Definition 2.17, and Figure 2.3). In the even part all even nodes are black below $\min(x(g_1), x(g_2))$. Since $x(g) \leq \min(x(g_1), x(g_2))$, and g is non-compact, g cannot be in the even part, i.e., $g \leq m$.

Our goal is now to find the potential odd candidates that might be g_1. On the one hand, Proposition 2.65 implies that $f(g_1+1) \geq (\frac{4}{3})x(g_1)$. On the other hand, every node in the odd part has finish time at least $f(g_1+1)$, which is obvious, e.g., by looking at Figure 2.16. Therefore, $(\frac{4}{3})x(g_1) \leq f(g_1+1) \leq f(g) \leq 4 \cdot x(g)$, by Proposition 2.65. We obtain that $x(g_1) \leq 3x(g)$.

Now let h be an odd node, s.t. $g_1 < h < g$. We show that $x(h) > (\frac{10}{9}) \cdot x(g_1)$. Since h is in the odd part, it has $x(g_1)$ black levels below $x(g_1)$. As for any node in the odd part, for h it holds as well that $f(h) \geq (\frac{4}{3})x(g_1)$. If h had only at most $(\frac{1}{9}) \cdot x(g_1)$ black levels above $x(g_1)$, then by Lemma 2.44 it would be worth placing these levels below $(\frac{10}{9}) \cdot x(g_1)$.

The candidates for g_1 are the nodes $h_1, h_2, \ldots, h_\kappa$, where h_1 is the nearest odd node to the left of g, such that $x(h_1) \leq 3x(g)$; and for $i \geq 1$, h_{i+1} is the nearest node to the left of h_i, such that $x(h_{i+1}) < (\frac{9}{10})x(h_i)$. Since all demands are at least $x(g)$, we get that $\kappa \leq 11$.

For the symmetric case that g_1 is even and g_2 is odd, we need to test at most 11 odd nodes to the right of g.

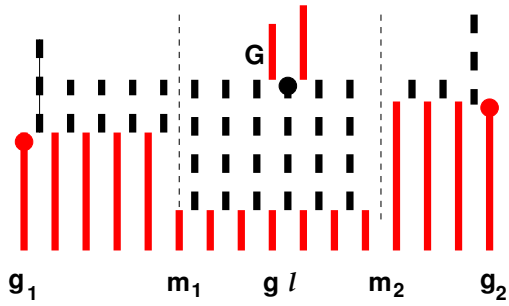

Figure 2.31: Illustration to Theorem 2.6, case (2).

(2) Second, assume that g_1 and g_2 are both even.

We show that this is impossible. Without loss of generality, let $x(g_1) \leq x(g_2)$. If $pit(g_1, g_2)$ is even, then the levels $[1, x(g_1)]$ are clear even, contradicting to $x(g) \leq x(g_1)$. Therefore, $l = pit(g_1, g_2)$ is odd. For illustration see Figure 2.31. Now two loc-max nodes $m_1 < l < m_2$ exist, so that the levels $[1, x(g_1)]$ are even on $\langle g_1, m_1 - 1\rangle$ and on $\langle m_2 + 1, g_2\rangle$, whereas some of these levels might be odd on $\langle m_1 + 1, m_2 - 1\rangle$. Obviously, g is not in the even parts, since $x(g) \leq x(g_1)$. On the other hand, if g is in the odd part, then either $g = l-1$, or $g = l+1$, because every other even node has larger demand than $\min(x(l-1), x(l+1))$ (one can see this again on Figure 2.16). This holds also for the loc-max nodes m_1 and m_2.

Suppose that $g = l - 1$, or $g = l + 1$. Now making g compact is for free on both sides, since $x(g) \leq \min(x(g_1), x(g_2))$. It costs $|G|$ due to increasing $f(l)$, but we decrease $f(g)$ by more than $|G|$, a contradiction.

(3) Finally, assume that g_1 and g_2 are both odd.

If $l = pit(g_1, g_2)$ is odd, then the levels $[1, \min(x(g_1), x(g_2))]$ are clear odd levels, and g_1 and g_2 are among the same candidate nodes that the algorithm finds in case (1) above.

Assume that $k = pit(g_1, g_2)$ is even. Now two nodes $m_1 < k < m_2$ exist, such that the levels $[1, \min(x(g_1), x(g_2))]$ are clear odd on $\langle g_1, m_1 - 1\rangle$ and on $\langle m_2 + 1, g_2\rangle$, whereas some of these levels might be even on $\langle m_1 + 1, m_2 - 1\rangle$. If g is in one of the odd parts (including m_1 and m_2), then the tests of case (1) will output g_1, resp. g_2.

If g is in the even part $\langle m_1+1, m_2-1\rangle$, then $g = k$, since all other even nodes in the even part have demand strictly larger than $x(k)$. We search for candidates to be g_1. Obviously, $x(g_1) < f(k) \leq 3x(g)$, where the latter follows from Proposition 2.65. We claim also, that $x(g_1) \leq x(k-1)$, otherwise it would be worth placing all the levels of $k-1$ below $x(g_1)$, by the same argument as the one we saw in case (2). Consequently, for every odd node h between m_1 and $k - 1$, trivially $x(g_1) \leq x(k - 1) < x(h)$.

If $x(g - 1) > 3x(g)$, then the first node $h_1 < g$ of demand at most $3x(g)$ must be in the odd part $\langle g_1, m_1\rangle$, and since for nodes in the odd part the same holds as in case (1), g_1 will be among the same $h_1, h_2, \ldots, h_\kappa$, as defined in (1).

If $x(g - 1) \leq 3x(g)$, then we *modify* the group of candidates of case (1), to the left of g (i.e., we drop the first variant completely): let $h'_0 := g - 1$, and let h'_1 be the nearest odd node to the left of $g - 1$, s.t. $x(h'_1) \leq x(h'_0)$. Then, unless

h'_0 or h'_1 is compact, h'_1 is in the odd part in case (1), and also in case (3). The recursive definitions of $h'_2, h'_3, \ldots, h'_\kappa$ are the same as in case (1). The modified set $\{h'_0, h'_1, \ldots, h'_\kappa\}$ includes the compact g_1 node in both cases (1) and (3).

The possibilities we test comprise the 2 trivial cases, at most 12 test nodes to the left of g, plus at most 11 test nodes to the right of g, a total of 25 tests. □

Corollary 2.78 *An exact optimal solution for pSMC problem on cycle conflict graphs can be computed in* $O(\min(n^2, n \log p))$ *time.* □

2.10 Discussion

In the previous section our main concern was to present the algorithm in a relatively exact, but still understandable manner. As one may see in this description, or in procedure SELECT of Section 2.6, our result in its current form is of rather theoretic nature. Although it may read strange, we firmly believe that practice and reality – let alone 'typical' instances of the problem – are in a sense much simpler. Below we mention a few aspects where most probably simplification is achieveable. The improvements would either require new ideas, or an (even) more detailed and tiresome investigation using the given proof methods.

The algorithm runs in $\mathcal{O}(\min(n^2, n \log p))$ time, but this expression hides a considerable constant, in the order of 10^3. This is mainly due to the fact that when we search for all possible $(i, j) = (i, r(i))$ and $(i, j) = (\ell(j), j)$ pairs, for fixed i we admit by far too many nodes j. As we pointed out in Section 2.8, the number of such tests could be reduced radically, by proving more sophisticated statements than Lemmas 2.67–2.72 about the structure of the solution on $\langle i, r(i) \rangle$.

Similarly, in part 2 of SELECT, we admit a possibly large set $\{k_1 \ldots, k_\xi\}$, from which to choose the node $pit(i, j)$.

Notice that the above issues become crucial only with large instances. As noted in the first paragraph of Section 2.8, by applying a straightforward check of all node-triples to be $(i, pit(i, r(i)), r(i))$, or $(\ell(j),\ pit(\ell(j), j), j)$ we obtain an $\mathcal{O}(n^3 \cdot \min(n, \log p))$ algorithm with a small constant. However, even in this case we have to face the problem of selecting the optimal $k = pit(i, j)$, respectively of choosing from the even k and odd l. Thus, the *code* of SELECT would not become much simpler, as we will argue in the next paragraphs.

Now let us turn to the cumbersome proof of Theorem 2.3 (Section 2.7.3). First of all, we remark that the very nature of this proof involves the potential of having a few typos or bugs left in the write-up. Nevertheless, after long investigation of this problem, now it stands clear that the really crucial cases are those treated in Claims 2.60 and 2.63. The obvious challenge concerning this proof, is the desire to make it simpler and more elegant or, on the other hand, to improve the results themselves.

The clumsiness of some results, like Lemmas 2.54 and 2.56, induce clumsiness of the algorithm, when it comes to choosing the only possible $k = pit(i, j)$ node for instance in procedure CHOOSE. The *early* selection of this node does not affect the output schedule, but it ensures the polynomial running time in that we need not store

complicated optimum functions $\mathcal{F}(\alpha)$ with too many breakpoints. Alternatively, in SELECT and CHOOSE we could try to focus more on the valid domain $\mathcal{D}_{\mathcal{F}} \subseteq [a_{\min}, a_{\max}]$, (i.e., on the really occuring situations), or at least find one value of $\mathcal{D}_{\mathcal{F}}$. There, finding the best $k = pit(i,j)$ should be trivial: by the uniqueness theorem, on the whole $\mathcal{D}_{\mathcal{F}}$ there is a best $\mathcal{F}_k(\alpha)$ function (over all k and l), so we would just need to compare fuction values to find the best candidate for k.

All in all, we believe that choosing the right $pit(i,j)$ could be a much more trivial process: For higher levels of the minimum-tree we conjecture that the lowest such stairs d and c for which a node k exists s.t. $f_{a_{\max}}(k) \leq \min(\overline{D}, \overline{C})$ are the only possible $d = L(k)$ and $c = R(k)$ nodes; moreover the k (of the same parity) of minimum $x(k)$ is the only possible $pit(i,j)$. Finally, it would be nice to get rid of the anomalies on the first two levels of the minimum-tree, where so far we could not exclude the presence of two candidates for $pit(i,j)$. Such a new result could make the algorithm significantly clearer. However, even if the latter simplification is possible, one would need new ideas to prove it.

The current algorithm does not try to optimize the total number of colors used (see the *chromatic strength* in Section 1.1); even worse, the type of solutions we are focusing on typically use many colors. However, once an optimal schedule is at hand, it should be possible to reduce the maximum finish time by inverse operations of the level exchange depicted in Figure 2.2.

It is natural to ask, whether the asymptotic running time $\mathcal{O}(\min(n^2, n \log p))$ is tight. At first glance one would say 'yes': having one block with a single loc-max node, there can be $\mathcal{O}(\log p)$ and also $\mathcal{O}(n)$ preemptions on average per node, so that the output alone consists of $\mathcal{O}(\min(n^2, n \log p))$ data. However, we can observe that in an optimal schedule any preemption level equals either the finish time of a node, or the one extra preemption level in the root of a minimum-tree (e.g. level 2 of node 7 on Fig 2.1). Hence the total number of *different* preemption levels is less than $2n$. If in the output we may assign a certain preemption level to a whole set of nodes at once (e.g., in $\langle i, m \rangle$ the odd jobs should be started, and even jobs should be stopped at time φ), then it is not excluded that by applying more sophisticated data-structures on the input instance (or by a completely different approach altogether), the time-bound could be reduced to linear. Note also, that the current running time is valid if we define $p := \max_i x(i) / \min_i x(i)$, instead of $p := \max_i x(i)$.

Finally, what is the importance of having a fast algorithm for path and cycle conflict graphs? Beside these two basic graph classes, our result could be well exploited on types of graphs, where nodes of higher degree are sparse, in the sense that the distance between any two nodes of degree at least 3 is large, so that the solution on the connecting path contains (with high probability) a compact node. A first step of research in this direction would be to find efficient algorithms for *generalized star* (or *spider*) graphs, which have one central node of high degree, and several paths meeting in the central node. We note here that the binary tree used in the NP-hardness proof of [73] has mainly degree 3 nodes, except for short paths 'attached' to the leaves (the paths are parts of the penalty gadgets; cf. Section 2.1.3), having only the last, degree 1 node compact in the optimum schedule.

Chapter 3

Monotone scheduling

This chapter deals with the problem of scheduling a set of n jobs on m machines of different speeds, or $Q||C_{\max}$ using the standard short notation. For a formal elementary introduction, we refer the reader to Section 1.2. Recall that we consider a scenario where the speed of each machine is known only by the (owner of the) machine, and it is supposed to declare its speed to a scheduling *mechanism*. In particular, we aim at designing a *truthful* mechanism $\mathcal{M} = (A, P)$, that consists of a scheduling algorithm A, and a payment function $P = (P_1, \ldots, P_m)$, where the payments are defined so that they motivate truthful speed declarations of the machines.

As our primary result, we obtain a deterministic, 2.8-approximation truthful mechanism for the scheduling problem. Furthermore, we derive improved approximation bounds of the LPT heuristic in the 'one fast machine' case, i.e., for speed vectors of the form $\langle 1, 1, \ldots, 1, s\rangle$.

We start the chapter by sketching the line of related results that our work is based on. In the course of this, in Section 3.1 we introduce two important features of the payment scheme P, namely *voluntary participation* and *frugality*, which will be revisited in relation to our work in Section 3.8.

Outline.

Section 3.1 cites the few most relevant papers about truthful mechanisms for $Q||C_{\max}$, as well as a classic paper about the approximation bounds of LPT scheduling. In Section 3.2 we present the basic terms and notation, some general properties of the greedy schedule LPT, and the monotone algorithm LPT*.

As a warm-up, in Section 3.3 we give a simple proof that LPT* is a 3-approximation algorithm. The main result, that LPT* is monotone, is proven in Section 3.4.

After that, in Sections 3.5 – 3.7 we return to approximation bounds:

We start with an outlook on the 'one fast machine' case by providing a tight approximation bound of $\frac{\sqrt{3}+1}{2}$ for LPT. Using a refined argument, but following the same lines, for arbitrary speed vectors we show an improved worst case ratio of 2.8 for LPT*. As for approximation lower bounds, instances having bound strictly larger than $\frac{\sqrt{3}+1}{2}$ are presented for LPT when all machine speeds are powers of 2.

We also show how to modify these instances to give a lower bound of more than $\sqrt{3}+1$ for LPT* on arbitrary speeds.

Finally, Section 3.8 treats properties of the payment function, and summarizes the results concerning the truthful mechanism. We end the chapter with discussions.

3.1 Related work

3.1.1 A randomized truthful 2-approximation mechanism [4, 5].

Recall that a scheduling algorithm for $Q||C_{\max}$ is called *monotone*, if by increasing the speed of any particular machine in the input, this machine is assigned not less work (total job size), than with its original speed. As a seminal contribution, Myerson [79] and independently Archer and Tardos [5] have proven that the monotonicity of algorithm A is a necessary and sufficient condition for a mechanism (A, P) to admit a truthful payment scheme (see Theorem 1.1 on page 11). These results relate to truthful mechanisms on a more general level (and in two different contexts), but the authors of [5] take the problem $Q||C_{\max}$ as their main example.

First of all, they show that a monotone allocation of jobs does not prohibit optimality. Consider any fixed order of the machines, $1, \ldots, i, \ldots, m$. A vector $\langle w_1, w_2, \ldots, w_m\rangle$ is *lexicographically smaller* than $\langle \bar{w}_1, \bar{w}_2, \ldots, \bar{w}_m\rangle$, if for some i $w_i < \bar{w}_i$, and $w_k = \bar{w}_k$ for $k < i$. An optimal schedule of the jobs, having a lexicographically minimal vector of assigned work, is a monotone allocation: Assume that the speed s_i of some fixed machine i is reduced. The assigned works do not change until i becomes a bottleneck machine, meaning that it is filled up to the optimum makespan. If after that the speed is further reduced to some s'_i, then the new optimum makespan will be at most w_i/s'_i, so i does not get more work than w_i in an optimum schedule.

We remark that the existence of an optimal, *monotone* allocation cannot be taken for granted. In the same paper, a simple input instance is shown for the $Q||\sum w_jC_j$ problem (scheduling with minimizing the weighted sum of job completion times), for which a monotone algorithm cannot achieve approximation ratio better than $\frac{2}{\sqrt{3}}$. In a different setting, another negative result concerning unrelated machine scheduling was proven by Nisan and Ronen [82], stating that no truthful mechanism for the $R||C_{\max}$ problem can have approximation ratio better than 2. This lower bound was recently improved to $1+\sqrt{2}$, by Christodoulou, Koutsoupias and Vidali [27].

In view of this, the monotone optimal allocation for $Q||C_{\max}$ seems promising. However, since $Q||C_{\max}$ is NP-hard, there is most probably no efficient algorithm to compute an optimal allocation.

As a polynomial time solution, in [5] a randomized truthful 3-approximation mechanism is presented, using the following simple algorithm:

The algorithm first computes an appropriate lower bound T on the optimum makespan. After that, it fills the machines up to time T one by one, in decreasing order of speed. This filling process is very simple. In order to make a so called *fractional assignment* of the jobs, the jobs are allocated to the machines (in decreasing order of size), so that when a machine's filling reached time T, then the superfluous

part of the topmost job is cut off, and put to the next machine. The lower bound T is selected so, that if a (fractional) job t_j is assigned to machine i, then $t_j < s_i \cdot T$, i.e., the job alone would fit below T on the machine. Thus, every job is cut into at most two parts in this allocation.

The randomized algorithm is now straightforward: If, in the fractional assignment, a p_j fraction of job t_j is assigned to machine i, and a $1-p_j$ fraction to machine $i+1$, then t_j is scheduled on i with probability p_j and on $i+1$ with probability $1-p_j$. This is certainly a 3-approximation algorithm, since each machine receives at most 2 extra jobs, the fractional jobs on the bottom and on the top of the machine.

In his thesis [4], Archer modified this randomized allocation as follows: Select a single α in $[0,1]$, uniformly at random. For every j, schedule job t_j, on machine i, if $\alpha \leq p_j$, and on machine $i+1$, if $\alpha > p_j$.

Let us assume that a fraction $1-p_k$ of job t_k was put to the bottom, and a fraction p_j of job t_j to the top of machine i. If $1-p_k+p_j < 1$, then $p_j < p_k$ excludes that both jobs are eventually allocated to i. The machine receives at most one of t_k or t_j as a full job. If $1-p_k+p_j \geq 1$, then even if both jobs are rounded to the machine i, this overloads the machine by at most $(p_k+1-p_j)\cdot s_i \cdot T \leq s_i \cdot T$. Thus, the allocation of Archer provides a 2-approximation.

According to the definition in [82], a randomized algorithm is considered truthful, if truth-telling is the best strategy for the agents (machines) for any possible outcome of the random coin flips of the algorithm. The paper [5] relaxes this requirement, and demands only that truth-telling is the best strategy for agents who try to maximize their *expected* profit.

Note that in our case the expected work $\bar{w}_i$ on machine i is exactly the work allocated to i in the fractional assignment. By Theorem 1.1, the mechanism admits a truthful payment scheme, if $\bar{w}_i$ is a monotone function of the (declared) speed. If the speed of machine i is decreased to s'_i, this increases the lower bound T to at most $\frac{s_i}{s'_i}T$. Consequently, the work $\bar{w}_i$ of i in the fractional assignment is at most $s'_i \cdot \frac{s_i}{s'_i}T = s_iT = \bar{w}_i$ if i was full originally. If it was not full, then increasing T reduces the received work, since all the preceding machines get more work than before. Therefore, the fractional assignment is monotone.

Computing the payments.

With payments as determined by Theorem 1.1 (see Section 1.2.1), the above algorithm constitutes a truthful mechanism. Recall that P_i stands for the payment handed to machine i, and $\langle b_1, \ldots b_m \rangle$ is the vector of *bids*, i.e., of the *declared* inverse speeds; b_{-i} denotes the bid vector modulo the bid b_i.

The theorem claims that the profit $P_i - w_i/s_i$ of agent i is maximized at $b_i = 1/s_i$, if and only if the payment has the form

$$P_i(b_{-i}, b_i) = h_i(b_{-i}) + b_i w_i(b_{-i}, b_i) - \int_0^{b_i} w_i(b_{-i}, u)\, du$$

for some arbitrary constant h_i.

Moreover, a reasonable mechanism must ensure, that the profit of a truth-telling agent will not happen to be negative, that is, the mechanism satisfies **voluntary participation** [5]. This can be achieved by setting h_i properly:

Theorem 3.1 [5] *A decreasing output function admits a truthful payment scheme satisfying voluntary participation if and only if* $\int_0^\infty w_i(b_{-i}, u)\, du < \infty$ *for all* i, b_{-i}. *In this case we can take the payments to be*

$$P_i(b_{-i}, b_i) = b_i w_i(b_{-i}, b_i) + \int_{b_i}^{\infty} w_i(b_{-i}, u)\, du.$$

Observe, that we obtain the above formula for the payments, if we set the constant $h_i(b_{-i})$ in Theorem 1.1 to be equal to $\int_0^\infty w_i(b_{-i}, u)\, du$.

The formula for the payments is shown to be computable in polynomial time for the randomized truthful mechanism. To prove this, the authors analyze the function w_i in the integral expression, and argue that the domain of $w_i(b_{-i}, u)$ can be partitioned into (a polynomial number of) intervals, and inside each interval w_i has a nice analytic form (e.g., linear). The voluntary participation condition, i.e., that $\int_0^\infty w_i(b_{-i}, u)\, du < 0$ is finite, is implied by the trivial observation, that the total assigned work to any machine is bounded by the sum of all job sizes ($w_i \leq \sum_j t_j$), whereas a machine with very low speed receives no jobs ($w_i = 0$), as soon as giving all its jobs to the fastest machine would result in a better makespan.

Frugality. Archer and Tardos call the mechanism **frugal**, if the sum of the payments that guarantee truthfulness and voluntary participation, does not exceed the total cost of the agents enormously. They show that their randomized monotone algorithm is frugal, in the sense that the ratio of the total payment and total cost is at most logarithmic, in case the fastest machine does not dominate the processing power:

Theorem 3.2 [5] *If the sizes of all* n *jobs differ by a factor of at most* r_1, *and the speeds of the two fastest machines differ by a factor of* r_2, *then the payment given by the mechanism exceeds the total expected cost incurred by all the agents by a factor of at most* $\mathcal{O}(r_2 \ln(r_1 n))$.

In particular, the ratio of payment to expected cost turns out to be at most $r_2(1 + 2\ln(r_1 n))$ for the fastest machine, and at most $1 + \ln(r_1 n / r_2)$ for other full machines in the fractional assignment.

The frugality of a truthful mechanism is far from being obvious. The same authors show in [6] that there exist problems, for which every reasonable mechanism has to pay $\Omega(n)$ times more than the actual cost.

3.1.2 A deterministic truthful mechanism [8].

The first *deterministic* monotone algorithm for $Q||C_{\max}$ was found by Auletta, DePrisco, Penna, and Persiano. As opposed to the randomized algorithm, which

used rounding of a fractional assignment, their starting point was a classic PTAS of Graham [44], for a *fixed number of machines*. In order to provide a $(1+\epsilon)$-approximate solution, the algorithm of Graham starts with an optimal schedule of the largest $h(\epsilon)$ jobs, and then allocates the remaining jobs according to the LPT rule. In order to ensure monotonicity, the authors carry out some modifications:

First, in place of LPT, they use the following version of LPT called UNIFORM. Assume that the speeds are the integers $s_1 \leq s_2 \leq \ldots \leq s_m$, then UNIFORM runs LPT on $S = \sum_i s_i$ identical machines. After that, it reorders the assigned work on the identical machines in increasing order. Finally, the set of machines (together with the assigned sets of jobs), is partitioned proportionally to the speeds $s_1, \ldots, s_m$. The first s_1 of the identical machines correspond to (the original) machine 1, the next s_2 identical machines to machine 2, etc.

Second, this algorithm UNIFORM is monotone, only if the speeds $s_1, \ldots, s_m$ are *divisible*, i.e., they are taken from a set $\{c_1, \ldots, c_i, \ldots\}$, where $c_i | c_{i+1}$. The idea of applying a common algorithm like LPT on *2-divisible* (or divisible) machines in order to get a monotone allocation, appeared first in this paper. The authors did not prove, however, they conjectured the monotonicity of LPT on divisible speeds.

The third modification is that the (lexicographically minimal) optimum schedule OPT on the large jobs, and UNIFORM on the small jobs, are run independently, unlike in Graham's PTAS. Since both OPT and UNIFORM are monotone algorithms, their independent combination is also monotone.

Compared to the PTAS, we lose a factor 2 on the approximation ratio due to rounding the arbitrary input speed vector to 2-divisible, and another factor 2 by running OPT and UNIFORM independently. The final result is a $(4+\epsilon)$-approximating truthful mechanism, computable in polynomial time for fixed number of machines.

3.1.3 A monotone FPTAS for fixed number of machines [3].

Andelman, Azar, and Sorani [3] improve on the above results concerning the $Q||C_{\max}$ problem in two ways: On the one hand, they provide the first (efficient) deterministic truthful mechanism for an arbitrary number of machines, which has worst case ratio 5. They combine former methods as follows. Like in the randomized algorithm of [5], they calculate a lower bound T on the optimum makespan. Instead of using a fractional assignment, they assign integral (complete) jobs to the machines, in decreasing order of machine speeds and job sizes, until the machines are filled up at least to time T. In order to obtain a *monotone* algorithm, they apply this technique on machines with speeds rounded down to $s_1/2.5^{l_i}$, where s_1 denotes the largest speed, and l_i is an integer. The integral assignment is a 2-approximation, and machine speeds are changed by a factor of at most 2.5, thus this is a 5-approximation mechanism. As an additional trick, the fastest speed is increased so that the first two (rounded) speeds differ by a factor of at least 4. By this, it is ensured that decreasing any other speed $i > 1$ results in an increase of T by a factor $\leq 5/4$, and therefore the machine i receives work $\leq 2 \cdot (\frac{5}{4} \cdot T) \cdot \frac{s_i}{2.5} = T \cdot s_i$ which is at least the original work of the machine.

On the other hand, for constant m, they show a simple monotone PTAS, and

a monotone FPTAS. In the PTAS algorithm the job sizes are normalized so that $\sum t_j = 1$; furthermore, jobs of size smaller than ϵ^2/m^2 are merged into chunks of size between $\epsilon^2/2m^2$ and ϵ^2/m^2. Observe that this modifies the job vector independently of the machine speeds. Finally, a lexicographically minimal optimal assignment of the new jobs is computed. This is a monotone allocation, it gives a $(1+3\epsilon)$-approximation, and the number of new jobs is $\mathcal{O}(2m^2/\epsilon^2)$, so computing the optimal schedule takes constant time.

In the monotone FPTAS mechanism machine speeds are first rounded to powers of $\frac{1}{(1+\epsilon)}$, and then the fastest speed is normalized to 1. A power l is determined s.t. a machine of speed $\frac{1}{(1+\epsilon)^l}$ certainly receives no work in an optimal solution. After that, for *all* possible speed vectors $\frac{1}{(1+\epsilon)^{i_1}}, \ldots, \frac{1}{(1+\epsilon)^{i_m}}$ $(0 \leq i_1, \ldots, i_m \leq l)$ a common $(1+\epsilon)$-approximation algorithm (of an FPTAS) is run. Every result is tested on the input (rounded) speed vector, (with the assigned works reordered according to machine speeds), and finally the schedule having minimum makespan is returned.

In proving that this algorithm is monotone, the key observation is that independently of the speeds, the solution is selected from the same set of schedules. Assume that a machine had finish time α in an optimal allocation of makespan T. If the speed of this machine is reduced by a factor of $(1+\epsilon)$, then with the same allocation the makespan will be at most $\max(\alpha(1+\epsilon), T)$.

We remark that all monotone algorithms cited above imply payment functions computable in polynomial time, and satisfying the voluntary participation condition. The papers [3, 8] do not treat frugality issues.

3.1.4 The approximation ratio of LPT [42].

Gonzalez, Ibarra, and Sahni [42] were the first to consider the greedy LPT heuristic on machines of different speeds. On the performance ratio for arbitrary speeds, an upper bound of $\frac{2m}{m+1}$, and an asymptotic lower bound of 3/2 is given (both of these bounds were later improved [33, 37]). Here we illustrate the upper bound proof by deriving an upper bound of 2 : Assume a counter-example using a minimum number of machines, so that $Lpt > 2 \cdot Opt$ where Lpt is the makespan of LPT and Opt is the optimum makespan of this instance. A simple argument shows (see Section 3.2.2), that no machine is idle in an optimal allocation of this instance. Consequently, $t_n/s_1 < Opt$, where s_1 denotes the smallest speed. However, in the LPT schedule every machine i has finish time at least $Lpt - t_n/s_i > Lpt - t_n/s_1 > Lpt - Opt > Opt$, implying that the total work scheduled is more than $Opt \cdot \sum s_i$, so there cannot be an assignment of makespan Opt, a contradiction.

In the second part of the paper, the authors turn to the 'one fast machine' case. They prove that then the worst case ratio of LPT is between 4/3 and $3/2 - 1/(2m)$. Moreover, they formulate the conjecture that the bound 4/3 is tight. In their lower bound instance the job sizes are $\langle 1.5, 1.5, 1, 1, 1\rangle$, and the speeds of the machines are $\langle 1, 1, 2+\epsilon\rangle$. LPT assigns jobs t_1, t_2 and t_5 to machine 3, and has makespan $\frac{4}{2+\epsilon} = 2 - \epsilon$. OPT assigns t_1 and t_2 to machines 1 and 2, respectively, and the three

jobs of size 1 to machine 3, implying makespan 1.5.

3.2 Preliminaries

We use t_j to denote both the jth job, and the size of the jth job in formulas. We assume $t_j \geq t_{j+1}\,(1 \leq j < n)$, and $s_i \leq s_{i+1}$ $(1 \leq i < m)$, i.e., the jobs sizes are in decreasing, resp. machine speeds are in increasing order. Throughout the chapter, t denotes the (size of) the last job t_n. We will use the short expressions 1-**job**, y-**job**, t-**job** for a job of size 1, y, t, etc. Similarly, a **1-machine** or a **4-machine** mean a machine of speed 1 or 4, respectively. We say that machine h is **to the right (left)** of i if $i < h$ $(h < i)$.

The **work** and the **finish time** of machine i in LPT is denoted by w_i, resp. $f_i = w_i/s_i$. In the upper bound proofs these values will be redefined so that they disregard the last job t_n. In other cases when the work w_i or finish time f_i of machine i is considered at an intermediate step of the algorithm, this is emphasized by a $\tilde{}$ or some superscript over w_i and f_i.

The **completion time** $C(t_j)$ of a job t_j assigned to machine i is the finish time of i right after t_j was scheduled.

The speed vector $\langle s_1, s_2, \ldots, s_m\rangle$, or the machines are called **2-divisible** if $s_i = 2^{l_i}$ $(l_i \in \mathbb{Z})$ for all i. In this definition we allow fractional speeds (e.g. 1/2), only for sake of simpler presentation of our proofs. Clearly, they are not essential to the result.

Next, we give a formal definition of the LPT algorithm. After that, we continue with a simple property of LPT.

LPT **algorithm:** Input: $\langle s_1, \ldots, s_m\rangle$ and $\langle t_1, \ldots, t_n\rangle$

At step j of LPT let w_i^j denote the work of machine i $(1 \leq i \leq m)$. LPT assigns t_j to machine h if

$$\frac{(w_h^j + t_j)}{s_h} = \min_i \frac{(w_i^j + t_j)}{s_i},$$

and h is the smallest machine index with this property.

3.2.1 LPT on consecutive identical machines

Proposition 3.1 *In* LPT *scheduling the following hold:*
(i) Let $s_i = s_{i+1}$. If t_j is the first job assigned to i, then t_{j+1} is the first job assigned to $i+1$.
(ii) Suppose that machine speeds are 2-divisible. If $s_h = s_i/2$, then h receives its first job after the first job of i and before the second job of i.

Proof. Suppose w.l.o.g., that $s_i = s_{i+1} = 1$.

(i) After t_j is assigned to i, machines of speed less than 1, and 1-machines to the right of $i+1$ are empty (they didn't get a job before i), 1-machines to the left of $i+1$ are non-empty (otherwise they would have received t_j). None of these machines is given a job before machine $i+1$. Now let us regard a fast machine i^*

of speed $s > 1$, with current finish time $\tilde{f}_{i^*}$. Since t_j was assigned to i and not to i^*, $\tilde{f}_{i^*} \geq t_j - t_j/s \geq t_{j+1} - t_{j+1}/s$. Therefore, t_{j+1} also prefers an empty 1-machine $i+1$ to machine i^*.

(ii) Now assume that the speeds are 2-divisible and let $s_h = 1/2$. It is obvious, that h doesn't receive a first job before i. Let t_j be the first job on i and t_{j+v} a later job. If h is empty, then t_{j+v} would have completion time $C(t_{j+v}) = 2t_{j+v}$ on h, whereas it would have completion time $C(t_{j+v}) = t_j + t_{j+v} \geq 2t_{j+v}$ on i. Since $h < i$, h gets a first job before the second job of i. □

Observe, that Proposition 3.1 (ii) holds only because LPT favours slower machines in case of ties. This feature of LPT is not essential, but it facilitates a shorter proof of Lemma 3.9, and saves us an even more intricate elaboration for case (2) in Section 3.4. However, both the monotonicity and the upper bound results do hold if LPT prefers higher index machines.

As we will point out there, one of the lower-bound examples in Section 3.7 is valid *only*, if ties are broken in favour of *faster* machines. The other two lower-bound instances are not sensitive to tie-breaking.

3.2.2 The principle of domination

In the upper bound proofs of Sections 3.3, 3.5.2 and 3.6 we will frequently apply the following simple tool, called *principle of domination* [37, 29]. In these proofs we consider a hypothetical *minimal* counter-example for an approximation upper bound of LPT. An instance is a **minimal** counter-example, if it has the smallest number of machines, and for this number of machines the smallest number of jobs. Let us consider the LPT schedule and some fixed optimal schedule OPT of an instance of the $Q||C_{\max}$ problem. Intuitively, some machine i dominates another machine i^* of the same (or larger) speed, if the jobs on i^* in OPT can be partitioned according to the jobs on i in LPT, so that each partition fits into its corresponding job. Formally:

Definition 3.2 [37] *We say that machine i* **dominates** *machine i^* if*
– $s_i \leq s_{i^*}$ *and*
– LPT *assigns the jobs $\tau_1, \ldots, \tau_k$ to i (disregarding t_n);* OPT *assigns the jobs $\tau_1^*, \ldots, \tau_l^*$ to i^* (t_n possibly included), and there is a function $F : \{\tau_1^*, \ldots, \tau_l^*\} \to \{\tau_1, \ldots, \tau_k\}$ such that for each τ_j, $\sum_{F(\tau_v^*)=\tau_j} \tau_v^* \leq \tau_j$.*

Proposition 3.3 (principle of domination [37]) *In a minimal counter-example for an approximation upper bound of* LPT*, no machine i dominates a machine i^*.*

Proof. Assume the contrary, that some machine i dominates a machine i^*. Omit from the instance the machine i and the jobs $\tau_1, \ldots, \tau_k$ that LPT assigned to i.

If we run LPT on the reduced instance, then each machine will get the same jobs as originally, and t_n cannot have smaller completion time than before, so the makespan did not decrease. However, the makespan of OPT does not increase, if we put all the jobs from i (as assigned by OPT) to i^*, and for all j, we put the jobs

from $F^{-1}(\tau_j)$ to where τ_j was previously assigned by OPT. So, we obtain a counter-example with less machines, contradicting the minimality of the original instance. □

Notice that as a corollary of the principle of domination, there are no empty machines in OPT.

3.2.3 The LPT* algorithm

We conclude this section with the definition of our monotone algorithm LPT*. In most of the chapter we will consider 2-divisible speed vectors. However, sometimes – e.g., when it comes to computing the payment function in Section 3.8 –, it is expedient to make a clear distinction between arbitrary input speed vectors and 2-divisible (rounded) speeds. In these rare cases $\langle \sigma_1, \ldots, \sigma_i, \ldots, \sigma_m \rangle \quad (\sigma_i \leq \sigma_{i+1})$ denotes the input speed vector of arbitrary positive speeds, and $\langle s_1, \ldots, s_i, \ldots, s_m \rangle$ denotes the rounded, 2-divisible speed vector.

LPT* **algorithm:** Input: $\langle \sigma_1, \ldots, \sigma_m \rangle$ and $\langle t_1, \ldots, t_n \rangle$

1. round the machine speeds down to $s_i := 2^{\lfloor \log \sigma_i \rfloor}$ $1 \leq i \leq m$ (the rounded speeds remain ordered);

2. run LPT on $\langle s_1, \ldots, s_m \rangle$ and $\langle t_1, t_2, \ldots, t_n \rangle$;

3. among machines of the same rounded speed, reorder the assigned work (i.e., the assigned sets of jobs), such that $w_i \leq w_{i+1}$ holds.

Clearly, LPT* runs in time $\mathcal{O}(m(n + \log m))$, since $\mathcal{O}(mn)$ time suffices to run LPT, and step 3. uses $\mathcal{O}(m \log m)$ time.

3.3 LPT* is a 3-approximation algorithm

The key result of this section is Theorem 3.3, claiming that LPT yields a 3/2-approximation on 2-divisible machines. We start the chapter with the proof of this theorem for two reasons: First, because this proof is much simpler than that of the 1.4-bound in Section 3.6. Second, and more importantly, since this proof serves as a perfect illustration of the primary method used in the more intricate monotonicity proof of Section 3.4.

We start by introducing some notation and making elementary observations. To get a contradiction, we will consider a minimal counter-example (see Sec. 3.2.2), for which the LPT schedule has more than 3/2-times the optimum makespan. In this input let the 2-divisible speed vector be $\langle s_1, s_2, \ldots, s_m \rangle$ and the job vector be $\langle t_1, t_2, \ldots, t_n \rangle$. Recall that $s_i \leq s_{i+1}$ and $t_j \geq t_{j+1}$.

Let OPT be any fixed optimal schedule of this input. *Opt* denotes the optimum makespan and *Lpt* denotes the makespan resulted by LPT on the above input. We will also use the short notation $\mu := Opt$.

As noted above, we assume that $Lpt > \frac{3}{2}\mu$. Let the last job $t := t_n$ be assigned to machine k in LPT. By the minimality of the instance, t has completion time *Lpt*,

i.e., t is a bottleneck job. (Note that jobs following a bottleneck job neither increase *Lpt*, nor decrease *Opt*.)

We denote by w_i^*, and $f_i^* = w_i^*/s_i$ the work and the finish time of machine i in OPT with respect to *all* the jobs. We denote by w_i, and $f_i = w_i/s_i$ the work and the finish time of machine i in LPT *before scheduling* t. Note that the last restriction has an influence only on f_k and w_k.

$Lpt > \frac{3}{2}\mu$ implies that for every machine i in LPT

$$f_i > \frac{3}{2}\mu - t/s_i. \tag{3.a}$$

We assume w.l.o.g. that the slowest machine in OPT has speed 1. Since machines in OPT are nonempty by the principle of domination, it follows that

$$t \leq \mu \cdot 1 = \mu. \tag{3.b}$$

The proof of Theorem 3.3 is based on the following simple technique: We strive for a contradiction by showing that the total work in LPT is strictly more than in OPT. First we show that only 1-machines may get more work in OPT than in LPT. Then we introduce the set of $\mathcal{P}$*-jobs*. These jobs are assigned to 1-machines in OPT, but they are larger than the jobs on 1-machines in LPT. Finally, in Lemma 3.9 we argue that the total work difference $\sum_{s_i>1}(w_i - w_i^*)$ on faster machines receiving the $\mathcal{P}$-jobs in LPT, exceeds the potential work difference on 1-machines $\sum_{s_i=1}(w_i^* - w_i)$, so that altogether more work is scheduled in LPT than in OPT.

Proposition 3.4 *(i) There exists at least one machine l s.t. $f_l < f_l^*$.*
(ii) Any such l is a 1-machine, and in LPT *at most one job is assigned to l.*
(iii)

$$\mu/2 < t. \tag{3.c}$$

Proof. (i) Since $t + \sum_i w_i = \sum_i w_i^*$, a machine l exists, s.t. $w_l < w_l^*$, i.e., $f_l < f_l^*$.

(ii) – (iii) For such a machine $f_l < f_l^* \leq \mu$ holds. According to (3.a), $\frac{3}{2}\mu - t/s_l < f_l < \mu$, i.e., $\mu/2 < t/s_l$. By (3.b), $t/s_l \leq \mu/s_l$, so we obtain $s_l < 2$. If $s_l < 1$ then $f_l^* = 0$. Consequently, $s_l = 1$ and $\mu/2 < t/s_l$ implies $\mu/2 < t$. Finally, the latter implies that at most one job fits to l below μ. □

Definition 3.5 *Let* $f_o := \max(t, \frac{3}{4}\mu)$.

Proposition 3.6 *On an arbitrary 1-machine i, there is at most* 1 *job in* OPT*, and it is larger than any job on a 1-machine in* LPT*. Moreover, on 1-machines $f_i \geq f_o$.*

Proof. (3.c) implies that there is at most 1 job on i in OPT, and it is larger than the jobs on 1-machines in LPT by the principle of domination.

By (3.a) and (3.b), $\frac{1}{2}\mu < f_i$, so there is at least 1 job on i in LPT. This fact together with (3.a) imply $f_i \geq \max(t, \frac{3}{2}\mu - t) \geq \frac{3}{4}\mu$. □

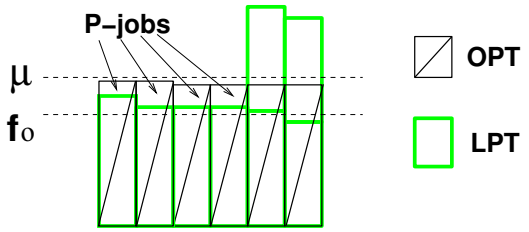

Figure 3.1: Lemma 3.8: Jobs on 1-machines in LPT and in OPT.

Definition 3.7 *Consider the 1-machines receiving only one job in* LPT. *Let* $p \geq 1$ *be the number of these machines, and* $\mathcal{P} = \{t_{j_1}, \ldots, t_{j_p}\}$ *be the set of jobs assigned to these machines in* OPT. *The jobs in* $\mathcal{P}$ *will be called* $\mathcal{P}$**-jobs**.

In Lemma 3.8 we upper bound $\sum_{s_i=1}(w_i^* - w_i)$. For an illustration see Fig. 3.1.

Lemma 3.8 *For the job set* $\mathcal{P} = \{t_{j_1}, \ldots, t_{j_p}\}$ *the following hold:*
(i) all of the jobs in $\mathcal{P}$ *are assigned to 1-machines in* OPT, *and to faster machines in* LPT*;*
(ii) $\frac{3}{4}\mu \leq f_o \leq t_{j_\tau} \leq \mu \qquad (1 \leq \tau \leq p)$;
(iii) $\sum_{s_i=1}(w_i^* - w_i) \leq \sum_{\tau=1}^{p}(t_{j_\tau} - f_o) \leq p \cdot \mu/4$.

Proof. Most of the statements are direct corollaries of Proposition 3.6:

(i) holds by the definition of $\mathcal{P}$-jobs and by the proposition.

(ii) Let t' be a single job on a 1-machine in LPT, and $t_{j_\tau} \in \mathcal{P}$ be the job on this machine in OPT, then $f_o \leq f_i = t' < t_{j_\tau}$.

(iii) trivially follows from (ii), since by Proposition 3.4, $f_i < f_i^*$, resp. $w_i < w_i^*$ is possible only on 1-machines with one job in LPT. □

Lemma 3.9 *In* LPT, *at most* $2^r - 1$ *of the* $\mathcal{P}$*-jobs are assigned to a machine* i *of speed* $s_i = 2^r$ $(r \geq 1)$. *If* $s_i \geq 4$ *then* $w_i - w_i^* \geq (2^r - 1) \cdot \mu/4$. *If* $s_i = 2$ *and* $\hat{t}$ *is the (only)* $\mathcal{P}$*-job assigned to* i, *then* $w_i - w_i^* \geq \hat{t} - f_o$.

Proof. While $\mathcal{P}$-jobs are being scheduled, 1-machines are still empty in LPT. Therefore, any $\mathcal{P}$-job t_{j_τ} has completion time $C(t_{j_\tau}) < t_{j_\tau}$, otherwise it would be assigned to a 1-machine. This implies that a machine of speed 2^r has less than 2^r $\mathcal{P}$-jobs.

Suppose that $s_i = 2^r \geq 4$. Then by (3.a) and (3.b), $f_i - f_i^* > \frac{3}{2}\mu - t/4 - \mu \geq \frac{3}{2}\mu - \mu/4 - \mu = \mu/4$. Consequently, $w_i - w_i^* > 2^r \cdot \mu/4$.

If $s_i = 2$, then (3.a) implies $w_i - w_i^* > 2 \cdot (\frac{3}{2}\mu - t/2 - \mu) = \mu - t$, and $\mu - t \geq \hat{t} - f_o$, since $\mu \geq \hat{t}$ by Lemma 3.8 (ii) and $t \leq f_o$ by Definition 3.5. □

Theorem 3.3 *Let* $\langle s_1, s_2, \ldots, s_m \rangle$ *be a 2-divisible speed vector, and* $\langle t_1, t_2, \ldots, t_n \rangle$ *be a fixed job vector. Let Opt be the optimum makespan, and Lpt be the makespan resulted by* LPT *on this input. Then* $Lpt \leq \frac{3}{2} \cdot Opt$.

Proof. Assume that the input is a minimal counter-example to the statement. If $w_i^* \geq w_i$ then $s_i = 1$ by Proposition 3.4. The work difference on 1-machines

$\sum_{s_i=1}(w_i^* - w_i)$ is at most $\sum_{\tau=1}^{p}(t_{j_\tau} - f_o)$ by Lemma 3.8, where the $\{t_{j_\tau}\}_{\tau=1}^{p}$ denote the $\mathcal{P}$-jobs.

According to Lemma 3.9, this difference is balanced out on faster machines, that receive the $\mathcal{P}$-jobs in LPT: On machines of speed 2 we obtained the lower bound on the difference $w_i - w_i^* \geq \hat{t} - f_o$, where $\hat{t}$ is the only $\mathcal{P}$-job assigned to the machine; on faster machines we obtained the lower bound $w_i - w_i^* \geq (2^r - 1) \cdot \mu/4$, where there are at most $2^r - 1$ $\mathcal{P}$-jobs assigned to the machine, and $\mu/4 > (t_{j_\tau} - f_o)$ for any t_{j_τ}. This implies

$$\sum_{s_i=1}(w_i^* - w_i) \leq \sum_{\tau=1}^{p}(t_{j_\tau} - f_o) \leq \sum_{s_i \neq 1}(w_i - w_i^*),$$

so that

$$\sum_{i=1}^{m} w_i^* \leq \sum_{i=1}^{m} w_i$$

a contradiction, since $t + \sum_i w_i = \sum_i w_i^*$. □

Corollary 3.10 LPT* *is a 3-approximation algorithm.*

Proof. Suppose that on input $\langle t_1, t_2, \ldots, t_n \rangle$

Opt is the optimum makespan at speed vector $\langle \sigma_1, \ldots, \sigma_m \rangle$;

Opt′ is the optimum makespan at speed vector $\langle s_1, \ldots, s_m \rangle$;

Lpt is the makespan provided by LPT at $\langle s_1, \ldots, s_m \rangle$;

*Lpt** is the makespan provided by LPT* at $\langle \sigma_1, \ldots, \sigma_m \rangle$,

then $Lpt^* \leq Lpt \leq \frac{3}{2} \cdot Opt' \leq \frac{3}{2} \cdot (2 \cdot Opt)$. The first and last inequalities follow from the fact that machine speeds are increased, resp. decreased by a factor between 1 and 2. Finally, $Lpt \leq \frac{3}{2} \cdot Opt'$ holds according to Theorem 3.3. □

3.4 LPT* is monotone

This section is devoted to the primary result of Chapter 3, stating that LPT scheduling is monotone on 2-divisible machines. After fixing notation and presenting the theorem, we provide some intuition about the proof in Section 3.4.1.

Suppose that in LPT schedule I the 2-divisible input speed vector contains one more copies of speed 1/2 and one less copies of speed 1 than in LPT schedule II, and otherwise the inputs of I and II are the same. Let k be *any* machine of speed 1/2 in I, and k' be *any* machine of speed 1 in II. Theorem 3.4 claims that machine k receives not more work in schedule I than machine k' in schedule II. Let $s_1 \leq s_2 \leq \ldots \leq s_m$ denote the machine speeds in I. We will view schedule II like this: in schedule II, the speed $s_k = 1/2$ of machine k is increased to $s'_k = 1$ while the speeds (and relative order) of other machines remain unchanged. We will refer to machine i in schedule II by i'. Clearly, in general k' is not the kth machine, and i' is not necessarily the ith machine in II (see, e.g., the machines k and k' in Fig. 3.4).

We classify the unchanged machines into three categories:

Definition 3.11 *Let $i \neq k$. We say that i is a* **slow machine** *if $s_i < 1/2$, a* **medium machine** *if $1/2 \leq s_i \leq 1$, and a* **fast machine** *if $1 < s_i$. We call a slow machine* **tardy***, if it has the speed of the slowest nonempty machines in II.*

In I and in II the machines receive the same job sequence $t_1 \geq \ldots \geq t_n$. If t_j is assigned to machine i, it has *(time)length* t_j/s_i. The completion time of t_j in I resp. in II is $C(t_j)$, resp. $C'(t_j)$. We denote by w_i and f_i the work and finish time of machine i in schedule I. For ease of use, w'_i and f'_i denote the respective values in schedule II (instead of, e.g., $w'_{i'}$ and $f'_{i'}$).

Theorem 3.4 *Let the* LPT *schedules I and II, machines k and k', furthermore the respective total works w_k and w'_k be as defined above, then $w_k \leq w'_k$.*

Theorem 3.5 LPT* *is monotone.*

Proof. Suppose that in the input of LPT*, the speed σ_i is increased to σ'_i and everything else remains unchanged. Then the index of speed σ'_i in the (re)ordered input speed vector is at least i. If $s_i = s'_i$, then step 3. of LPT* implies that i receives not less work with increased speed. If $s_i < s'_i$, then a repeated application of Theorem 3.4 implies Theorem 3.5. □

3.4.1 The proof of Theorem 3.4.

In this subsection we prove Theorem 3.4. The proof is by contradiction: we assume $w_k > w'_k$. Let $W_{\neq k} := \sum_{i \neq k} w_i$ and $W'_{\neq k} := \sum_{i \neq k} w'_i$. In most subcases of the proof we show $W_{\neq k} \geq W'_{\neq k}$, contradicting $w_k > w'_k$. In the remaining subcases we get a contradiction by proving that the number of assigned jobs is strictly larger in schedule I than in schedule II.

Let t_a be the first job assigned to k' in II. We will call a machine **dead**, if it receives no job after t_a in II, and we call it **living** otherwise. Notice that right before job t_a is scheduled, the schedules I and II are exactly the same. Therefore, on dead machines $w \geq w'$, and there are at least as many jobs on the machine in I as in II. Unless it is necessary to mention dead machines explicitly, we concentrate on living machines.

Let $t := t_n$ denote the last job. We can assume that w_k becomes larger than w'_k only after job t; job t is assigned to k in I, but it is not assigned to k' in II.

Let $t_a = t_{a_1} \geq t_{a_2} \geq t_{a_3} \geq \ldots$ be the jobs assigned to k'. It facilitates a more handy proof if we normalize job sizes so that $t_a = 1$. We can do this without loss of generality. Consequently, the length of t_a on k' is $t_a/s'_k = 1/1 = 1$.

Because the finish time f'_k of k' plays a central role in our comparisons, we provide it with special notation: **let** $\lambda := f'_k$ **be the finish time of** k'. That is, $\lambda = w'_k/1$. Since $w_k > w'_k$, for the finish time of k in I, $f_k = w_k/\frac{1}{2} > w'_k/\frac{1}{2} = 2\lambda$ holds. Moreover, in I a machine $i \neq k$ of speed 2^r ($r \in \mathbb{Z}$) has finish time

$$f_i > 2\lambda - \frac{t}{2^r}, \tag{3.d}$$

otherwise this machine (and not k) would receive the last job t.

Let W_T and W'_T denote the total work on tardy machines in schedule I and II, respectively. We provide some intuition about the first part of our proof. This part follows the same lines as the proof of Theorem 3.3. First, we show that if $w < w'$ on a slow machine, then it must be a tardy machine. After that, we prove that *if* $w \geq w'$ *on every medium and fast machine*, then $W_{\neq k} \geq W'_{\neq k}$. Luckily, this is the case if at least 2 jobs are assigned to k' in II (case (1)). How do we show $W_{\neq k} \geq W'_{\neq k}$? In Lemma 3.15 we introduce the set of $\mathcal{P}$-jobs. These are very small jobs, that *follow* the jobs of tardy machines in II, but are still put on tardy machines in I. In the same lemma an upper bound on $W'_T - W_T$ is stated in terms of the number p of $\mathcal{P}$-jobs. In Lemma 3.19 it is shown that the difference $W'_T - W_T$ is balanced out on non-tardy machines, that receive the $\mathcal{P}$-jobs in II. In case (2), the same argument about $\mathcal{P}$-jobs is used parallel to other techniques, in order to show $W_{\neq k} \geq W'_{\neq k}$.

First, we provide upper bounds on the finish time of living machines in II:

Proposition 3.12 *In schedule II,* $f'_i \leq \max(2, 3\lambda/2)$ *for any living machine* i'.

Proof. Let $f'_{\max}$ be the maximum finish time among all *living* machines in II. Note that this maximum is realized by a bottleneck job t_β not before $t_a = t_{a_1}$. Let first $a_1 \leq \beta < a_2$. Then $f'_{\max} \leq 2t_a = 2$, otherwise k' would have received t_β. Second, suppose $a_\eta \leq \beta < a_{\eta+1} \quad (\eta > 1)$. Then $f'_{\max} \leq t_{a_1} + t_{a_2} + \ldots + t_{a_\eta} + t_{a_\eta} \leq \lambda + t_{a_\eta} \leq \lambda + \lambda/2 = 3\lambda/2$. □

Proposition 3.13 *In* LPT *schedule II,*
(i) If t_j *is a job on a slow machine* i'*, then* $t \leq t_j \leq \lambda/3$*, and* $f'_i \leq 4\lambda/3$.
(ii) If t_j *is the* 2*nd job on a 1/2-machine* i'*, then* $t \leq t_j \leq \lambda/3$*, and* $f'_i \leq 4\lambda/3$.
(iii) If t_j *is the* 3*rd job on a 1-machine* i'*, then* $t \leq t_j \leq \lambda/2$*, and* $f'_i \leq 3\lambda/2$.
In cases (ii) and (iii) $f'_i \leq f_i$.

Proof. (i) – (ii) $4t_j \leq f'_i$, because in (i) $s_i \leq 1/4$, in (ii) $s_i = 1/2$ and t_j is the 2nd job. Let t_{j+v} $(v \geq 0)$ be the last job on i'. Since t_{j+v} is not assigned to k', it follows that $4t_j \leq f'_i \leq \lambda + t_{j+v} \leq \lambda + t_j$, and therefore $t_j \leq \lambda/3$ and $f'_i \leq 4\lambda/3$.

(iii) If there are at least 3 jobs assigned to a 1-machine in II, then the second job has completion time at most λ, otherwise the third job would be assigned to k'. So the second and further jobs have size at most $\lambda/2$. Let t_{j+v} $(v \geq 0)$ be the last job on i'. Since t_{j+v} is not assigned to k', it follows that $f'_i \leq \lambda + t_{j+v} \leq 3\lambda/2$.

Finally, in (ii) and (iii) $f'_i \leq f_i$ is implied by (3.d). □

Proposition 3.14 *Suppose that* $w_i < w'_i$ *holds for a slow machine* i *of speed* $s_i = 1/2^l$ $(l \geq 2)$*. Then* i *is a tardy machine, and each tardy machine receives at most* 1 *job in schedule II.*

Proof. We show that in II there is at most one job assigned to any machine of speed $1/2^l$, and machines of speed $1/2^{l+1}$ are empty. Suppose that there is a job t_j assigned to a machine of speed $1/2^l$ as a second job, or assigned to a strictly slower machine in

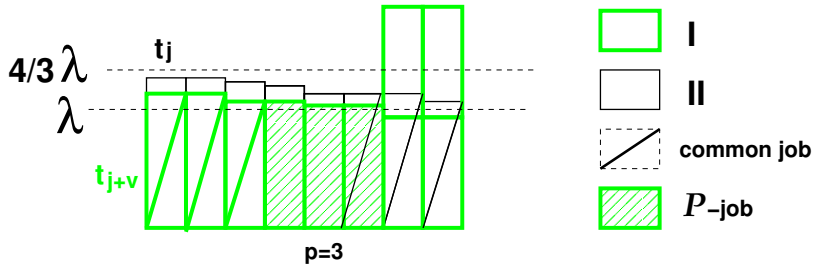

Figure 3.2: Lemma 3.15: Jobs on tardy machines in schedules I and II.

II. According to Proposition 3.13 (i), in either case $2^{l+1}t_j \leq 4\lambda/3$ holds. Moreover, it follows that $2^l t \leq 2^l t_j \leq 2\lambda/3$. Now, (3.d) implies that $f_i \geq 2\lambda - 2^l t \geq 4\lambda/3 \geq f'_i$, contradicting $w'_i > w_i$. □

In Lemma 3.15 we upper bound $W'_T - W_T$. For an illustration to the lemma see Figure 3.2.

Lemma 3.15 *Suppose that $w_i < w'_i$ holds for a tardy machine i, and tardy machines have speed $1/2^d$. Let t_j be the first job assigned to tardy machines in II. There is a set of jobs $\mathcal{P} = \{t_J, t_{J+1}, \ldots, t_{J+p-1}\}$ for some $p \geq 0$, so that*
(i) in I all these p jobs are assigned to tardy machines, and in II all these p jobs are assigned to faster than tardy machines;
(ii) $\lambda/2^d \leq t_{J+\zeta} < t_j \leq \frac{4}{3}\lambda/2^d \quad (0 \leq \zeta \leq p-1)$;
(iii) $W'_T - W_T \leq p \cdot (\frac{4}{3}\lambda/2^d - t) \leq p \cdot (\frac{1}{3}\lambda - t)$.

Proof. According to Proposition 3.14, in II there is at most 1 job on each tardy machine. We claim that in I, there is at least 1 job on each tardy machine. Note that job t_j is the first job assigned to the leftmost tardy machine in II. If f' denotes the finish time of the leftmost tardy machine, then using Proposition 3.13 (i), $2^d t \leq 2^d t_j \leq f' \leq \frac{4}{3}\lambda$. Now the claim follows from (3.d), since for a tardy machine $f > 2\lambda - 2^d t \geq 2\lambda - \frac{4}{3}\lambda > 0$.

By Proposition 3.1, and since each tardy machine has at most 1 job in II, $w_i < w'_i$ is possible only if II starts to fill tardy machines earlier than I. Having y tardy machines, $\{t_j, t_{j+1}, \ldots, t_{j+y-1}\}$ are the only jobs on tardy machines in schedule II; whereas there is a series of y consecutive jobs $\{t_{j+v}, t_{j+v+1}, \ldots, t_{j+v+y-1}\}$ on tardy machines in schedule I.

W.l.o.g. we assume $v < y$. We omit the common jobs $\{t_{j+v}, t_{j+v+1}, \ldots, t_{j+y-1}\}$ from II and I. We also omit all possibly remaining (single) jobs of size t_j from I, and the same number of jobs from II (these jobs in II then all have size t_j). Finally, we omit tardy machines having at least 2 jobs in I, with all their jobs, and the same number of machines together with their job from II. Modulo jobs of equal size, we may assume that tardy machines with ≥ 2 jobs in I are the rightmost machines, so that the remaining jobs on tardy machines in I are consecutive: $\{t_{j+y}, \ldots, t_{j+y+p-1}\}$. Let $\mathcal{P}$ be the set of these jobs, that is, $J := j + y$.

Now (i) obviously holds.

(ii) We have seen that $t_{j+y} < t_j \leq \frac{4}{3}\lambda/2^d$. Moreover, a tardy machine in I has finish time $f \geq \max(2^d t, 2\lambda - 2^d t) \geq \lambda$. Since each $\mathcal{P}$-job is alone on a tardy machine in I, each has size $\geq \lambda/2^d$.

(iii) For a tardy machine with at least 2 jobs in I, $f \geq \max(2\cdot 2^d t, 2\lambda - 2^d t) \geq \frac{4}{3}\lambda \geq 2^d t_j$ holds. Consequently, we omitted at least as much total job size from schedule I as from schedule II. Thus, the total work difference on tardy machines is not more than the remaining work difference: $W'_T - W_T \leq \sum_{\zeta=0}^{p-1}(t_j - t_{J+\zeta}) \leq p \cdot (\frac{4}{3}\lambda/2^d - t)$. □

Definition 3.16 *The jobs in the set $\mathcal{P}$ of Lemma 3.15 will be called* **$\mathcal{P}$-jobs**.

In Propositions 3.17 and 3.18 we examine, how many $\mathcal{P}$-jobs can be assigned to a non-tardy machine in II.

Proposition 3.17 *Let $\tilde{f}'_i$ be the finish time of a machine i' after the first $\mathcal{P}$-job is assigned to i' in II. Then $\tilde{f}'_i > \lambda$. In particular, there are no $\mathcal{P}$-jobs on k'.*

Proof. Let $1/2^d$ be the speed of tardy machines, and let $s_i = 2^r \quad (-d < r)$ be the speed of i'. Let t_j be the first job assigned to a tardy machine in II. Since t_j is not assigned to i', it follows that just before t_j is scheduled, i' has finish time $\geq 2^d t_j - t_j/2^r$. Let $t_{J+\zeta}$ be the $\mathcal{P}$-job assigned to machine i'. Now we use Lemma 3.15 (ii). After scheduling $t_{J+\zeta}$, machine i' has finish time $\tilde{f}'_i \geq 2^d t_j - t_j/2^r + t_{J+\zeta}/2^r > 2^d t_{J+\zeta} \geq \lambda$. For the last inequality observe that $t_j(2^d - 1/2^r) > t_{J+\zeta}(2^d - 1/2^r)$. This follows from $t_j > t_{J+\zeta}$ and from $1/2^d < 2^r$. □

Proposition 3.18 *Suppose that a machine i' of speed $s_i = 2^r$ ($r \in \mathbb{Z}$) has finish time $\tilde{f}'_i > \lambda$ in some step T of* LPT *II. If $r \geq 1$, then there are at most $2^r - 1$ jobs assigned to i' after T; if $r \leq 0$, then there are no further jobs assigned to i'.*

Proof. The second statement is easy to see: the finish time of k' at any time step is at most λ, so k' would receive a job before i' of speed $s_i \leq 1$. The proof of the first statement is similar: *after* T, the 2^rth job $\hat{t}$ assigned to i' would have completion time $C'(\hat{t}) > \lambda + 2^r\hat{t}/2^r = \lambda + \hat{t}$, but assigned to k', it would have completion time $C'(\hat{t}) \leq \lambda + \hat{t}$, a contradiction. □

Lemma 3.19 *Suppose that at least one $\mathcal{P}$-job is assigned to some machine $i' \neq k'$.*
(i) If i' is a fast machine of speed $s_i = 2^r \quad (r \geq 1)$, then it receives at most 2^r $\mathcal{P}$-jobs and $w_i - w'_i > 2^r(\frac{1}{3}\lambda - t)$. Furthermore, $w'_i \leq 2^r\frac{4}{3}\lambda$.
(ii) If i' is a medium or slow (non-tardy) machine, then it receives 1 $\mathcal{P}$-job, and $w_i - w'_i > \frac{4}{3}\lambda/2^d - t$, where the speed of tardy machines is $1/2^d$.

Proof. The possible number of $\mathcal{P}$-jobs on i' follows from Propositions 3.17 and 3.18.

We claim that in case (i), $w_i > 2^r\frac{5}{3}\lambda; \quad w'_i \leq 2^r\frac{4}{3}\lambda$, so $w_i - w'_i > 2^r\frac{1}{3}\lambda > 2^r(\frac{1}{3}\lambda - t)$. In (ii), if $s_i = 1/2^l \quad (l \geq 0)$, then $w_i > 2\lambda/2^l - t; \quad w'_i \leq \frac{4}{3}\lambda/2^l$, so $w_i - w'_i > \frac{2}{3}\lambda/2^l - t \geq \frac{4}{3}\lambda/2^d - t$.

We prove the upper bounds on w'_i and the lower bounds on w_i :

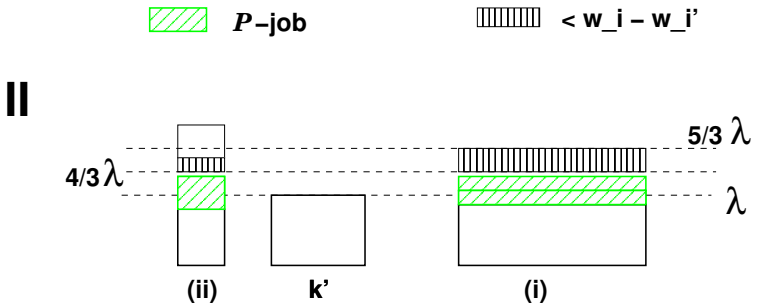

Figure 3.3: Lemma 3.19: A fast (i) and a medium (ii) machine receiving $\mathcal{P}$-jobs in schedule II.

A $\mathcal{P}$-job has size $t_{J+\zeta} \leq \lambda/3$ by Proposition 3.13 (i). Since $t_{J+\zeta}$ is not assigned to k', it has completion time $C'(t_{J+\zeta}) \leq \lambda + t_{J+\zeta} \leq \frac{4}{3}\lambda$. The same holds for any job following $\mathcal{P}$-jobs. So, the upper bounds on w_i' follow.

Finally, the lower bounds on w_i follow from (3.d) and from $t \leq \lambda/3$: in case (i) $w_i = 2^r \cdot f_i$, and $f_i > 2\lambda - t/2^r > 2\lambda - t \geq 2\lambda - \lambda/3 = 5\lambda/3$; in (ii) $w_i = f_i/2^l > (2\lambda - t \cdot 2^l)/2^l = 2\lambda/2^l - t$. □

Corollary 3.20 *If $w_i \geq w_i'$ for every medium and fast machine, then $W_{\neq k} \geq W'_{\neq k}$.*

Proof. By Proposition 3.14, $w_i \geq w_i'$ on every non-tardy machine. There are $p \geq 0$ $\mathcal{P}$-jobs on tardy machines in I. Let $i'_1, i'_2, \ldots, i'_\xi$ be the machines with at least one $\mathcal{P}$-job in II. By Lemmas 3.15 and 3.19,

$$W'_T - W_T \leq p \cdot (\frac{4}{3}\lambda/2^d - t) \leq \sum_{\tau=1}^{\xi} (w_{i_\tau} - w'_{i_\tau}).$$

To sum up, the potential total difference $W'_T - W_T$ on tardy machines is compensated for on non-tardy machines receiving $\mathcal{P}$-jobs in II, and $W_{\neq k} \geq W'_{\neq k}$ follows. □

(1) Assume that in schedule II at least 2 jobs are assigned to machine k'.

Lemma 3.21 *If there are at least 2 jobs assigned to k', then $W_{\neq k} \geq W'_{\neq k}$.*

Proof. We show that if i is a fast or medium machine then $f_i \geq f_i'$, that is, $w_i \geq w_i'$. Combining this with Corollary 3.20 yields the lemma.

(i) First, we claim that $f_i > 2$ on every machine i of speed $\geq 1/2$ in I. Since t is the last job, $w'_k \geq t_{a_1} + t_{a_2} \geq 1 + t$. In schedule I, let $\tilde{w}_k$ and $\tilde{f}_k$ be the work and finish time of k *before the last step*. From $\tilde{w}_k = w_k - t > w'_k - t \geq 1$ we obtain $\tilde{f}_k > 2$. If there were a machine i of $f_i \leq 2$ and $s_i \geq 1/2$, then machine i would receive t and not machine k.

(ii) Second, we claim that $f_i > 3\lambda/2$ on every machine of speed ≥ 1 in I. Note that $t \leq \lambda/2$ holds, because $\lambda = w'_k \geq 1 + t$. Moreover, (3.d) implies $f_i > 2\lambda - t/2^r \geq 2\lambda - t \geq 3\lambda/2$ if $r \geq 0$.

Now (i), (ii) and Proposition 3.12 imply the statement of the Lemma, unless i has speed $1/2$. Let $s_i = 1/2$. If there is only one job t_j assigned to i' in II, then

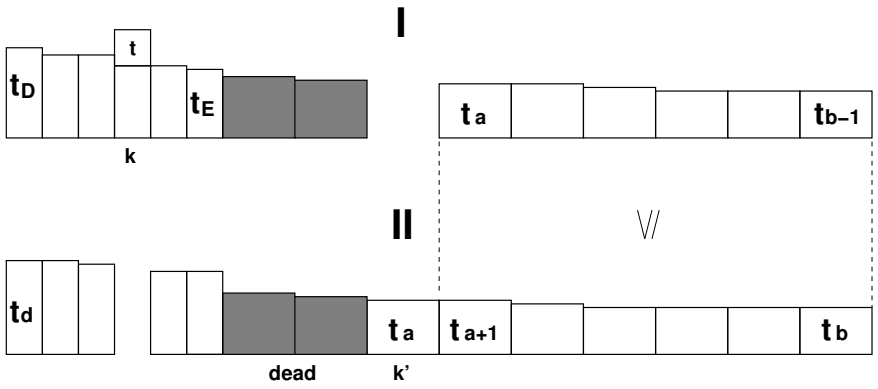

Figure 3.4: Definition 3.22: The (first) jobs on 1/2-machines and 1-machines in I and II.

$t_j \leq t_a = 1$, since $s_i < s'_k$, and t_a is the first job on k'. Consequently, $f'_i \leq 2$, which together with (i) yields $f_i \geq f'_i$. If there are at least two jobs assigned to i', then according to Proposition 3.13, $f_i \geq f'_i$. □

(2) Assume that in schedule II only job t_a is assigned to machine k'.

In case (2) the proof is more involved, and consists of further subcases. As mentioned before, our goal is to get a contradiction by showing either $W_{\neq k} \geq W'_{\neq k}$, or that I has strictly more jobs than II. In the general part we introduce further notation and derive necessary conditions for $w < w'$ on a medium machine. As a side effect, this will prove the theorem if there are no fast machines. Recall that $t_a = 1$, so $\lambda = f'_k = w'_k/1 = t_a/1 = 1$.

Definition 3.22 *Let $t_a, t_{a+1}, \ldots, t_b$ $(a \leq b)$ be the first jobs assigned to 1-machines to the right of k' in II. Let T_b be the time step, before t_b is scheduled.*

Let t_D be the first job on the leftmost 1/2-machine, and t_E be the first job on the rightmost 1/2-machine in I. Finally, t_d denotes the first job assigned to medium machines after $T_b + 1$ in II, if such a job exists (see Figure 3.4).

According to Proposition 3.1, the jobs $t_{a+1}, \ldots, t_b$ are well-defined. At time T_b schedules I and II are the same, except that jobs on 1-machines are shifted due to machine k'; 1/2-machines and slow machines are empty. Observe that $D \leq E$, and if 1/2-machines exist in II, then t_d is the first job on the leftmost 1/2-machine. If job t_d does not exist at all, then obviously $w \geq w'$ holds for all medium machines.

Proposition 3.23 *In schedule I, there are at least 3 jobs and strictly more work than $1 + t_E$ assigned to each living 1-machine.*

Proof. Recall, that k is a 1/2-machine, and it receives at least two jobs: the last job t and a job of size $\geq t_E$. Thus, $w_k \geq t_E + t$.

Let i be a living 1-machine. Note that i has a first job of size ≤ 1 in both schedules. By Proposition 3.1, a potential second job on i has size $\leq t_E$ in schedule I. Suppose that $w_i \leq 1 + t_E$. Then $w_i \leq 1 + w_k - t < 2w_k - t$. The last inequality

follows from $w_k > w'_k = 1$. Therefore, if t were assigned to i instead of k, then $C(t) = (w_i + t)/1 < 2w_k = w_k/\frac{1}{2} = f_k$ would hold, a contradiction. Consequently, $w_i > 1 + t_E$, and at least 3 jobs are assigned to i. □

Proposition 3.24 *If $w_i < w'_i$ for a medium machine i, then $d < D$ and $t_d > 1 - t$.*

Proof. To get a contradiction, assume that $d \geq D$, that is, schedule II starts to fill 1/2-machines not before schedule I. Let t_g $(d \leq g)$ be the last job on i'. If i' is a 1-machine, then t_g is the second job on i' by Proposition 3.13. Moreover, $d \geq D$ implies $g \geq E$ (see Fig. 3.4). By Proposition 3.23, $w_i > 1 + t_E \geq 1 + t_g = w'_i$, a contradiction.

If i' is a 1/2-machine, then by Proposition 3.13, t_g is the only job on i'. If $i < k$, then $w_i < w'_i$ is impossible, since II starts to fill machines not before I. If $k < i$, then a $v \geq 0$ exists, s.t. t_{g-v} is the first job on k in I. Since t is not assigned to i, $w_i \geq w_k - t \geq t_{g-v} \geq t_g = w'_i$, a contradiction.

It remains to prove that $t_d > 1 - t$. Due to (3.d), $w'_i > w_i > 2 - t$ if i is a 1-machine, and $w'_i > w_i > 1 - t$ if i is a 1/2-machine. Consequently, $1 - t < t_g$ holds in both cases. Now the statement follows from $t_g \leq t_d$. □

Corollary 3.25 *If there are no fast machines, then $W_{\neq k} \geq W'_{\neq k}$.*

Proof. Recall that in II there is at least one 1-machine, and in I there is at least one 1/2-machine. If there are no fast machines, then according to Proposition 3.1, $b = D$ must hold (see Fig. 3.4). Therefore, $d > b = D$, and Proposition 3.24 implies $w_i \geq w'_i$ on every medium machine i. Finally, Corollary 3.20 yields the proof. □

We end the general part of case (2), with a technical proposition:

Proposition 3.26 *Suppose that in case (2), in schedule II there is a job $\bar{t} < 1$ on a machine $\bar{i}$ of speed $s_{\bar{i}} = 2^u$ $(u \geq 1)$. Then in schedule II*
(i) every job t^* has completion time $C'(t^*) \leq 1 + \bar{t}(1 - 1/2^u) + t^*/2^u$;
(ii) every living machine has finish time at most $f' \leq 3/2 + \bar{t}/2$.

Proof. (i) The job $\bar{t}$ has completion time $C'(\bar{t}) \leq 1 + \bar{t}$, otherwise $\bar{t}$ would have been scheduled to k'. If job t^* is scheduled before $\bar{t}$, it has no larger completion time than it would have on $\bar{i}$, that is $C'(t^*) \leq 1 + \bar{t} - \bar{t}/2^u + t^*/2^u = 1 + \bar{t}(1 - 1/2^u) + t^*/2^u$. If t^* is scheduled after $\bar{t}$, then $C'(t^*) \leq 1 + t^* \leq 1 + \bar{t}(1 - 1/2^u) + t^*/2^u$. The last inequality follows from $t^*(1 - 1/2^u) \leq \bar{t}(1 - 1/2^u)$.

(ii) If t^* is the last job on a living machine, then $t^* \leq t_a = 1$, so the finish time of the machine is at most $f' \leq 1 + \bar{t}(1 - 1/2^u) + 1/2^u = 1 + \bar{t} + (1 - \bar{t})1/2^u \leq 1 + \bar{t} + (1 - \bar{t})1/2 = 3/2 + \bar{t}/2$. □

(2.1) Assume that $t > 1/3$.

If $t > 1/3$, then Proposition 3.13 (i) implies that all the slow machines are empty in II. Regarding only medium and fast machines, we will show that if $i \neq k$, then the number of jobs assigned to i by I is not smaller than the number of jobs assigned to i' by II; whereas schedule I assigns strictly more jobs to machine k, than schedule II to k', so we get a contradiction.

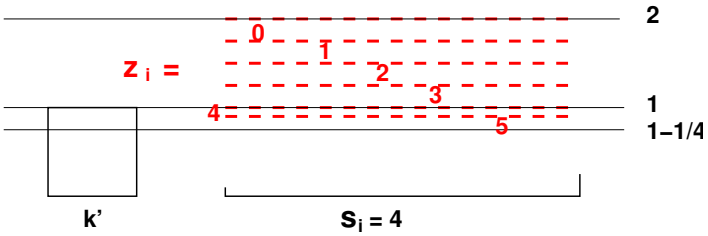

Figure 3.5: Definition 3.28: Zones of a machine of speed 4.

Definition 3.27 *Let i be a fast machine. f_i^b denotes the common finish time of i and i' at time T_b.*

Note that if $s_i = 2^r$ $(r \geq 1)$, then $f_i^b \geq 1 - 1/2^r$, otherwise t_a would be assigned to i'. For the (final) finish time in II, $f_i' \leq 2$ holds by Proposition 3.12. Now we partition the time interval $(1, 2]$ into 2^r equal zones of length $1/2^r$, and partition $(1 - 1/2^r, 1]$ into two further zones (see Figure 3.5):

Definition 3.28 *For a fast machine i of speed $s_i = 2^r$ we define the* **zone** z_i *as*

$$z_i = \begin{cases} 2^r + 1 & \text{if} \quad 1 - 1/2^r \leq f_i^b \leq 1 - t/2^r \\ 2^r & \text{if} \quad 1 - t/2^r < f_i^b \leq 1 \\ z & \text{if} \quad 2 - (z+1)/2^r < f_i^b \leq 2 - z/2^r \quad (0 \leq z \leq 2^r - 1). \end{cases}$$

We also say that f_i^b is **in the z_ith zone**.

Proposition 3.29 *After T_b, in schedule II there are at most $2^r + 1$ further jobs assigned to any fast machine i' of speed 2^r $(r \geq 1)$.*

Proof. Let $\hat{t}$ be the *second* job that is put to i' after T_b. We show that $C'(\hat{t}\,) > 1 = \lambda$. Then by Proposition 3.18, II assigns at most $2^r - 1$ further jobs to i' after $\hat{t}$, and this proves our proposition, since $2 + 2^r - 1 = 2^r + 1$.

Suppose first, that in schedule II every job on fast machines is larger than $1/2$. Then $C'(\hat{t}\,) \geq f_i^b + 2 \cdot \hat{t}/2^r > 1 - 1/2^r + (2 \cdot \frac{1}{2})/2^r = 1$.

Second, suppose that in II there is a job $\bar{t} \leq 1/2$ on a machine $\bar{i}$ of speed $s_{\bar{i}} = 2^u$ $(u \geq 1)$. We show that for every fast machine $w \geq w'$ holds. By Proposition 3.26 (ii), for any living machine $f' \leq 3/2 + 1/2 \cdot 1/2 \leq 7/4$. On the other hand, on fast machines in I, (3.d) implies that $f > 2 - t/2 \geq 2 - 1/2 \cdot 1/2 = 7/4$.

Since $w_k > w_k'$ holds, and slow machines are empty, there must be a medium machine for which $w < w'$ holds. Therefore, by Proposition 3.24, $t_d > 1 - t \geq 1/2$. Recall that t_d is the first job on a $1/2$-machine, or the second job on a 1-machine, so $C'(t_d) \geq 2t_d$.

If a job is assigned to i' after T_b and before t_d, then this job has size $> 1 - t$, so $C'(\hat{t}\,) > f_i^b + (1-t)/2^r + t/2^r \geq 1$.

On the other hand, suppose that before scheduling t_d, the finish time of i' is still f_i^b. Since t_d is not assigned to i', it follows that $f_i^b + t_d/2^r \geq 2t_d$, i.e., $f_i^b \geq t_d(2 - 1/2^r) > 1/2 \cdot (2 - 1/2^r) = 1 - 1/2 \cdot 1/2^r$. So, $C'(\hat{t}\,) > f_i^b + 2/3 \cdot 1/2^r > 1$. □

Proposition 3.30 *Let i be a fast machine of speed 2^r. If f_i^b is in the z_ith zone, then after T_b, i gets at least z_i jobs in I, and i' gets at most z_i jobs in II.*

Proof. Let us first consider schedule I. If $z_i \leq 2^r$, then $f_i - f_i^b > 2 - t/2^r - (2 - z_i \cdot 1/2^r) \geq (z_i - 1) \cdot 1/2^r = (z_i - 1) \cdot 1/s_i$. If $z_i = 2^r + 1$ then $f_i - f_i^b > 2 - t/2^r - (1 - t/2^r) = 1 = 2^r \cdot 1/2^r$. Since after T_b, jobs have size at most 1, schedule I must assign at least z_i jobs to i after T_b.

Now consider schedule II. If $z_i \leq 2^r - 1$, then $f_i^b > 2 - (z_i + 1) \cdot 1/2^r = 1 + (2^r - (z_i + 1)) \cdot 1/2^r$. Suppose that at least $(z_i + 1)$ more jobs are assigned to i' in II, and the last of these is $\hat{t} \leq 1$. Then i' has finish time $f_i' \geq f_i^b + (z_i + 1) \cdot \hat{t}/2^r > 1 + (2^r - (z_i + 1) + (z_i + 1))\hat{t} \cdot 1/2^r = 1 + \hat{t}$, where the last inequality exploits $2^r \geq z_i + 1$ and $1 \geq \hat{t}$. But this is impossible, because $\hat{t}$ would have been assigned to k'.

If $z_i = 2^r$ then $f_i^b > 1 - t/2^r$. The completion time of the next job exceeds $1 = \lambda$, and according to Proposition 3.18, II assigns $\leq 2^r - 1$ further jobs to i'. If $z_i = 2^r + 1$, then after T_b II assigns at most z_i jobs to i' by Proposition 3.29. □

Lemma 3.31 *If $t > 1/3$, then i receives at least as many jobs in schedule I as i' in schedule II; k receives at least 2 jobs in I, and k' receives only 1 job in II.*

Proof. (i) To machine k, schedule I assigns at least 2 jobs, otherwise $w_k > w_k'$ is impossible; on the other hand, II assigns only job t_a to k'.

(ii) To a 1/2-machine, II assigns at most 1 job by Proposition 3.13 (ii), since $t > \frac{1}{3} = \frac{\lambda}{3}$; schedule I assigns at least 1 job, otherwise k would not receive a second job.

(iii) To a 1-machine i, II assigns at most 3 jobs, because the completion time of the 3rd job exceeds $3 \cdot 1/3 = 1$, and any further job would prefer k' to i'. According to Proposition 3.23, schedule I assigns at least 3 jobs to a living 1-machine.

(iv) Proposition 3.30 proves the lemma for fast machines. □

(2.2) Assume that $t \leq 1/3$.

Case (2.2) is divided into three subcases, depending on the size of the smallest job assigned to any fast machine. Although this part is lengthy and intricate, it applies combinations of the two basic types of arguments, i.e., a comparison of the total work or of the number of jobs on fast and medium machines. In the rest of the proof W_F, W_M, and W_S denote the total work assigned to fast, medium and slow machines in schedule I, disregarding machine k. Similarly, W_F', W_M', and W_S' denote the respective values in schedule II, disregarding k'. Notice that the first three values total to $W_{\neq k}$, whereas the second three total to $W_{\neq k}'$.

(2.2.1) In II the smallest job assigned to any fast machine has size at most 1/3.

We claim that on fast machines and on 1-machines $w \geq w'$. Since there is a job $\bar{t} \leq 1/3$ on a fast machine, Proposition 3.26 (ii) implies $f_i' \leq 5/3$ for any living machine i' in II. On the other hand, if $s_i \geq 1$, then $f_i > 2 - t \geq 2 - 1/3 = 5/3$ by (3.d). If $w \geq w'$ on 1/2-machines as well, then we are done

by Corollary 3.20. Therefore, we assume that there is a 1/2-machine i, s.t. $w_i < w'_i$. Now by Proposition 3.24, $d < D$ holds. Furthermore, $t_b \geq t_d > 1 - t \geq 2/3$ (see Fig. 3.4). Finally, note that in schedule I, any 1/2-machine has work $w > 1/2(2 - t/\frac{1}{2}) = 1 - t \geq 2/3$.

In the following we upper bound $W'_M - W_M$. The argument is again analogous to the proof of Theorem 3.3. However, we need to be careful when we combine the results of the next lemmas with the bound on $W'_T - W_T$ from Lemma 3.15.

Lemma 3.32 *Suppose that $w_i < w'_i$ holds for a 1/2-machine i. Then there is a set of $q \geq 0$ jobs $\mathcal{Q} = \{t_d, t_{d+1}, \ldots, t_{d+q-1}\}$ so that*
(i) in II all the q jobs are assigned to 1/2-machines and in I all the q jobs are assigned to fast machines;
(ii) $\frac{2}{3} < t_{d+q-1} \leq t_d \leq \frac{5}{6}$ and
(iii) $W'_M - W_M \leq q \cdot \frac{1}{6}$.
The jobs in $\mathcal{Q}$ will be called $\mathcal{Q}$-jobs.

Proof. Let $\{t_d, \ldots, t_{d+y-1}\}$ denote the first jobs on 1/2-machines in II, and let $\{t_D, \ldots, t_{D+y}\}$ be the first jobs on 1/2-machines including k, in I (see Fig. 3.4). We want to compare W'_M and W_M, so we have to exclude machine k from the comparison.

If $D < d + y$, then we omit machines having the common jobs $\{t_D, \ldots, t_{d+y-1}\}$, and all the jobs on these machines from both I and II. We claim that thus we omit more work from schedule I than from schedule II: Let $0 \leq \eta \leq y-1$, and $t_{D+\eta}$ be the first job assigned to some machine i' in II, and to another machine $i^* \neq k$ in I. Then in I, $w_{i^*} \geq \max(t_{D+\eta}, 1-t) \geq \max(t_{D+\eta}, 2/3)$, whereas in II, $w'_i \leq \max(t_{D+\eta}, 2/3)$ by Proposition 3.13 (ii). For this omitted pair of machines $w_{i^*} \geq w'_i$. If in I the machine with $t_{D+\eta}$ is $i^* = k$, then instead of k we omit one more machine $\hat{i}$ from I (i.e., the one having job t_{d+y}). Since now for any 1/2-machine $w \geq w_k - t \geq t_{D+\eta}$ holds, we obtain again $w_{\hat{i}} \geq w'_i$.

If for a remaining 1/2-machine $w'_i \leq 2/3$ (e.g., if i' has at least 2 jobs in II), we omit i' and its jobs from II and an arbitrary 1/2-machine i^* from I. Then $w_{i^*} > 1-t$ and $w'_i \leq 2/3$.

In total we omitted not less work from I than from II. Furthermore, whenever there are at least 2 jobs on i', for the omitted pair of machines even $w_{i^*} - w'_i \geq (1-t) - 2/3 = 1/3 - t$ holds. For some $q \geq 0$, $\mathcal{Q} := \{t_d, t_{d+1}, \ldots, t_{d+q-1}\}$ are the remaining jobs in II.

(i) Since $b \leq d$ and $d+q-1 < D$, by Proposition 3.1 (ii) the jobs $\{t_d, \ldots, t_{d+q-1}\}$ are scheduled to fast machines (not to 1-machines) in schedule I.

(ii) $t_d \leq \frac{5}{6}$ because on living machines $f' \leq 5/3$; also, $\frac{2}{3} < t_{d+q-1}$ because we omitted machines with smaller jobs.

(iii) q machines remained in I and other q machines remained in II. For the total remaining work $W'_M - W_M \leq \sum_{\tau=0}^{q-1}(t_{d+\tau} - (1-t)) \leq q \cdot (\frac{5}{6} - \frac{2}{3}) = q \cdot \frac{1}{6}$. □

Lemma 3.33 *Let i be a fast machine receiving $v \geq 0$ of the $\mathcal{Q}$-jobs in I. If $s_i = 2^r$ ($r \geq 1$), then $v \leq 2^r$, and $w_i - w'_i > v\frac{1}{6}$. Furthermore, $w_i > 2^r \frac{11}{6}$.*

Proof. Suppose that $v \geq 2^r + 1$, and let $\hat{t}$ be the last $\mathcal{Q}$-job on i. Then $C(\hat{t}) > f_i^b + (2^r+1)\hat{t}/2^r \geq 1 - 1/2^r + (2^r+1)\hat{t}/2^r = 2\hat{t} + (1-\hat{t})(1-1/2^r) \geq 2\hat{t} + \frac{1}{6}(1-1/2^r) > 2\hat{t}$. But if $\hat{t}$ were assigned to an empty 1/2-machine in I, then $C(\hat{t}) = 2\hat{t}$ would hold, a contradiction.

If i is a dead machine, then $w_i - w'_i > v\frac{2}{3}$ trivially holds.

We claim that $w_i > 2^r \frac{11}{6}$, and on living machines $w'_i \leq 2^r \frac{5}{3}$, so $w_i - w'_i > 2^r(\frac{11}{6} - \frac{5}{3}) \geq 2^r \frac{1}{6}$. The upper bound on w'_i follows from $f'_i \leq 5/3$. The lower bound on w_i holds, since (3.d) implies $f_i > 2 - \frac{1}{3}/2 = \frac{11}{6}$ on fast machines. □

The following corollary completes the proof in case (2.2.1).

Corollary 3.34 *In case* (2.2.1)*,* $W_{\neq k} \geq W'_{\neq k}$.

Proof. We assumed that there are 1/2-machines s.t. $w < w'$. Besides, there might be tardy machines with $w < w'$ and for all other machines $w \geq w'$ holds (see Proposition 3.14).

Suppose first that $w \geq w'$ also holds on tardy machines, so $W_S \geq W'_S$. By Lemmas 3.32 and 3.33, there are q $\mathcal{Q}$-jobs on fast machines in I, and

$$W'_M - W_M \leq q \cdot \frac{1}{6} \leq W_F - W'_F. \tag{3.e}$$

So we obtain $W_{\neq k} = W_S + W_M + W_F \geq W'_S + W'_M + W'_F = W'_{\neq k}$.

Second, suppose that there are also tardy machines with $w < w'$. In Corollary 3.20 we assumed that there are tardy machines so that $w < w'$, whereas $w' \leq w$ holds on all other machines; in the previous paragraph we assumed that $w < w'$ on some 1/2-machines, and $w' \leq w$ on all other machines. Fortunately, it is now possible to combine the arguments that we used in these two cases. We proved Corollary 3.20 using Lemmas 3.15 and 3.19. These two lemmas have no preconditions on schedules I and II. In particular, they state that there are p $\mathcal{P}$-jobs, s.t.

$$W'_T - W_T \leq p \cdot (\frac{4}{3} \cdot 1/2^d - t) \leq \sum_{\tau=1}^{\xi} (w_{i_\tau} - w'_{i_\tau}),$$

where $i'_1, i'_2, \ldots, i'_\xi$ are the machines receiving $\mathcal{P}$-jobs in II, and $1/2^d$ is the speed of tardy machines. When 'merging' this inequality with (3.e), special care must be taken when a machine i_τ $(1 \leq \tau \leq \xi)$ is a 1/2-machine or it is a fast machine receiving $\mathcal{Q}$-jobs in I. We elaborate on these two cases:

If machine i_τ is a 1/2-machine, then i'_τ receives ≤ 1 $\mathcal{P}$-job by Lemma 3.19. Moreover, if it does receive a $\mathcal{P}$-job, then ≥ 2 jobs are assigned to i'_τ. So in Lemma 3.32 we matched this machine with some i^* in I, and showed $w_{i^*} - w'_{i_\tau} \geq (1/3 - t) \geq \frac{4}{3} \cdot 1/2^d - t$. We did not exploit this difference in Lemma 3.32. Thus, after matching the 1/2-machines of I and II as described in Lemma 3.32 (instead of the natural $i -\!\!- i'$ matching), this accounts for the difference on a tardy machine associated with the $\mathcal{P}$-job, moreover there is no $\mathcal{Q}$-job on i'_τ.

Now let some i_τ be a fast machine of speed 2^r, s.t. i_τ receives $\leq 2^r$ $\mathcal{Q}$-jobs in I and i'_τ receives $\leq 2^r$ $\mathcal{P}$-jobs in II. By Lemma 3.33, $w_{i_\tau} > 2^r \frac{11}{6}$; whereas by

Lemma 3.19, $w'_{i_\tau} \leq 2^r\frac{4}{3}$. So, $w_{i_\tau} - w'_{i_\tau} \geq 2^r(\frac{11}{6} - \frac{4}{3}) = 2^r\frac{1}{6} + 2^r\frac{1}{3}$. So the total difference on this machine $w_{i_\tau} - w'_{i_\tau}$ covers the necessary amount to balance out the difference associated with these $\mathcal{Q}$-jobs *and* $\mathcal{P}$-jobs. □

(2.2.2) In II the smallest job assigned to any fast machine has size in the interval $(1/3, 2/3]$.

We show that on fast machines $w \geq w'$. Proposition 3.26 (ii) implies $f'_i \leq 3/2 + 2/3 \cdot 1/2 = 11/6$ for any living machine i' in II; whereas according to (3.d), $f_i > 2 - t/2 \geq 2 - 1/3 \cdot 1/2 = 11/6$. By Corollary 3.20 we may assume that there is a medium machine i, s.t. $w_i < w'_i$. Therefore, like in the previous case, $d < D$ and $t_b \geq t_d > 1 - t \geq 2/3$.

Recall that $T_b + 1$ is the time step, right after t_b was scheduled. Note that since $d < D$, t_b is not assigned to a medium machine in I. For the total work received by medium machines (disregarding k and k') before $T_b + 1$, trivially ${W_M^b}' \leq W_M^b$ holds (see Fig. 3.4). For sake of simplicity, we will provide a unified discussion for 1/2-machines and 1-machines; to this end, from now on we consider jobs and work on medium machines received *after* $T_b + 1$:

On 1-machines we start the numeration of jobs after T_b+1; *also, we only consider work received after* $T_b + 1$.

In schedule I, any 1/2-machine has work $w > 1 - t \geq 2/3$; similarly, each living 1-machine receives work $w > 1 - t$ (after $T_b + 1$). Both bounds follow from (3.d). In schedule II, every 1-machine receives at most one job (after $T_b + 1$), since $t_b > 1 - t$, and after the next job the finish time exceeds $1 = \lambda$.

Definition 3.35 *We will call a job* **little** *if it has size* $\leq 1/3$ *and* **normal** *otherwise. Let* $T_{1/3}$ *be the time step, after the last normal job was scheduled.* X, *and* X' *denote the number of living 1-machines (disregarding* k'*), that do not receive any job between* $T_b + 1$ *and* $T_{1/3}$ *in schedule I resp. in schedule II.*

X and X' denote the number of 'empty spaces' on living 1-machines at time $T_{1/3}$. At time $T_{1/3}$, these X resp. X' machines have just one job, and it is of size at most $t_a = 1$. In case $X < X'$ the argument resembles that in (2.2.1). Case $X \geq X'$ will show similarity to (2.2.3).

(2.2.2.1) Assume that $X < X'$.

Lemma 3.36 *If* $X < X'$, *then a set of jobs* $\mathcal{Q} = \{t_{D_1}, \ldots t_{D_\zeta} \ldots, t_{D_q}\}$ *exist, s.t.*
(i) each t_{D_ζ} *is on a medium machine in I, and on a fast machine in II.*
(ii) $W'_M - W_M \leq \sum_{\zeta=1}^{q} \min(1/3, 1 - t_{D_\zeta})$.
We will call the above q jobs $\mathcal{Q}$*-jobs.*

Proof. In schedule I, all first jobs on 1/2-machines are normal jobs. Otherwise (after $T_b + 1$) all first jobs on 1-machines would be little jobs and $X \geq X'$ would hold. Since $d < D$, in II the first jobs on 1/2-machines are also all normal. We omit machine k from the 1/2-machines in I.

Assume that $t_g > 1/3$ is the first job on some medium machine i' in II, and at the same time, the first job on another medium machine $i^* \neq k$ in I. The work

assigned to i' and i^* is $w'_i \leq \max(\frac{2}{3}, t_g)$ by Proposition 3.13 (ii), respectively $w_{i^*} \geq \max(1-t, t_g)$. If i' or i^* is a 1-machine the same bounds trivially hold. We omit all such $i^* - -i'$ machine pairs with all their jobs. We observe that $w_{i^*} \geq w'_i$, since $\max(1-t, t_g) \geq \max(\frac{2}{3}, t_g)$. Moreover, if there are at least 2 jobs on i', then $w_{i^*} - w'_i \geq (1-t) - 2/3 = 1/3 - t$ holds.

Now consider the X' 1-machines in II that are 'empty' at time $T_{1/3}$. These machines all receive at most one little job and work $w' \leq 1/3$ after $T_b + 1$. We pick these machines and all other machines of work at most $2/3$ from II, and match them with the X machines having little first jobs, and with some arbitrary other machines from I. We omit all these machines and jobs as well. Since we had at least $X' > X$ machines to match, we could omit all the X machines from I. For these machine pairs $w - w' \geq (1-t) - \frac{2}{3} = \frac{1}{3} - t$ holds.

Finally, we have $q \geq 0$ machines left in II, and q machines left in I. Each of the q machines in I has a first job of size $> 1/3$. Let these q first jobs be $\mathcal{Q} = \{t_{D_1}, \ldots t_{D_\zeta} \ldots, t_{D_q}\}$. These q jobs are not assigned to medium machines in II, since they would be first jobs by Proposition 3.13 (ii) and we would have omitted them. So, they must be assigned to fast machines by Proposition 3.13 (i). If we match the remaining machines in I and II arbitrarily, for the work difference of such a machine pair $w' - w \leq \min(1 - 2/3, 1 - t_{D_\zeta})$ holds with the respective job size t_{D_ζ}, and this proves the lemma. □

Lemma 3.37 *Let $X < X'$ and the $\mathcal{Q}$-jobs be defined as in Lemma 3.36. Let i be a living fast machine, $s_i = 2^r \quad (r \geq 1)$, then in II i' receives $\leq 2^r - 1$ $\mathcal{Q}$-jobs. If $\hat{t}$ is the last $\mathcal{Q}$-job assigned to i', then $w_i - w'_i \geq (2^r - 1)\min(1/3, 1 - \hat{t}\,)$.*

Proof. Recall that $d < D$, and $\mathcal{Q}$-jobs are not scheduled before t_D. Let $\tilde{f}'_i$ be the finish time of i' before t_d is scheduled. Since t_d is not assigned to i', but to a medium machine, $\tilde{f}'_i + t_d/2 \geq 2t_d$, so $\tilde{f}'_i \geq \frac{3}{2} \cdot t_d$. Therefore, $\tilde{f}'_i \geq \frac{3}{2} \cdot t_d > \frac{3}{2} \cdot (1-t) \geq \frac{3}{2} \cdot \frac{2}{3} = 1$, and by Proposition 3.18, i' receives $\leq 2^r - 1$ further jobs.

Let t^* be the very last job – not necessarily $\mathcal{Q}$-job – on i'. Suppose first that $t^* > 2/3$. Since a job $\bar{t} \leq 2/3$ is assigned to a fast machine of speed 2^u , Proposition 3.26 (i) implies $f'_i \leq 1 + \frac{2}{3}(1 - 1/2^u) + t^*/2^u \leq 5/3 + (t^* - 2/3) \cdot 1/2^u \leq 5/3 + (t^* - 2/3) \cdot 1/2 = 4/3 + t^*/2$. Using (3.d) and $1/3 \geq 1 - t^*$ we obtain

$$f_i - f'_i \geq (2 - 1/3 \cdot 1/2^r) - (4/3 + t^*/2) = (1/2 - 1/2^r) \cdot 1/3 + (1 - t^*) \cdot 1/2 \geq$$

$$\geq (1/2 - 1/2^r) \cdot (1 - t^*) + (1 - t^*) \cdot 1/2 = (1 - 1/2^r)(1 - t^*).$$

$$w_i - w'_i \geq (2^r - 1)(1 - t^*) \geq (2^r - 1)(1 - \hat{t}\,).$$

Now suppose that $t^* \leq 2/3$. Then $C'(t^*) \leq 5/3$, otherwise t^* would have been assigned to k'. Consequently, $f_i - f'_i \geq 2 - 1/3 \cdot 1/2^r - 5/3 = (1 - 1/2^r) \cdot 1/3$, and $w_i - w'_i \geq (2^r - 1) \cdot 1/3$. □

Corollary 3.38 *In case* (2.2.2.1), $W_{\neq k} \geq W'_{\neq k}$.

Proof. If $w \geq w'$ on every slow machine, then $W_S \geq W'_S$. According to Lemmas 3.36 and 3.37, q $\mathcal{Q}$-jobs exist, s.t.

$$W'_M - W_M \leq \sum_{\zeta=1}^{q} \min(1/3, 1 - t_{D_\zeta}) \leq W_F - W'_F,$$

and the statement follows.

If there is a tardy machine, for which $w < w'$ holds, then in addition to $\mathcal{Q}$-jobs, there are p $\mathcal{P}$-jobs assigned to medium and slow machines in II s.t.

$$W'_T - W_T \leq p \cdot (\frac{4}{3} \cdot 1/2^d - t) \leq \sum_{\tau=1}^{\xi} (w_{i_\tau} - w'_{i_\tau}),$$

where $i'_1, i'_2, \ldots, i'_\xi$ are the machines receiving $\mathcal{P}$-jobs in II, and $1/2^d$ is the speed of tardy machines. Since $\mathcal{P}$-jobs are little jobs, in case (2.2.2) they are not assigned to fast machines.

After the (re-)matching of medium machines as done in Lemma 3.36, we want to apply the arguments of Corollary 3.20 and Lemmas 3.36 and 3.37. We need to examine the case when a $\mathcal{P}$-job is assigned to a medium machine i'_τ : By Lemma 3.19, i'_τ receives 1 $\mathcal{P}$-job, and if it is a 1/2-machine, then ≥ 2 jobs are assigned to i'_τ, so $w'_{i_\tau} \leq 2/3$, by Proposition 3.13 (ii); if i'_τ is a 1-machine then $w'_{i_\tau} \leq 2/3$ since i'_τ gets one job after $T_b + 1$. In Lemma 3.36 we matched this machine with some i^* in I, and showed $w_{i^*} - w'_{i_\tau} \geq (1/3 - t) \geq \frac{4}{3} \cdot 1/2^d - t$. We did not exploit this difference in Lemma 3.36. Thus, similarly to Corollary 3.34, merging the two above inequalities yields the proof. □

(2.2.2.2) Assume that $X \geq X'$.

Lemma 3.39 *If $X \geq X'$, then $W_{\neq k} \geq W'_{\neq k}$.*

Proof. Suppose that $X > 0$. Then there is a 1-machine h in I, having a little job as first job (after T_b+1), consequently there is no normal job assigned to a slow machine in I: such a normal job would have been assigned to h instead of a slow machine. So, by Proposition 3.13 (i), both in I and in II, all the normal jobs are assigned to medium and fast machines. By the same argument, there is no normal job assigned to a 1/2-machine as second job in I and II. Therefore, *at the intermediate step* $T_{1/3}$, for the total work $\widetilde{W}_F + \widetilde{W}_M + \widetilde{w}_k = \widetilde{W}'_F + \widetilde{W}'_M + \widetilde{w}'_k$ and $\widetilde{w}_k \leq \widetilde{w}'_k$, consequently $\widetilde{W}_F + \widetilde{W}_M \geq \widetilde{W}'_F + \widetilde{W}'_M$ hold. Now we compare the total amount of little jobs – jobs received after $T_{1/3}$ – on medium and fast machines in I and II.

Fast machines don't get little jobs in II in case (2.2.2); 1-machines receive at most one job after $T_b + 1$ in II. On 1-machines, the total work of little jobs is at most $\frac{1}{3}X'$ in schedule II; while it is at least $(1-t)X \geq (1-t)X' = (\frac{1}{3} - t)X' + \frac{2}{3}X'$ in schedule I.

At time $T_{1/3}$, on each 1/2-machine there is at most 1 job in both schedules (see above), and for each of them $\tilde{w}' \geq \tilde{w}$ holds since $d < D$. If $2/3 \leq \tilde{w}'$, then the 1/2-machine gets no more job in II, by Proposition 3.13 (ii). Let i be a 1/2-machine

and $\tilde{w}_i \leq \tilde{w}'_i < 2/3$ at time $T_{1/3}$. In schedule I i receives at least $(1 - t - \tilde{w}_i) \geq (\frac{1}{3} - t) + \frac{2}{3} - \tilde{w}_i$ more work; while in schedule II i' receives at most $\frac{2}{3} - \tilde{w}'_i \leq \frac{2}{3} - \tilde{w}_i$ more work.

Let y be the number of 1/2-machines s.t. $\tilde{w}' < 2/3$ at time $T_{1/3}$. We have shown above that $W_F + W_M \geq W'_F + W'_M + (y + X')(\frac{1}{3} - t)$ holds after scheduling all the jobs. If $w \geq w'$ for every slow machine, then obviously $W_{\neq k} \geq W'_{\neq k}$. Otherwise, on tardy machines $W'_T - W_T \leq p \cdot (\frac{4}{3} \cdot 1/2^d - t) \leq p \cdot (\frac{1}{3} - t)$, where p is the number of $\mathcal{P}$-jobs. Note that fast machines don't receive $\mathcal{P}$-jobs, and medium machines receive altogether at most $y + X'$ $\mathcal{P}$ jobs in II. So the part $(y + X')(\frac{1}{3} - t)$ of the above difference is enough to compensate for $\mathcal{P}$-jobs on medium machines, and Corollary 3.20 applies.

Finally, let $X = X' = 0$. If there is a 1/2-machine in I with a little job as first job, then there is no normal job on slow machines in I (this normal job would rather have been assigned to the 1/2-machine that was still empty), and the proof is the same as above. If on all 1/2-machines the first job is normal in I, then the proof is the same as in case (2.2.2.1). □

(2.2.3) In II all jobs assigned to fast machines are larger than 2/3.

Similarly to case (2.1), the proof of this case is based on comparing the number of jobs assigned to fast and medium machines in I and II. Let $T_{2/3}$ be the time step after the last job of size $> 2/3$ was scheduled. Note that after $T_{2/3}$, no job is assigned to any fast machine in II.

We may assume $T_b < T_{2/3}$. Otherwise $w \geq w'$ on fast machines, and $t_b \leq 2/3 \leq 1 - t$ implies that $w \geq w'$ on medium machines, by Proposition 3.24. So Corollary 3.20 immediately yields $W_{\neq k} \geq W'_{\neq k}$.

We distinguish two subcases:

(2.2.3.1) Assume that between T_b and $T_{2/3}$ at most 2^r jobs are assigned to any fast machine of speed 2^r $(r \geq 1)$ in schedule II.

We merge the last two zones of Definition 3.28:

Definition 3.40 *Let i be a fast machine of speed 2^r $(r \geq 1)$. Let f_i^b and the zones z_i be defined as in Definitions 3.27 and 3.28. The new zones $\bar{z}_i$ are*

$$\bar{z}_i = \begin{cases} z_i & \text{if} \quad z_i \leq 2^r - 1 \\ 2^r & \text{if} \quad 1 - 1/2^r \leq f_i^b \leq 1. \end{cases}$$

A *hole* on a medium or fast machine, roughly means an empty space, that might or will be filled with one job:

Definition 3.41 *Suppose that i is a fast machine of speed 2^r, and f_i^b is in the $\bar{z}_i$th zone. At some time step after T_b, machine i has x_i* **holes** *if it received $\bar{z}_i - x_i < \bar{z}_i$ jobs after T_b, and* 0 **holes** *otherwise.*

A 1/2-machine has 1 **hole** *if it is empty, and* 0 **holes** *otherwise. A 1-machine has* 1 **hole** *if it has* 1 *job of size at most $t_a = 1$, and* 0 **holes** *otherwise; k' has* 0 **holes**.

Note that at time $T_b + 1$, the total number of holes in I and II is the same: machine k' has 0 holes in II and k has 1 hole in I; but the machine receiving job t_b in I, has one hole less in I than in II.

Now $\bar{X}$ and $\bar{X}'$ denote the number of empty spaces (holes) on fast and medium machines at time $T_{2/3}$:

Proposition 3.42 *Let x_i be the number of holes on machine i in schedule I, resp. x'_i be the number of holes on machine i' in schedule II at time $T_{2/3}$. Let $\bar{X} = \sum_i x_i$ and $\bar{X}' = \sum_i x'_i$. Then $\bar{X} - \bar{X}' \geq v$, where v is the number of jobs assigned to slow machines in I before $T_{2/3}$.*

Proof. In schedule II, slow machines do not receive jobs before $T_{2/3}$, by Proposition 3.13 (i). A 1/2-machine receives at most 1 job before $T_{2/3}$, by Proposition 3.13 (ii); a 1-machine receives at most 1 job between $T_b + 1$ and $T_{2/3}$, because after two jobs of size $> 2/3$, the machine has finish time > 1. A fast machine i receives $\leq \bar{z}_i$ jobs by Proposition 3.30 and because we are in case (2.2.3.1). Thus every job between $T_b + 1$ and $T_{2/3}$, fills exactly one hole in II.

In schedule I, slow machines receive v jobs between $T_b + 1$ and $T_{2/3}$, so at least v jobs do not fill any hole on medium or fast machines in I. We obtain $\bar{X} \geq \bar{X}' + v$. □

Proposition 3.43 *After $T_{2/3}$, a medium or fast machine i' receives $\leq \frac{2}{3}x'_i$ further work in schedule II; machine $i \neq k$ receives $\geq (1-t)x_i \geq \frac{2}{3}x_i$ further work in schedule I. Machine k receives $\geq (1-t)x_k + t$ further work in I.*

Proof. First, consider schedule II. Large machines do not get further jobs after $T_{2/3}$. If a 1/2-machine i receives only one more job, this job has size at most 2/3; if it receives ≥ 2 jobs, then by Proposition 3.13 (ii) it receives total work $w'_i \leq 2/3$. On a 1-machine, only the last job may have completion time larger than 1 (because k' has finish time 1). Since $t_b > 2/3$, this implies that even if the 1-machine gets more than 1 further jobs, the total work received after $T_{2/3}$ is at most 2/3.

In schedule I, if there is a hole on a medium machine at $T_{2/3}$, then it receives further work $\geq 1 - t$, by (3.d). If there are $x_i > 0$ holes on a fast machine i at $T_{2/3}$, then the finish time $\tilde{f}_i$ of i at $T_{2/3}$ is not higher than the x_ith zone (see Figure 3.5). Let $s_i = 2^r$, then $f_i - \tilde{f}_i > 2 - t/2^r - (2 - x_i \cdot 1/2^r) \geq (1-t)x_i \cdot 1/2^r$. So i receives more than $(1-t)x_i$ total further work in I.

If $x_k = 0$, then k is assigned job t after $T_{2/3}$; if $x_k = 1$, then k is empty at $T_{2/3}$, and later it receives more than $w'_k = 1$ further work. □

Lemma 3.44 *In case* (2.2.3.1)*, for the final total work $W_{\neq k} + w_k > W'_{\neq k} + w'_k$ holds.*

Proof. Suppose first, that there were no $\mathcal{P}$-jobs, i.e., $w \geq w'$ for every slow machine. Let $\{t_{j_1}, \ldots, t_{j_\tau}, \ldots, t_{j_v}\}$ be the jobs assigned to slow machines in I before $T_{2/3}$. Note that if job $t_{j_\tau} > 2/3$ is assigned to a slow machine in I, then on this slow machine $w - w' \geq t_{j_\tau} - 1/3$, by Proposition 3.13 (i). In total, $W_S - W'_S \geq \sum_{\tau=1}^{v} t_{j_\tau} - v \cdot 1/3$.

Now we provide a lower bound on $\Delta := (W_F + W_M + w_k) - (W'_F + W'_M + w'_k)$. At time $T_{2/3}$, the difference $(\widetilde{W}_F + \widetilde{W}_M + \widetilde{w}_k) - (\widetilde{W}'_F + \widetilde{W}'_M + \widetilde{w}'_k) = -(t_{j_1} + \ldots + t_{j_v})$.

Due to Proposition 3.43, after $T_{2/3}$ these machines receive at most $\frac{2}{3} \cdot \bar{X}'$ more work in II, and at least $(1-t) \cdot \bar{X} + t$ more work in I. Consequently, using Proposition 3.42

$$\Delta \geq (1-t) \cdot \bar{X} + t - \frac{2}{3} \cdot \bar{X}' - \sum_{\tau=1}^{v} t_{j_\tau} \geq$$

$$(1-t) \cdot (\bar{X}' + v) + t - \frac{2}{3} \cdot \bar{X}' - \sum_{\tau=1}^{v} t_{j_\tau} \geq$$

$$\bar{X}'(1/3 - t) + v \cdot 2/3 + t - \sum_{\tau=1}^{v} t_{j_\tau}.$$

Summing up, $(W_{\neq k} + w_k) - (W'_{\neq k} + w'_k) = (W_S + W_F + W_M + w_k) - (W'_S + W'_F + W'_M + w'_k) \geq \bar{X}'(1/3 - t) + v \cdot 1/3 + t > 0$.

Now we consider the case when there are $p > 0$ $\mathcal{P}$-jobs, i.e., a tardy machine i exists s.t. $w_i < w'_i$. Note that since fast machines do not receive jobs of size $\leq 2/3$, $\mathcal{P}$-jobs are assigned to slow and medium machines in II. By Lemma 3.15, $W'_T - W_T \leq p \cdot (\frac{4}{3} \cdot 1/2^d - t) \leq p \cdot (1/3 - t)$, where $1/2^d$ is the speed of tardy machines. Medium machines with 1 hole, resp. the $\leq v$ slow machines involved in the above calculation receive altogether at most $v + \bar{X}'$ $\mathcal{P}$-jobs by Lemma 3.19. The part $v \cdot 1/3 + \bar{X}'(1/3 - t)$ in the above difference accounts for these $\leq v + \bar{X}'$ $\mathcal{P}$-jobs; for $\mathcal{P}$-jobs on other slow machines Lemma 3.19 and applies directly. □

(2.2.3.2) There is a fast machine of speed 2^u receiving at least $2^u + 1$ jobs between T_b and $T_{2/3}$ in schedule II.

After a technical proposition, the proof follows the same lines as in case (2.2.3.1), with only slight modifications.

Proposition 3.45 *In schedule II, after $T_{2/3}$ there are no holes on 1/2-machines, and for the finish time of any 1-machine $\tilde{f}' > 5/6$ holds.*

Proof. Consider schedule II at the time step $T_{2/3}$. Let $\bar{i}$ be a fast machine of speed 2^u, receiving $2^u + 1$ jobs after T_b. Let $\bar{t}$ be the last of these jobs. Then $\bar{i}$ has finish time $f'_{\bar{i}} \geq 1 - 1/2^u + (2^u + 1)\bar{t}/2^u \geq 2\bar{t}$. Thus, if there were an empty 1/2-machine, then $\bar{t}$ would have been assigned to this machine, and $C'(\bar{t}) = 2\bar{t}$ would hold. So, there is no empty 1/2-machine at time $T_{2/3}$.

If there were a 1-machine with finish time at most 5/6, then by the same argument, this would imply $1 - 1/2^u + (2^u + 1)\bar{t}/2^u < 5/6 + \bar{t}$, that boils down to $(1 - \bar{t})/2^u > 1/6$. The latter contradicts to $\bar{t} > 2/3$ and $u \geq 1$. □

Now we modify again the last two zones of Definition 3.28:

Definition 3.46 *Let i be a fast machine of speed 2^r, and z_i be defined as in Definition 3.28. The new zones $\hat{z}_i$ are*

$$\hat{z}_i = \begin{cases} z_i & \text{if} \quad z_i \leq 2^r - 1 \\ 2^r & \text{if} \quad 1 - \frac{2}{3} \cdot 1/2^r < f_i^b \leq 1 \\ 2^r + 1 & \text{if} \quad 1 - 1/2^r \leq f_i^b \leq 1 - \frac{2}{3} \cdot 1/2^r. \end{cases}$$

We define the number of **holes** at some time step after T_b on fast and medium machines analogously to Definition 3.41, with the zones $\bar{z}_i$ replaced by $\hat{z}_i$ in the definition. Like before, at time T_b+1 the total number of holes is the same in I and in II. Let $T_{1/3}$ and $T_{1/2}$ be the time steps after the last job of size $> 1/3$ resp. $> 1/2$ was scheduled.

Proposition 3.47 *Let $\hat{x}_i$ be the number of holes on i in I, and $\hat{x}'_i$ be the number of holes on i' in II at time $T_{1/3}$. Let $\hat{X} := \sum_i \hat{x}_i$ and $\hat{X}' := \sum_i \hat{x}'_i$. Then $\hat{X} - \hat{X}' \geq v$, where v is the number of jobs assigned to slow machines in I before $T_{1/3}$. The same statement holds, having $T_{1/3}$ replaced everywhere by $T_{1/2}$.*

Proof. Like in Proposition 3.42, it is enough to show that every job received between T_b+1 and $T_{1/3}$ fills a hole in schedule II.

By Proposition 3.13 (i), slow machines do not receive jobs before $T_{1/3}$. We claim that a medium machine receives at most 1 job between T_b+1 and $T_{1/3}$. For 1/2-machines this follows from Proposition 3.13 (ii). Since we assumed that $t_b > 2/3$, such a job on a 1-machine has completion time more than 1, and any further job would prefer machine k'.

It remains to show that a fast machine i' receives at most $\hat{z}_i$ jobs. For $\hat{z}_i \leq 2^r - 1$ the proof is the same as in Proposition 3.30. The same way as in Proposition 3.45, it follows that in II, a fast machine i' has finish time > 1 at time $T_{2/3}$. So, if $\hat{z}_i = 2^r$, then after one job, respectively if $\hat{z}_i = 2^r + 1$, then after at most two jobs of size $> 2/3$, i' has finish time > 1. By Proposition 3.18, at most $2^r - 1$ further jobs are put here. □

Proposition 3.48 *Let $\hat{x}_i$ denote the number of holes on a machine i at time $T_{1/3}$ (resp. $T_{1/2}$). After $T_{1/3}$ (resp. $T_{1/2}$), a 1-machine with 1 hole receives at most $1/3$ (resp. at most $1/2$) further work in schedule II.*

A medium or fast machine $i \neq k$ receives at least $(2/3 - t)\hat{x}_i \geq \frac{1}{3}\hat{x}_i$ further work in schedule I. Machine k receives at least $(1-t)\hat{x}_k + t$ further work.

Proof. In schedule II, disregarding slow machines, only 1-machines with 1 hole might receive jobs after $T_{2/3}$, since the finish time of every other medium machine exceeds 1. Moreover, since at $T_{2/3}$ the finish time of a 1-machine is $> 5/6$, it is easy to show that the total work received after $T_{1/3}$ is $\leq 1/3$; whereas the total work received after $T_{1/2}$ is $\leq 1/2$.

For schedule I the proof is the same as in Proposition 3.43, except when for a fast machine i of speed $s_i = 2^r$, f_i^b is in zone $\hat{z}_i = 2^r + 1$ at time T_b. In this case, after $\hat{z}_i - \hat{x}_i$ more jobs, the finish time of i is $\tilde{f}_i \leq 1 - 2/3 \cdot 1/2^r + (2^r + 1 - \hat{x}_i) \cdot 1/2^r = 2 - 2/3 \cdot 1/2^r - (\hat{x}_i - 1) \cdot 1/2^r$. Therefore, due to (3.d), at least $(\hat{x}_i - 1) + 2/3 - t \geq \hat{x}_i \cdot (2/3 - t)$ more work is assigned to i in I. □

Lemma 3.49 *In case* (2.2.3.2), $W_{\neq k} + w_k > W'_{\neq k} + w'_k$.

Proof. Suppose first, that no $\mathcal{P}$-jobs are assigned *to slow machines* in II. In this case the proof follows the same lines as in Lemma 3.44:

Let $\{t_{j_\tau}\}_{\tau=1}^{v}$ be the jobs assigned to slow machines in I before $T_{1/3}$. If job $t_{j_\tau} > 1/3$ is assigned to a slow machine in I, then on this slow machine $w - w' \geq t_{j_\tau} - 1/3$. (If at least 2 jobs are on one slow machine in I, then even stronger inequalities hold.) In total, $W_S - W'_S \geq \sum_{\tau=1}^{v} t_{j_\tau} - v \cdot 1/3$.

After $T_{1/3}$, medium and fast machines and k receive at most $\frac{1}{3} \cdot \hat{X}'$ more work in II and at least $(2/3 - t) \cdot \hat{X} + t$ more work in I by Proposition 3.48. Consequently,

$$(W_F + W_M + w_k) - (W'_F + W'_M + w'_k) \geq$$

$$(2/3 - t) \cdot \hat{X} + t - 1/3 \cdot \hat{X}' - \sum_{\tau=1}^{v} t_{j_\tau} \geq$$

$$(2/3 - t) \cdot (\hat{X}' + v) + t - 1/3 \cdot \hat{X}' - \sum_{\tau=1}^{v} t_{j_\tau} \geq$$

$$\hat{X}'(1/3 - t) + v \cdot 1/3 + t - \sum_{\tau=1}^{v} t_{j_\tau}.$$

To sum up, $(W_S + W_M + W_F + w_k) - (W'_S + W'_M + W'_F + w'_k) \geq \hat{X}'(1/3 - t) + t > 0$. Like in the proof of Lemma 3.44, the part $\hat{X}'(1/3 - t)$ of the difference compensates for the $\leq \hat{X}'$ potential $\mathcal{P}$-jobs assigned to 1-machines in II.

Second, let there be $\mathcal{P}$-jobs assigned to slow machines in II. Recall that $\mathcal{P}$-jobs are assigned to faster than tardy machines in II, which implies $1/2^d \leq 1/2^3$, for the speed of tardy machines, and by Proposition 3.13 (i), $t \leq 1/6$. We consider $\hat{X}'$, $\hat{X}$ and $\{t_{j_\tau}\}_{\tau=1}^{v}$ at time $T_{1/2}$. We obtain $W_S - W'_S \geq \sum_{\tau=1}^{v} t_{j_\tau} - v \cdot 1/3$. After $T_{1/2}$, medium and fast machines receive at most $1/2 \cdot \hat{X}'$ more work in II and at least $(2/3 - t) \cdot \hat{X} + t \geq 1/2 \cdot \hat{X} + t$ more work in I. So,

$$(W_F + W_M + w_k) - (W'_F + W'_M + w'_k) \geq$$

$$(2/3 - t) \cdot \hat{X} + t - 1/2 \cdot \hat{X}' - \sum_{\tau=1}^{v} t_{j_\tau} \geq$$

$$\hat{X}'(1/6 - t) + v \cdot 1/2 + t - \sum_{\tau=1}^{v} t_{j_\tau}.$$

To sum up, $(W_S + W_M + W_F + w_k) - (W'_S + W'_M + W'_F + w'_k) \geq v \cdot 1/6 + \hat{X}'(1/6 - t) + t > 0$. Now we have $d \geq 3$, so $W'_T - W_T \leq p \cdot (\frac{4}{3} \cdot 1/2^d - t) \leq p \cdot (1/6 - t)$. Therefore, the part $v \cdot 1/6 + \hat{X}'(1/6 - t)$ of the difference compensates for the $\leq v + \hat{X}'$ $\mathcal{P}$-jobs by the same argument as in Lemma 3.44. □

□ *Theorem 3.4*

3.5 The 'one fast machine' case

In this section we take a detour to the 'one fast machine' case. While trying to prove a better than 1.5 approximation bound for LPT on 2-divisible machines, we found a tight asymptotic worst case bound of $\frac{\sqrt{3}+1}{2} \approx 1.3660$ for speed vectors of the form

$\langle 1, 1, \ldots, 1, s\rangle$. As noted earlier, the previous best lower and upper bounds in this case were 4/3 and 3/2, respectively [42]. Moreover, the bound 4/3 was conjectured to be tight.

The worst case ratio of LPT on 2-divisible machines will be (re)discussed in Sections 3.6 and 3.7. The lower bound instances and upper bound proofs given there, are refined versions of those in this chapter.

3.5.1 A lower bound: $\frac{\sqrt{3}+1}{2} - \epsilon$

In this section we present an instance of the $Q||C_{\max}$ problem with $m-1$ machines of speed 1 and one machine of speed 2^r ($r \in \mathbb{N}$). Since our primary concern are 2-divisible machines, our example speed vector is 2-divisible. We remark however, that an arbitrary (sufficiently large) s would do in place of the fast speed 2^r. Note also, that LPT with arbitrary tie-breaking produces the same schedule of the instance. We will call the machine of speed 2^r the **fast machine**.

Theorem 3.6 *For any $\epsilon > 0$ there is a speed vector $\langle s_1 = 1, \ldots, s_{m-1} = 1, s_m = 2^r\rangle$ and a job vector $\langle t_1, \ldots, t_n\rangle$, such that for this instance $Lpt/Opt > (\sqrt{3}+1)/2 - \epsilon$.*

Proof. The proof is given by Instance A below. The approximation ratio of LPT on this instance can be arbitrarily close to $(\sqrt{3}+1)/2 \approx 1.366$. In particular, $Lpt > \sqrt{3}+1-\epsilon'$ and $Opt < 2+\epsilon'$, where $\epsilon' > 0$ is arbitrarily small if m and r are large enough.

Instance A. Let $x = 3-\sqrt{3} \approx 1.268$ and $y = \sqrt{3}-1 \approx 0.732$. We start by describing the assignment of jobs to machines in LPT (see Figure 3.6): The fast machine first receives 2^r-1 jobs of size x; then it is filled with as many jobs of size 1 as fit below time 2; finally it gets 2^r-1 jobs of size y. At this point the number of jobs on the fast machine is $2\cdot(2^r-1) + \lfloor 2\cdot 2^r - (2^r-1)\cdot x\rfloor$, and the total work on the fast machine amounts to at least $(2^r-1)\cdot x + 2\cdot 2^r - (2^r-1)\cdot x - 1 + (2^r-1)\cdot y = 2^r(2+y) - 1 - y$.

The set of 1-machines is divided into blocks. The number of 1-machines in one block is $(x-1)/\delta$, where $\delta > 0$ is arbitrarily small and it divides $x-1$ evenly. The LPT schedule on a block is as follows: Each 1-machine has a large and a small job. The large jobs range from $x-\delta$ down to 1 by steps of δ and the small jobs range from y up to $1-\delta$ by steps of δ. Every 1-machine has total work $y+x-\delta = 2-\delta$.

We claim that if $1/2^r < \delta$, then the above assignment is an LPT schedule: all x-jobs on the fast machine are completed by time $x - x/2^r$; after that, 1-machines receive their first jobs, all of size less than x. These jobs would have higher completion time on the fast machine m. Since an additional 1-job on a 1-machine would not be completed before time 2, the 1-jobs are all assigned to m. Now the 1-machines receive their second jobs with completion time $2-\delta < 2-1/2^r$ where $2-1/2^r$ is a lower bound on the current completion time of m. Finally, after (at most) 2^r-1 jobs of size y, a last y-job is assigned to one of the 1-machines, yielding makespan $(y+2-\delta) = \sqrt{3}+1-\delta$. On the fast machine this last job would have been completed after $(2^r(2+y)-1)/2^r = \sqrt{3}+1-1/2^r > \sqrt{3}+1-\delta$.

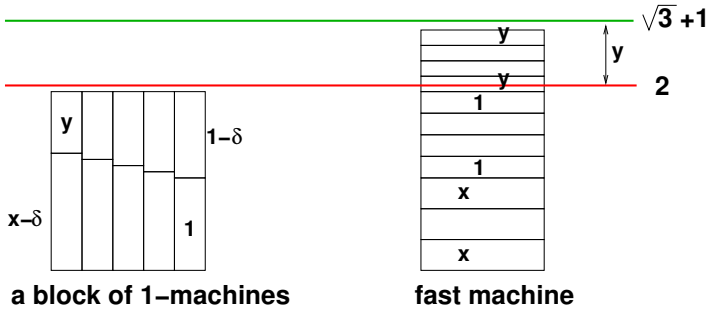

Figure 3.6: Instance A: the assignment of jobs before the last job in LPT.

Now we rearrange the jobs on the machines in order to get the optimum schedule. We claim that a block of 1-machines can be used to exchange an x-job for a 1-job *or* to exchange a 1-job for a y-job. The first happens if we shift the large jobs within a block, insert a job of size x instead of $x-\delta$, and take out a job of size 1. The second happens, if we shift the small jobs within a block, insert a 1-job and take out a y-job. In either case the new finish time on 1-machines will be 2.

Let the number of blocks be $2\cdot(2^r-1)+\lfloor 2\cdot 2^r-(2^r-1)\cdot x\rfloor$, so that every job of size x or size 1 on the fast machine can eventually be exchanged for a y-job. Moreover, we put the very last job of size y on the fast machine. Now the total work on the fast machine is at most $y\cdot(2(2^r-1)+2\cdot 2^r-(2^r-1)\cdot x)+y = y\cdot 4\cdot 2^r - y\cdot 2^r\cdot x + y(x-1) = 2^r\cdot y\cdot(4-x)+y\cdot(x-1) = 2^r\cdot 2+y\cdot(x-1)$. Thus, the optimum makespan is at most $2+y(x-1)/2^r$. Clearly, the desired bound is obtained if $\epsilon' > \delta > 1/2^r > y(x-1)/2^r$ for some appropriate ϵ' (if for the ϵ given in the theorem $1-\epsilon > \frac{4}{\sqrt{3}+3}$ holds, then we can take $\epsilon'=\epsilon$).

We also note concerning the second schedule, that since all jobs on the fast machine have the smallest job size, we could only get a better schedule, if the fast machine received less jobs; but then there would be at least 3 jobs on one of the 1-machines, resulting in a larger makespan. Thus, this schedule is really optimal. □

3.5.2 Tight upper bound

We consider the special case of $Q||C_{\max}$ when $s_1 = s_2 = \ldots = s_{m-1} = 1$ and $s_m = s > 1$. We show that in this case the bound given in Section 3.5.1 is tight:

Theorem 3.7 *For any instance of the $Q||C_{\max}$ problem for which $s_1 = s_2 = \ldots = s_{m-1} = 1$ and $s_m = s > 1$ holds, $Lpt/Opt < (\sqrt{3}+1)/2$.*

In the rest of the section we prove Theorem 3.7. The proof is by contradiction: we consider a minimal counterexample, i.e., an instance with minimum number of machines, for which $Lpt/Opt \geq (\sqrt{3}+1)/2$. We fix any optimal schedule of this instance and denote it by OPT.

This proof – and also the proof in Section 3.6 – is based on the following elementary technique: Our starting point is the LPT schedule. First we rearrange the jobs of LPT within 1-machines. Then we pick jobs $\{t_j^*\}$ of machine m and put them to 1-machines according to how they are scheduled in OPT. We will have to

put other jobs from 1-machines back to machine m. This exchanging process will be carried out sometimes one by one, other times by moving sets of jobs. We will calculate the minimum possible ratio: (work moved to m)/(work moved from m). This ratio depends on which time period of machine m the jobs $\{t_j^*\}$ are taken from. Proposition 3.52 is a basic technical tool for distinguishing these time periods.

For sake of convenience, we assume w.l.o.g. that $Opt = 2$, and so $Lpt \geq \sqrt{3}+1$. Let $t = t_n$ be the size of the last job, and f_i denote the finish time of machine i *before the last job is scheduled.* $Lpt \geq \sqrt{3}+1$ implies $f_i \geq \sqrt{3}+1-t$ for $1 \leq i \leq m-1$ and $f_m \geq \sqrt{3}+1-t/s$. We will carry out a case analysis, and obtain that $Lpt/Opt \geq (\sqrt{3}+1)/2$ is impossible in all of the cases. Lemmas 3.50, 3.54 and 3.55 yield the proof of Theorem 3.7.

Analogues to the following lemma can already be found in [42].

Lemma 3.50 *If $t \leq \sqrt{3}-1$, or $t > 1$, then $\frac{Lpt}{Opt} \geq \frac{\sqrt{3}+1}{2}$ is impossible.*

Proof. If $t \leq \sqrt{3}-1$, then for each machine $f_i \geq \sqrt{3}+1-t \geq 2$. So for the total amount of work $\sum_{j=1}^{n} t_j > \sum_{j=1}^{n-1} t_j = \sum_{i=1}^{m} s_i \cdot f_i \geq (m-1+s)\cdot 2$ holds, contradicting $Opt = 2$.

Now let $t > 1$. Since t is the smallest job, now in OPT there is one job of size at most 2 on every 1-machine. Let $\{t_1^*, \ldots, t_{m-1}^*\}$ be the set of these jobs. It follows from the principle of domination that for all $1 \leq i \leq m-1$, the job t_i^* is strictly larger than any job assigned to a 1-machine in LPT. Therefore, in LPT t_i^* is on machine m, and has completion time at most 2. Otherwise it would have been assigned to a 1-machine. For every 1-machine in LPT $f_i \geq \max(t, \sqrt{3}+1-t) \geq (\sqrt{3}+1)/2 > 4/3$ holds. Let $W^* = \sum_{i=1}^{m-1} t_i^*$, be the total work on 1-machines in OPT. Furthermore, let W denote the total work on 1-machines in LPT, disregarding t_n. Then $W/W^* \geq (4/3)/2 = 2/3$, since this ratio holds on every 1-machine.

Now we exchange the jobs of the LPT schedule in order to get OPT: First we put t_n on machine m, so that m has total work at least $(\sqrt{3}+1)s$. Second, we put the jobs t_i^* from machine m – from below time 2 – to 1-machines, and all the jobs from 1-machines to m. Now the reduced amount of work on machine m is at least $(W/W^*)\cdot 2s + (\sqrt{3}-1)s$, and this work can be done in time 2, so

$$\frac{W}{W^*} \cdot 2s + (\sqrt{3}-1)s \leq 2s$$

$$\frac{2}{3} \cdot 2 + (\sqrt{3}-1) \leq 2$$

a contradiction. □

In the rest of the proof of Theorem 3.7, we assume $\sqrt{3}-1 < t \leq 1$. Now in OPT there are at most 2 jobs on every 1-machine, since $3(\sqrt{3}-1) > 2$. Furthermore, in LPT every 1-machine has finish time $f_i \geq \sqrt{3}+1-1 = \sqrt{3}$. Thus, on a 1-machine in LPT there is either a job of size $\geq \sqrt{3}$, or at least two jobs.

Let $t_a \geq t_{a+1} \geq \ldots \geq t_{a+m-2} = t_b$ be the first jobs assigned to the 1-machines in LPT as described by Proposition 3.1 (i) (see Fig. 3.7). Let $t'_a \leq t'_{a+1} \leq \ldots \leq t'_b$ denote

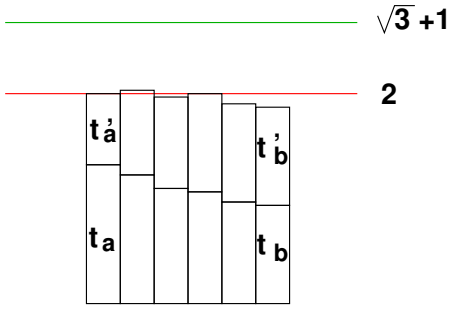

Figure 3.7: The first two jobs on 1-machines in LPT.

the second jobs on the respective machines[1] if they exist (these are not consecutive jobs). If $t'_a, t'_{a+1}, \ldots, t'_{a+v}$ do not exist, then let $t'_a = t'_{a+1} = \ldots = t'_{a+v} = 0$.

Proposition 3.51 *Let $t_a > 2 - t$. In* OPT *let t' be a job on a 1-machine and t'' be another job on the same machine if such a t'' exists. Now $t' > t'_a$ holds. Furthermore, if $t' \in \{t'_a, t'_{a+1}, \ldots, t'_b\}$, then $t'' \in \{t_a, t_{a+1}, \ldots, t_b\}$.*

Proof. If t'' does not exist, then let $t'' = 0$. Observe, that $t'' \leq 2 - t < t_a$, otherwise t'' and t' would not fit onto one machine in OPT. Therefore, by the principle of domination $t' > t'_a$. Moreover, if t'' is not one of $t_a, t_{a+1}, \ldots, t_b$, then $t'' \leq t_b$. Consequently, $t' > t'_b = \max(t'_a, t'_{a+1}, \ldots, t'_b)$, so $t' \notin \{t'_a, t'_{a+1}, \ldots, t'_b\}$. □

Proposition 3.52 *Let t^* be a job assigned to machine m in* LPT *and to a 1-machine in* OPT*. Let T^* denote the completion time of t^* in* LPT*. If $t^* > t_a$, then $T^* \leq t^* \leq 2$. If $t^* > t'_a$, then $T^* \leq \max(2, t_b + t^*)$.*

Proof. If $t^* > t_a$, then $T^* \leq t^*$, otherwise t^* would have been assigned to a 1-machine before t_a.

If $t^* \leq t_a$ then, by the principle of domination, on the 1-machine of t^* in OPT there is another job. Let t^{**} denote this job.

Suppose first, that $t^{**} \leq t_b$, then $t^* > t'_b$, so $T^* \leq t^* + t_b$, otherwise t^* would have been scheduled on top of t_b.

Second, let $t^{**} = t_{a+v}$ be one of the first jobs on 1-machines. Then $t^* > t'_{a+v}$, and $T^* \leq t_{a+v} + t^* = t^{**} + t^* \leq 2$, otherwise t^* would have been assigned as a second job after t_{a+v}. Finally, if $t^{**} > t_a$, then $t^* + t_a < t^* + t^{**} \leq 2$, so either $T^* \leq 2$, or $t^* \leq t'_a$, otherwise t^* would have been assigned as a second job after t_a. □

Corollary 3.53 *Let t^* and T^* be defined as in Proposition 3.52. If $t_a > 2 - t$, then $T^* \leq \max(2, t_b + t^*)$.*

Proof. Since $t_a > 2 - t$, by Proposition 3.51, $t^* > t'_a$. Now Proposition 3.52 implies $T^* \leq \max(2, t_b + t^*)$. □

Lemma 3.54 *If $\sqrt{3} - 1 < t \leq 1$ and $t_b \leq 1$, then $\frac{Lpt}{Opt} \geq \frac{\sqrt{3}+1}{2}$ is impossible.*

[1] A different order, due to jobs of equal size would be easy to handle by reordering the 1-machines.

Proof. As before, we put t_n on machine m, so that it has total work at least $(\sqrt{3}+1)s$. Let $t^* \leq 2$ be a job that is on a 1-machine in OPT, but on machine m in LPT. Then either $t^* \geq t_a$ or $t^* \leq t_b \leq 1$.

We consider two cases. Suppose first, that $t_a > 2 - t$. By Corollary 3.53, the completion time of any t^* in LPT is at most $\max(2, t_b + t^*) = 2$. We start by rearranging the jobs *within* 1-machines in LPT: From 1-machines with at least two jobs, we match jobs that belong to the same 1-machine in OPT, and delete the matched pairs of jobs together with a 1-machine for each pair. As a consequence of Proposition 3.51, any job *not* in $\{t_a, t_{a+1}, \ldots, t_b\}$ that is on a 1-machine in OPT, is by now deleted. We can rearrange the remaining jobs so, that on every remaining 1-machine there is either one job of size at least $\sqrt{3}$, or (at least) two jobs, so that at most one of these jobs remains on the 1-machine in OPT.

Now we put jobs from m to 1-machines. If there is no remaining job on the 1-machine, then we exchange total work of at most 2, for one job of size at least $\sqrt{3}$, or for two jobs of total size at least $2t$. Otherwise we exchange $t^* \leq t_b \leq 1$, for one job of size at least t. The size reduction cannot be smaller than $\min(\sqrt{3}/2,\ 2t/2,\ t/1) = \min(\sqrt{3}/2,\ t)$.

The reduced work on the fast machine is at most $2s$, so

$$\min(\frac{\sqrt{3}}{2}, t) \cdot 2s + (\sqrt{3}-1)s \leq 2s$$

therefore either $\sqrt{3}/2 \cdot 2 + \sqrt{3} - 1 \leq 2$, a contradiction; or $2t + \sqrt{3} - 1 \leq 2$, that is $t \leq (3-\sqrt{3})/2$, contradicting to $\sqrt{3} - 1 < t$.

Second, suppose that $t_a \leq 2 - t < \sqrt{3} + 1 - t$. Now in LPT there are at least two jobs on each 1-machine. First we rearrange jobs within 1-machines, so that every job that is on a 1-machine in OPT, gets on its final place, and there are still at least two jobs of size $\geq t'_a$ on every 1-machine.

Now we put jobs $\{t^*\}$ from machine m to 1-machines. If $2 \geq t^* > 2 - t$, then we exchange it for two jobs of total size $\geq 2t$.

If $1 \geq t^* > t'_a$, then we exchange it for one job of size at least t. In both cases the size reduction of the t^* is not less than $t/1$, and according to Proposition 3.52, completion time of t^* in LPT is at most $\max(2, t_b + t^*) = 2$.

If $t^* \leq t'_a$, we exchange it for a larger job, so there is no size reduction.

Finally, if $2 - t \geq t^* > 1$, then $t^* \geq t_a$, since t^* was on machine m. In this case t^* has completion time at most $t^* \leq 2 - t$. The size reduction can be $t/(2-t)$.

We get the inequality:

$$\frac{t}{(2-t)} \cdot (2-t) + t \cdot t + (\sqrt{3}-1) \leq 2$$

Solving the inequality yields $-\sqrt{3} \leq t \leq \sqrt{3} - 1$, contradicting to $t > \sqrt{3} - 1$. □

Observe, that the conditions in Instance A of Section 3.5.1 correspond to the case $t_a \leq 2 - t$ of Lemma 3.54, therefore the obtained bounds for t were tight.

Lemma 3.55 *If $\sqrt{3} - 1 < t \leq 1$ and $t_b > 1$, then $\frac{Lpt}{Opt} \geq \frac{\sqrt{3}+1}{2}$ is impossible.*

Proof. We will call the jobs $\{t_a, t_{a+1}, \ldots, t_b > 1\}$ **large jobs**, and all other jobs on 1-machines in LPT **small jobs**. We show that the ratio of changed work is at least $\min(\frac{t}{1}, \frac{\sqrt{3}}{2})$ below time $3-t$; otherwise it is at least $\sqrt{3}/2$. Therefore, inequality (3.f) below models the reduction of finish time of machine m correctly.

Assume that $t_b > 2-t$, that is, all large jobs are very large. We claim that then there is no job that is on a 1-machine in both LPT and OPT. If a large job stayed on a 1-machine, no other job would fit on the same machine, violating the principle of domination. If, e.g., t'_b stayed on a 1-machine, it could only be matched with a job of size at most $2-t < t_b$, again violating the principle of domination. Thus, when we exchange jobs according to OPT, the total work of 1-machines is exchanged for (part of the) work on m. Since machines have finish time $\geq \sqrt{3}$ in LPT, respectively ≤ 2 in OPT, the reduction of work on machine m can not be less than $\sqrt{3}/2$.

Now let $1 < t_b \leq 2-t$. We can rearrange the jobs within 1-machines as follows: we match jobs that are together on a 1-machine in OPT, and delete this machine with the two jobs. Since we did not match two large jobs, now every machine with at least two jobs in LPT, can still have at least one large job and another job, so the total work on any machine *after rearrangement* is at least $1+t > 1+\sqrt{3}-1 = \sqrt{3}$. Machines with one job also have work at least $\sqrt{3}$.

Let first $t_a > 2-t$. Proposition 3.51 implies that small jobs that remain on 1-machines are already deleted. Thus, some of the large jobs will remain on the 1-machine, and all other jobs will be exchanged for jobs on m. If we change the content of a 1-machine completely, that yields a reduction of $\sqrt{3}/2$ in the best case. If the smaller job is exchanged for a job $t'_a < t^* \leq 2-t_b \leq 1$ then we get the ratio $t/1$. Such a job t^* has completion time $\leq \max(2, t_b + t^*) \leq 2-t+1 = 3-t$ in LPT by Proposition 3.52.

Second, if $t_a \leq 2-t$, then every 1-machine has at least two jobs in LPT. Now, after the rearrangement, there is one large job and at least one other job on each 1-machine, and at most one of these jobs stays on the machine. If only the small job or both jobs are exchanged, then the proof is the same as in case $t_a > 2-t$. If only the large job is exchanged for a job $t^* > t_b$, then the ratio is $t_b/t^* \geq 1/(2-t) \geq t/1$. Such a t^* precedes t_a, and has completion time at most $t^* \leq 2-t$ in LPT.

In any of the above cases, the best work reduction on machine m, that we can hope for, is

$$\min(\frac{t}{1}, \frac{\sqrt{3}}{2}) \cdot (3-t) + \frac{\sqrt{3}}{2}(\sqrt{3}+1-3+t) \leq 2. \tag{3.f}$$

Since $\frac{\sqrt{3}}{2} \cdot (\sqrt{3}+1) > 2$, we only need to deal with the inequality

$$t \cdot (3-t) + \frac{\sqrt{3}}{2}(\sqrt{3}-2+t) \leq 2.$$

We obtain the solution: $t > 3.15...$, or $t \leq \frac{6+\sqrt{3}-\sqrt{31-4\sqrt{3}}}{4} \approx 0.7064 < \sqrt{3}-1$, a contradiction. □

3.6 LPT* provides a 2.8-approximation

In this section we return to the case of 2-divisible machines. In Section 3.3 we obtained a worst case ratio of 3 for LPT*. Here we improve on this ratio:

Theorem 3.8 LPT* *is a 2.8-approximation algorithm.*

Theorem 3.8 directly follows from Theorem 3.9 below, basically the same way as described in Corollary 3.10.

Theorem 3.9 *Let* $\langle s_1, \ldots, s_m \rangle$ *and* $\langle t_1, \ldots, t_n \rangle$ *be an instance of* $Q||C_{\max}$. *If the speed vector* $\langle s_1, \ldots, s_m \rangle$ *is 2-divisible, then* $\frac{Lpt}{Opt} < 1.4$

3.6.1 The proof of Theorem 3.9

Just like in Section 3.5.2, we assume that the contrary holds, and we fix a minimal counter-example with 2-divisible machines. Let OPT be an arbitrary optimal schedule of this instance.

The proof technique is similar to that of Theorem 3.7: We start from the LPT schedule, then we rearrange jobs, so that more and more jobs get to their final place in OPT. We delete machines that received all their jobs according to OPT. We strive to get into a state, when the set of remaining machines has more total work than $Opt \cdot S$, where S denotes the sum of speeds of the remaining machines. Recall that $t = t_n$.

Since job sizes can be normalized, we assume w.l.o.g. that $Opt = 2$. Moreover, since machine sizes can be normalized too, we may assume that $1/2 < t \leq 1$. This implies that $1/2$ is the smallest possible size of a nonempty machine in OPT, and the instance is minimal – without empty machines in OPT –, so $s_1 \geq 1/2$. We will call machines of speed at least 2 **fast machines.**

Let f_i denote the finish time of machine i in LPT, *before* t_n *is scheduled.* We assume that $Lpt \geq 2.8$, and the instance was minimal. Consequently, $2.8 > f_i \geq 2.8 - t/s_i$ for $1 \leq i \leq m$.

We start from the LPT schedule, and we exchange the jobs in several rounds. In the first round machines of speed $1/2$ receive their final job (the jobs allocated to $1/2$-machines in OPT), and can be deleted. After this we show, that we got into a similar situation as in Lemmas 3.54 and 3.55. Despite the similarity, we have to deal with two additional difficulties: On the one hand, the first round of exchanges has already resulted in some reduction of work by the time we want to apply the arguments of the lemmas. This is a minor problem, and in most cases it does not affect the original argument. It receives some attention, e.g., in Lemma 3.62. On the other hand, we may have more than one fast machines, and therefore we cannot assume that at the beginning they have finish time $f_i \geq 2.8$ (Recall, that in Section 3.5.2 we could assume $f_m \geq \sqrt{3} + 1$, because at the beginning of the exchanges t_n was put on top of machine m.) The second difficulty is more crucial, and this is the intrinsic reason why the $(\sqrt{3} + 1)/2$ worst case ratio does not hold in case II.

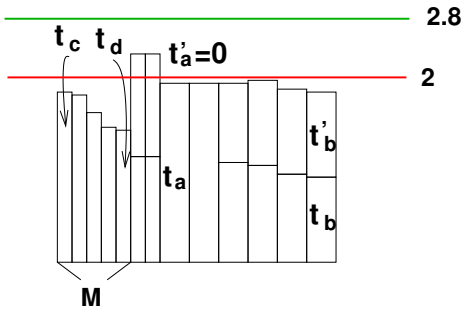

Figure 3.8: Jobs on 1/2-machines and 1-machines in LPT.

As a first step, we delete the job t_n from LPT. Let M denote the (possibly empty) set of 1/2-machines that are assigned only 1 job in LPT, and let $t_c, t_{c+1}, \ldots, t_d$ be these jobs (see Figure 3.8). Then t_d is the smallest among them, moreover $t_d \geq \max(t, 2.8 \cdot 1/2 - t) \geq 1.4/2 = 0.7$.

Now we do the first round of exchanges: In OPT there is one job of size at most 1 on every 1/2-machine. By the principle of domination all of these jobs precede t_c; they are assigned to machines of speed at least 1, and all of them have completion time at most 2 in LPT, otherwise they would have been assigned to a 1/2-machine. (In particular, t_n is not one of these jobs.)

In LPT there is one job of size at least 0.7 on every machine in M. We exchange these jobs for the same number of (single) jobs that are assigned to 1/2-machines in OPT, and then delete all machines of M together with their new job. The resulting schedule will be called LPT_0. Let f_i^0 be the finish time of machine i in LPT_0.

Lemma 3.56 *If $t < 0.8$, then $f_i^0 \geq 2$ for all i.*

Proof. Let $\tau_1, \tau_2, \ldots$ be the jobs assigned to i in LPT; and $\tau_1', \tau_2', \tau_3', \ldots$ be the jobs on i in LPT_0. On any machine i there are at most $2 \cdot s_i$ exchanged jobs, since more jobs cannot precede t_c.

(1) If i is a 1/2-machine with at least two jobs, then $f_i^0 > (2 \cdot 1/2)/(1/2) = 2$.

(2) Assume that $s_i = 1$. Now $f_i \geq 2.8 - t > 2$, so we are done if there is no exchanged job on i. We are also done, if there are at least 4 jobs on i, because $t > 1/2$.

If τ_1 or/and τ_2 is exchanged then $\tau_1 \leq 1$ must hold, and therefore there had to be 3 jobs on i in LPT. Suppose, that $\tau_1' + \tau_2' + \tau_3' < 2$. Since $\tau_1' + \tau_3' \geq t_d + t \geq 1.4$, it follows that $\tau_2' < 0.6 < t_d$. Thus τ_2' was not exchanged, moreover $1 \cdot f_i = \tau_1 + \tau_2 + \tau_3 = \tau_1 + \tau_2' + \tau_3' < 1 + 0.6 + 0.6 = 2.2 = 2.8 - 0.6 \leq 2.8 - t$, a contradiction.

(3) If $s_i = 2$, then $f_i \geq 2.8 - t/2 > 2.8 - 0.8/2 = 2.4$. If there are at most two exchanged jobs on i, then the work of i could be reduced by at most $2 \cdot (1 - 0.7) = 0.6$ Thus, $2f_i^0 \geq 2f_i - 0.6$, that is, $f_i^0 \geq f_i - 0.3 > 2.1$.

If there are at least 3 exchanged jobs on i, then ≥ 3 jobs on i precede t_c, which is possible only if $\tau_1 \leq 1$. Now ≤ 4 jobs do not suffice for $f_i > 2.4$. If there are altogether 5 jobs on i, then $2 \cdot 2.4 < \sum_{v=1}^{5} \tau_v \leq 4 \cdot 1 + \tau_5$, so $0.8 < \tau_5 \leq t_d$. The possible work reduction is at most $4 \cdot (1 - 0.8) = 0.8$, and therefore $f_i^0 \geq f_i - 0.8/2 > 2$. If there

are at least 6 jobs on i, then $2 \cdot f_i^0 \geq \sum_{v=1}^{6} \tau'_v \geq 3t_d + 3t = 3(t_d + t) \geq 3 \cdot 1.4 = 4.2$, so $f_i^0 \geq 2.1$.

(4) If $s_i \geq 4$ then $f_i \geq 2.8 - t/4 > 2.8 - 0.8/4 = 2.6$. The reduced finish time is at least $f_i^0 \geq (0.7/1) \cdot 2 + f_i - 2 > 1.4 + 0.6 = 2$. □

Corollary 3.57 *If $t < 0.8$, then $Lpt/Opt \geq 1.4$ is impossible.*

Proof. Let S denote the sum of speeds of the machines in LPT_0. We obtained that the total remaining work is at least $2 \cdot S + t_n \leq Opt \cdot S = 2S$, a contradiction. □

Lemma 3.58 *Let $0.8 \leq t$. If $f_i^0 < 2$, then $s_i = 1$. Moreover, every 1/2-machine was deleted in the first round.*

Proof. First we claim that in this case LPT assigns one job to every 1/2-machine, so every 1/2-machine was in M, and was later deleted. If two jobs were on a 1/2-machine, then this machine would have total work at least 1.6, and finish time at least $3.2 > 2.8 > f_i$, contradiction.

Second, let $s_i \geq 2$. Since $t_d \geq t$, the possible decreased finish time of i is $f_i^0 \geq (t/1) \cdot 2 + f_i - 2 \geq t \cdot 2 + 0.8 - t/2 = 1.5t + 0.8 \geq 1.5 \cdot 0.8 + 0.8 = 2$. □

The rest of the proof of Theorem 3.9 follows the same lines as the proof in Section 3.5.2. We assume $0.8 \leq t \leq 1$. *Instead of* LPT_0, *our starting schedule is* LPT: We delete all the 1/2-machines and their jobs. On the remaining machines we calculate *with the original sizes* of jobs, as it is in LPT. Nevertheless, we keep in mind, that every job of size ≤ 1, and of completion time ≤ 2 on a 1-machine or on a fast machine, can 'shrink' to size (at least) t_d before putting it to its machine in OPT. Such a shrinkage is equivalent to an exchange for a job on a 1/2-machine in the first round.

We 'put back' the job t_n on top of any fast machine. After that we put jobs from fast machines to 1-machines and vice versa, to have the optimal schedule on 1-machines. However, we will show that the total amount of work on fast machines remains too much to fit in the desired optimum time. Lemmas 3.60 to 3.63 provide essentially the same case analysis as Lemma 3.54 and Lemma 3.55.

We adopt the notation of Section 3.5.2: $t_a, t_{a+1}, \ldots, t_b$ and $t'_a, t'_{a+1}, \ldots, t'_b$ are jobs *on 1-machines in* LPT (see Fig. 3.8). On every 1-machine there is either one job of size at least $2.8 - t \geq 1.8$; or there are at least two jobs. We will use Proposition 3.51, Corollary 3.53 and the following analogue of Proposition 3.52:

Proposition 3.59 *Let t^* be a job assigned to a fast machine in* LPT *and to a 1-machine in* OPT. *Let T^* denote the completion time of t^* in* LPT. *If $t^* > t_a$, then $T^* \leq t^* \leq 2$. If $t^* > t'_a$, then $T^* \leq \max(2, t_b + t^*)$.*

Proof. If t^* is replaced by a job of size $\geq t_d$ in LPT_0, then $T^* \leq 2$. Otherwise essentially the same argument holds as for Proposition 3.52. □

Lemma 3.60 *Let $0.8 \leq t \leq 1$. If $t_b \leq 1$ and $t_a > 2 - t$, then $Lpt/Opt \geq 1.4$ is impossible.*

Proof. Let t^* be a job on a 1-machine in Opt, and on a fast machine with completion time T^* in Lpt. Then either $t^* \geq t_a > 2 - t$ or $t^* \leq t_b \leq 1$. Corollary 3.53 implies $T^* \leq 2$.

We start by rearranging the jobs within 1-machines, according to Opt. Machines that received all their jobs as in Opt, are deleted. By Proposition 3.51, this is possible to do in such a way, that machines with two jobs in Lpt still have two jobs after rearrangement. Moreover, at most one of the two jobs remains on the same 1-machine, and all other jobs will be put to fast machines (see the proof of Lemma 3.54). Single large jobs will not remain on 1-machines by the principle of domination. At this point, every 1-machine has work at least $\min(2t, 1.8)$.

If all the jobs on a 1-machine are put to fast machines, then it receives one job of size at most 2, or two jobs of total size at most $2 \cdot 1$. So the work reduction factor due to such an exchange is not less than $2t/2 = t/1$ or $1.8/2 = 0.9$.

If one job remains on the 1-machine, then its other job of size at least t is put to fast machines, and in return it gets a job of size at most 1. The possible reduction ratio is $t/1$.

Previous exchanges with 1/2-machines in the first round could not decrease this final ratio.

Now we calculate the total work on all fast machines after exchanging jobs with 1-machines. The finish time of any fast machine was $f_i \geq 2.8 - t/s_i \geq 2.8 - t/2$. Let S be the sum of speed of all fast machines. Including t_n, the total work on fast machines was originally at least $t_n + S(2.8 - t/2)$. The reduced work is at least $t_n + S \cdot (2 \cdot \min(t, 0.9) + 0.8 - t/2)$. If $t \leq 0.9$, we get $S(2t + 0.8 - t/2) \geq S(3t/2 + 0.8) \geq S(3 \cdot 0.8/2 + 0.8) = S \cdot 2$ If $0.9 < t$, we get $S(2 \cdot 0.9 + 0.8 - t/2) \geq S(2.6 - 0.5) = S \cdot 2.1$.

In either case, after the exchanges with 1-machines, the total work of fast machines is strictly more than $S \cdot Opt$, a contradiction. □

Lemma 3.61 *Let* $0.8 \leq t \leq 1$. *If* $t_b \leq 1$ *and* $t_a \leq 2 - t$, *then* $Lpt/Opt \geq 1.4$ *is impossible.*

Proof. Since $t_a \leq 2 - t < 2.8 - t$, in Lpt there are at least 2 jobs on every 1-machine. First we rearrange the jobs within 1-machines, so that at least 2 jobs of size $\geq t'_a$ remain on every 1-machine. We match jobs that are on the same 1-machine in Opt and delete these completed machines. Next, we exchange the jobs between fast machines and 1-machines. Let t^* be a job on a 1-machine in Opt, and on a fast machine with completion time T^* in Lpt. We exchange t^* according to one of the following scenarios (see Proposition 3.59):

If $2 - t < t^* \leq 2$, then $T^* \leq 2$, and we exchange t^* for 2 jobs of total size at least $2t$. The reduction factor is $2t/2 = t$.

If $1 < t_a \leq t^* \leq 2 - t$, then $T^* \leq t^*$, and we exchange t^* for one job of size at least t. The reduction factor is $t/(2 - t)$.

If $t'_a < t^* \leq 1$, then $T^* \leq \max(2, t_b + t^*) = 2$, and we exchange t^* for one job of size at least t. The reduction factor is again t.

Finally, if $t^* \leq t'_a$, then we exchange it for a job of size $\geq t'_a$, and there is no size reduction.

Previous exchanges with jobs on 1/2-machines could not yield better ratios.

Let LPT_1 denote the new schedule on the set of fast machines (1-machines are deleted). Let f_i^1 denote the finish time of fast machine i in LPT_1. We claim that $f_i^1 \geq 2$ for every fast machine, and for the machine with t_n on top $f_i^1 \geq 2 + t_n$. This will complete the proof of Lemma 3.61.

If $s_i \geq 4$, then for the finish time in LPT $f_i \geq 2.8 - \frac{t}{4}$ holds. For the reduced finish time we get: $f_i^1 \geq (2-t) \cdot \frac{t}{(2-t)} + t \cdot t + 0.8 - t/4 = t^2 + 0.75t + 0.8 \geq 0.8^2 + 0.75 \cdot 0.8 + 0.8 = 2.04$.

Now let $s_i = 2$. If in LPT_1 there are at least five jobs assigned to i, then it has finish time $f_i^1 \geq 5t/2 \geq (5 \cdot 0.8)/2 = 2$.

Now we assume that $f_i^1 < 2$ and there are at most 4 jobs assigned to i in LPT_1. The original and the new total work assigned to i are $w_i \geq 5.6 - t \geq 4.6$ resp. $w_i^1 < 2 \cdot 2 = 4$. Let $\tau_1, \tau_2, \ldots$ be the jobs assigned to i in LPT (not in this order).

Since $f_i \geq 2.8 - t/2 \geq 2.3$, the last job of i is not an exchanged job. On the other hand, at least one of the jobs must be exchanged.

Suppose that on i there were at most 4 jobs in LPT, and at most 3 of them are exchanged for a job of size t each. If only τ_1 is exchanged, then $4 > w_i^1 = t + w_i - \tau_1 \geq t + 5.6 - t - (2-t) = 3.6 + t$, yielding $0.4 \geq t$, a contradiction. If τ_1 and τ_2 are exchanged, then $4 > w_i^1 = 2t + w_i - (\tau_1 + \tau_2) \geq 2t + 5.6 - t - 2(2-t) = 1.6 + 3t$, that is, $0.8 > t$, contradiction. If τ_1, τ_2 and τ_3 are exchanged, then $5.6 - t \leq w_i = \tau_1 + \tau_2 + \tau_3 + \tau_4 \leq 2(2-t) + 1 + 1 = 6 - 2t$, implying $t \leq 0.4$, contradiction.

The case when a large job of size ≤ 2 is exchanged for $2t$ can be elaborated on using the last two formulas, and the inequality $2 \leq 2(2-t)$. □

In the rest of the proof we will call the jobs $\{t_a, t_{a+1}, \ldots, t_b > 1\}$ **large jobs**, and all other jobs on 1-machines in LPT **small jobs**.

Lemma 3.62 *Let* $0.8 \leq t \leq 1$. *If* $t_b > 1$ *and* $t'_a = 0$ *then Lpt/Opt* ≥ 1.4 *is impossible.*

Proof. Recall that $t'_a = 0$ means that in LPT t_a is a single job, i.e., $t_a \geq 2.8 - t \geq 1.8$.

Due to Proposition 3.51, and because two large jobs do not belong to the same 1-machine in OPT, it is possible to rearrange the jobs within 1-machines as follows: machines with two jobs in LPT still have two jobs, and at least one large job remains on every 1-machine; on some 1-machines with 2 jobs the large job remains on the same machine in OPT, but all other jobs will be exchanged.

Now the total work on 1-machines is at least $1 + t \geq 1.8$. This lower bound holds for machines with single jobs as well. On some 1-machines with 2 jobs the large job remains on the same machine in OPT, but all other jobs will be exchanged.

As a next step, we exchange the jobs between 1-machines and fast machines. Let t^* be a job on a fast machine having completion time T^* in LPT, and be on a 1-machine in OPT. We analyze the possible work reduction in different time zones on the fast machine as follows:

If the jobs on the 1-machine are exchanged completely, in general we get a reduction factor at least $1.8/2 = 0.9$.

If a large job remains on the 1-machine, and t^* is exchanged with another job on this machine, then $t^* \leq 1$, and the reduction factor is $t/1 \geq 0.8$. By Corollary 3.53, $T^* \leq \min(2, t_b + t^*) \leq 2 - t + 1 = 3 - t \leq 3 - 0.8 = 2.2$. (Note that $t_b \leq 2 - t$, otherwise t^* does not fit on top of a large job.) Moreover, when t_a is scheduled in LPT – that is, so far only jobs of size $> 2 - t$ appeared –, fast machines have a finish time at least $t_a/2 \geq 1.8/2 = 0.9$. Thus, t^* must be in the time zone between 0.9 and 2.2

Finally, an anomaly might occur affecting the above factors, due to potential shrinkage of jobs, i.e., the exchanges with 1/2-machines at the very beginning. Suppose that t^* and t^{**} (original size) are on the same 1-machine in OPT, and on fast machines in LPT. If both of them are of size ≤ 1, then the factor remains 1.8/2. On the other hand, let $t^* \leq 1$, and $T^* \leq 2$, and t^* exchanged in the very first round for a job $\geq t$. Then (instead of $t^* + t^{**} \leq 2$) we have the bound $t^{**} \leq 2 - t$, and so $t^* + t^{**} \leq 3 - t$. The reduction factor in this case is $1.8/(3 - t)$.

However, since $t^{**} > t^*$, and $T^* \leq 2$, it is easy to show that in LPT t^{**} has completion time $\leq (2 \cdot 2 - 1 + 2 - t)/2 \leq (5 - 0.8)/2 = 2.1 \leq 2.2$.

In LPT the total work on fast machines is at least $S(2.8 - t/2) + t_n$, where S denotes the total speed of fast machines. Suppose that due to job exchanges the total work reduced to at most $S \cdot 2$. This would imply:

$$\min(t, \frac{1.8}{(3-t)}) \cdot (2.2 - 0.9) + 0.9 \cdot (2.8 - \frac{t}{2} - (2.2 - 0.9)) \leq 2$$

If $1.8/(3-t) > t$, we get $0.85t + 1.35 \leq 2$, that is $t \leq 0.76... < 0.8$, a contradiction.

If $1.8/(3 - t) \leq t$, the inequality is

$$1.3 \cdot \frac{1.8}{3-t} + 0.9 \cdot (1.5 - \frac{t}{2}) \leq 2,$$

and this is false for any real t, so we got a contradiction. □

Lemma 3.63 *Let* $0.8 \leq t \leq 1$. *If* $t_b > 1$ *and* $t'_a \neq 0$, *then Lpt/Opt* ≥ 1.4 *is impossible.*

Proof. In this case there is a large job and a small job on every 1-machine in LPT. We rearrange them according to OPT, so that on each remaining machine there is still a large job and another job of size $\geq t'_a$. Moreover, in OPT, either the large job, or the other job, or none of them stay on the (same) 1-machine, all other jobs will be put on fast machines.

Next, we exchange the jobs between fast machines and 1-machines one by one. Let t^* be a job on a fast machine having completion time T^* in LPT, and be on a 1-machine in OPT. One of the following cases holds:

If $t^* \leq t'_a$, then we exchange it for a larger job, and this means no work reduction on the fast machines.

If $t'_a < t^* \leq 1$, we exchange it for a job of size $\geq t$, and $T^* \leq \max(2, t_b + t^*) \leq t_b + 1$. The reduction factor is $t/1$. Previous shrinkage of t^* to t_d could not improve this factor, since $t'_a \leq t_d$. Moreover, such a t^* must have been scheduled above the time $t_a/2 > 0.5$.

If $1 < t^* \leq t_b$, then t^* takes away the place of a large job, so we exchange it for a large job of size at least t_b, and there is no work reduction.

If $t_b \leq t^* \leq 2 - t$, then again we exchange it for a job of size at least t_b. The reduction factor is $t_b/(2-t) > 1/(2-t) \geq t$. Moreover, $T^* \leq t^* < 2$.

Finally, if $2 - t < t^* \leq 2$, then $T^* \leq 2$, and we t^* is exchanged for two jobs of total size at least $t_b + t$. The reduction factor is not less than $(t_b + t)/2 \geq t_b/(2-t)$; where the latter follows from $t_b + t \leq 2$.

We obtain the following inequality describing the necessary work reduction on fast machines:

$$\frac{t_b}{2-t} \cdot 0.5 + (t_b + 1 - 0.5) \cdot t + [2.8 - \frac{t}{2} - (t_b + 1)] \leq 2.$$

The coefficient of t_b is $[0.5/(2-t)+t-1]$. This coefficient is positive for $0.8 \leq t \leq 1$. Therefore, the inequality should hold if we substitute t_b by $1 < t_b$. The resulting inequality is

$$\frac{0.5}{2-t} + (1.5) \cdot t + 0.8 - \frac{t}{2} \leq 2,$$

solving to either $t \geq 1.6 + \sqrt{0.66}$; or $t \leq 1.6 - \sqrt{0.66} \approx 0.7876...$, a contradiction. □

3.7 Approximation lower bounds of LPT*

Instance A of Section 3.5.1, provides a lower bound on the worst case ratio of LPT not only in the 'one fast machine' case, but also for 2-divisible machines. In view of this, several questions arise about 2-divisible machines: Can the asymptotic lower bound of $(\sqrt{3}+1)/2$ be increased on 2-divisible machines? Or, on the contrary, do we conjecture that the same tight bound holds for 2-divisible machines as for 'one fast machine'? The upper bound of 1.4 for LPT automatically implies the upper bound 2.8 for LPT*. This immediate implication fails concerning the lower bounds. Could it happen, that LPT* has actually a better approximation ratio than twice the ratio of LPT on 2-divisible machines? In this section we answer some of these questions. We show that the bound $(\sqrt{3}+1)/2$ is not tight on 2-divisible machines. Moreover, LPT* is, in fact, 'twice as bad' as LPT, as we demonstrate it in Section 3.7.2. The actual tight bound for LPT* as well as for LPT, remains open.

3.7.1 Improved lower bounds for 2-divisible machines

We describe two instances on 2-divisible machines for which $Lpt/Opt > (\sqrt{3}+1)/2$. Both are refined versions of Instance A of Section 3.5.1. We are able to improve on this lower bound by using two different approaches: in Instance B we manage to exchange jobs larger than $x = 3 - \sqrt{3}$ for jobs of size $t = t_n$. In Instance C we make use of the fact that the last job t is smaller than y so that later even y-jobs above time 2 can be exchanged for smaller ones. Instance B has an approximation bound arbitrarily close to $(\sqrt{409}+29)/36 \approx 1.3673 > (\sqrt{3}+1)/2$. However, this instance is not suitable if in LPT ties are broken in favour of slow machines. Therefore we also present Instance C, which is valid for any kind of tie-breaking and has approximation bound arbitrarily close to $955/699 \approx 1.3662 > (\sqrt{3}+1)/2$.

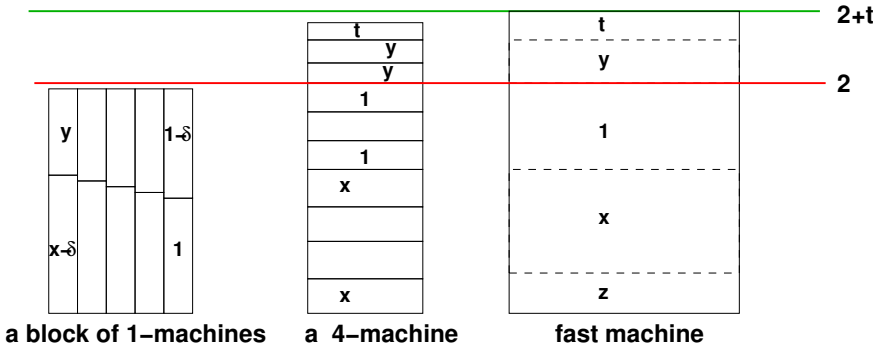

Figure 3.9: LPT schedule modulo the last job in Instance B.

Besides providing a slightly better lower bound than $(\sqrt{3}+1)/2$, these two examples are of interest, because they also give an insight into the potential difficulties in determining a more exact upper bound for LPT on 2-divisible machines.

Theorem 3.10 *If we restrict the problem $Q||C_{\max}$ to 2-divisible speed vectors, then the ratio Lpt/Opt can be arbitrarily close to* $955/699 \approx 1.3662$. *Moreover, if in* LPT *ties are broken in favour of faster machines, then the same asymptotic worst case ratio is at least* $(\sqrt{409}+29)/36 \approx 1.3673$.

Proof. The proof is given by instances B and C:

Instance B. In this case we assume that in LPT ties are always broken in favour of faster machines. Just like in Instance A, we have plenty of blocks of 1-machines and a fast machine of speed 2^r. Moreover, we have several 4-machines. First we describe the assignment of jobs in LPT (see Fig 3.9): Let $x = 1.25$ and $y = 0.75$. A block of 1-machines is scheduled like in Instance A: The large jobs in a block range from $x - \delta$ down to 1; the small jobs range from y up to $1 - \delta$. Every block will be later used for exchanging an x-job for a 1-job or exchanging a 1-job for a y-job.

On any 4-machine in LPT there are 10 jobs: 4 jobs of size x; 3 jobs of size 1; 2 jobs of size y and 1 job of size $t \leq y$, where $t = t_n$. The total work on a 4-machine is $9.5 + t$.

Let $z = 8 - 9t \geq 8 - 9y = x$. On the fast machine there are $2^r/4$ jobs of size z. After that, it is filled up with x-jobs until time x; with 1-jobs until time 2; with y-jobs until time $9.5/4$ and with t-jobs until $2 + t - \delta$. The total work on the fast machine is at least $2^r(2 - \delta + t) - t$.

It is straightforward to check that this is an LPT schedule, either by setting $1/2^r < \delta$, like in Instance A, or by allowing one more job of size between 1 and y on the fast machine. Finally, a last job of size t is assigned to a 1-machine yielding makespan $2+t-\delta$. On the fast machine this job would be completed after $2+t-\delta$; on a 4-machine, the finish time would be $(9.5 + 2t)/4 \geq 2 + t$, where the last inequality holds because $t \leq y = 0.75$.

The goal is to determine a possibly large t value, so that the optimum makespan can be arbitrarily close to 2. In order to get an optimum makespan we rearrange

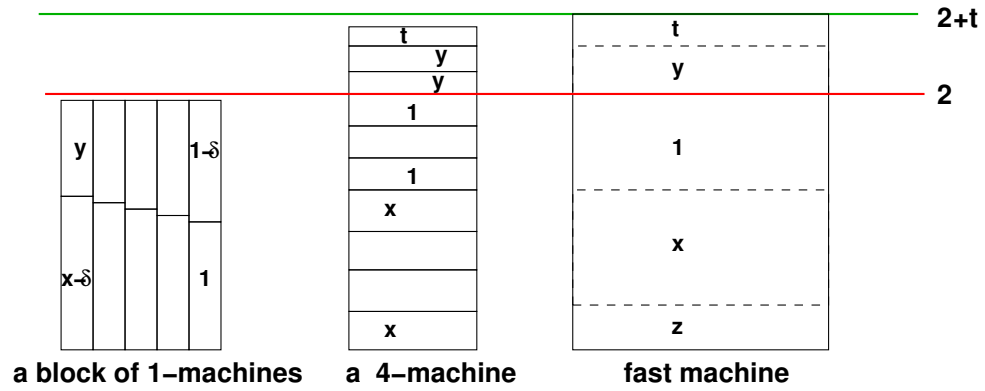

Figure 3.10: Instance C: The completion times of different job types in LPT.

the jobs as follows. As a first step, we put the very last job on the fast machine. Second, we exchange every job of size x or 1 (on 4-machines and the fast machine) for a job of size y using the blocks of 1-machines. At this point, on every 4-machine there are 9 jobs of size y and 1 job of size t. Since $(10 \cdot y)/4 < 2$, the job of size t can be exchanged for a y without violating the desired optimum makespan. Thus, we can use 4-machines to exchange all y-jobs on the fast machine for t-jobs. Moreover, on $2^r/4$ of the 4-machines we also exchange all the jobs for t-jobs. On these $2^r/4$ machines we will have 10 jobs of size t. Finally we use each of these $2^r/4$ machines for exchanging a z-job for a t-job. This is possible, since $(9 \cdot t + z)/4 = 2$. Now every job on the fast machine is exchanged for a t-job. If we may calculate with fractional jobs on the fast machine, the following inequality models the desired shrinkage of finish time (we calculate with no shrinkage above time 2):

$$\frac{z}{4} \cdot \frac{t}{z} + (x - \frac{z}{4}) \cdot \frac{t}{x} + (2 - x) \cdot \frac{t}{1} + t \leq 2$$

$$\frac{t}{4} + t - \frac{(8 - 9t)t}{4 \cdot 1.25} + 0.75t + t \leq 2 \tag{3.g}$$

By solving the inequality we get: $\frac{-\sqrt{409}-7}{18} \leq t \leq \frac{\sqrt{409}-7}{18} \approx 0.734$

Using $t = \frac{\sqrt{409}-7}{18}$ yields the approximation $(2 + t)/2 = (\sqrt{409} + 29)/36$. The surplus on the optimum makespan due to calculating with fractional jobs and due to the very last job is not more than $(x + 1 + t)/2^r < \epsilon$ if r is large enough.

Instance C. In this example it is irrelevant how ties are broken in LPT. We have blocks of 1-machines like before, furthermore we have several 16-machines and one 1024-machine. Let $x = 377/300 \approx 1.2567$ and $y = 2 - x = 223/300 \approx 0.7433$.

In LPT the blocks of 1-machines are scheduled like in the previous instances, and each block can be used to exchange a job of size x for a job of size 1 or a job of size 1 for a job of size y; 1-machines have finish time $2 - \delta$, where $\delta > 0$ is an arbitrarily small number which divides $x - 1$ evenly.

Every 16-machine receives 15 jobs of size x; 13 jobs of size 1; 14 jobs of size y and 1 job of size t, where t is the smallest job size. The total work on any 16-machine is $15x + 13 + 14y + t = 77/300 + 42 + t$.

The 1024-machine has 1023 jobs of size x; 762 jobs of size 1; 947 jobs of size y and 63 jobs of size t. The total work on the 1024-machine is $1023x + 762 + 947y + 63t = 38/75 + 2751 + 63t$.

The assignment of different types of jobs in LPT is depicted in Figure 3.7.1. One can easily verify that one more x-job would finish at time x; one more 1-job would finish above time 2; and one more y-job would finish after the y-jobs on other machines. The first jobs of 1-machines are scheduled after the x-jobs and before the 1-jobs; the second jobs of 1-machines after the 1-jobs and before the y-jobs, because even the first y-job on the fast machines is completed after time 2.

If the last job of size t is assigned to a 1-machine, then it has finish time $t+2-\delta$, where δ can be arbitrarily small. Being assigned to a 16-machine, it would have finish time $(77/300 + 42 + 2t)/16$; on the 1024-machine it would have finish time $(38/75 + 2751 + 64t)/1024$. The condition $(77/300 + 42 + 2t)/16 \geq 2 + t$ holds if $t < 0.7326$; whereas $(38/75 + 2751 + 64t)/1024 \geq 2 + t$ holds if $t < 0.7328$.

In order to get the optimum schedule, we place the last job on the 1024-machine, and we exchange all jobs of size x or 1 for jobs of size y, using the blocks of 1-machines. At this point the 16-machines have 42 jobs of size y and one job of size t. Since $43y < 16 \cdot 2$, we can exchange y-jobs on the 1024-machine for the t-jobs on 16-machines. Thus, every job on the 1024-machine could be exchanged for a t-job. Now the total work on the 1024 machine is $2796t \leq 1024 \cdot 2$ if $t \leq 512/699 \approx 0.73247$. Consequently, if we take $t = 512/699$ then $Lpt/Opt = (2+t-\delta)/2 = 955/699-\delta/2 > (\sqrt{3}+1)/2$ for small enough δ. □

3.7.2 Lower bound for LPT*

We consider the monotone algorithm LPT* (see Section 3.2.3). Recall that this algorithm receives arbitrary input speeds $\langle \sigma_1, \ldots, \sigma_m \rangle$ which, in the first step, are rounded down to the next power of 2, so as to get a 2-divisible speed vector $\langle s_1, \ldots, s_m \rangle$. After that LPT is run with the rounded speeds. Finally, the works (sets of jobs) assigned to each machine are ordered increasingly among machines having the same rounded speed. Obviously, the actual finish times and the makespan Lpt^* depend on the true speeds $\langle \sigma_1, \ldots, \sigma_m \rangle$.

Observe that, as opposed to the upper bounds, a worst case ratio $Lpt/Opt > \alpha$ does not immediately imply $Lpt^*/Opt > 2\alpha$. Here we show how to modify Instance A to get $Lpt^*/Opt > \sqrt{3} + 1 - \epsilon$ in the worst case. For sake of simplicity here we do not consider the other Instances B and C; however, these can be modified in a similar way in order to obtain even higher lower bounds.

Theorem 3.11 *For arbitrary $\epsilon > 0$ an instance of $Q||C_{max}$ exists, s.t. on this instance $Lpt^*/Opt > \sqrt{3} + 1 - \epsilon$.*

Proof. Consider Instance A of Section 3.5.1. Notice, that even without the very

last job, the instance proves $Lpt/Opt > (\sqrt{3}+1)/2 - \epsilon$ for every $\epsilon > 0$, since the fast machine has finish time $f_m > \sqrt{3}+1-\delta-y/2^r > \sqrt{3}+1-2\delta$.

Let Instance A' = Instance A $-\{$the last y-job$\}$.

In the following we define Instance D. After rounding the machine speeds, the instance will consist of k copies of Instance A', where $k \approx 2{\cdot}2^r$. Consequently, running LPT with the rounded speeds as input, results in k copies of the LPT schedule of Instance A'. We need that $Lpt^* > \sqrt{3}+1-\epsilon'$, and $Opt < 1+\epsilon'$ holds for some appropriate ϵ'. Therefore, one of the large machines m_1 will have $\sigma_{m_1} = s_{m_1} = 2^r$. For all the other fast and 1- machines let the true speed be $\sigma_i = 2s_i - \epsilon''$. The optimal schedule with the rounded speeds has makespan less than 2. With the original speeds the finish times can become less than 1, except for machine m_1. So each of the other $k-1$ fast machines takes over a y-job from m_1, so that $Opt < 1+\epsilon'$ holds. □

3.8 The truthful mechanism

In this last section we complete our monotone algorithm LPT* into a deterministic truthful 2.8-approximation mechanism. To this end, we will discuss how to compute the payments, and characterize the payment function. In particular, we will treat the questions of *voluntary participation* and *frugality* (see Section 3.1.1). The argument will be very similar to that of Archer and Tardos [5], even though we apply it for a different algorithm. Our results concerning the payments compare well with those of [5]. The overall results about the truthful mechanism are summarized by Theorem 3.12 and Corollary 3.65.

Given the bid vector $\langle b_1, b_2, \ldots, b_m\rangle$ $(b_i \geq b_{i+1})$ for the inverse speeds, algorithm LPT* computes a schedule, and Theorem 3.5 guarantees, that for all i and b_{-i}, the work curve $w_i(b_{-i}, b_i)$ is a decreasing function of b_i. In order to obtain a truthful mechanism with voluntary participation we have to determine the payment scheme as given by Theorem 3.1:

$$P_i(b_{-i}, b_i) = b_i w_i(b_{-i}, b_i) + \int_{b_i}^{\infty} w_i(b_{-i}, u)\, du. \tag{3.h}$$

In what follows, we fix machine i and the bids b_{-i}, *and* b_i. We will analyze the function $w_i(b_{-i}, u)$ on the interval (b_i, ∞), as determined by LPT*. We are going to show that $\int_0^\infty w_i(b_{-i}, u)\, du < \infty$, moreover our work curve $w_i(b_{-i}, u)$ is a simple step function, and $P_i(b_{-i}, b_i)$ is efficiently computable. Finally, in Section 3.8.1 we prove that our mechanism is *frugal* (cf. Section 3.1.1). When dealing with these issues, for the mechanism it is irrelevant whether b_i or u denotes the real inverse speed or some arbitrary bid. For ease of presentation, from now on we assume that the b stand for the inverse speeds, i.e., $b_h = 1/\sigma_h$ and $s_h = 2^{\lfloor \log \sigma_h \rfloor}$ $(1 \leq h \leq m)$. Furthermore, $u = 1/\sigma$, where $\sigma \in (0, \infty)$ stands for the variable speed of i; and $s = 2^{\lfloor \log \sigma \rfloor}$ is the respective rounded speed.

Let $W = \sum_1^n t_j$ denote the total work. Clearly, i receives work of at most W. On the other hand, if $t_n/s > W/s_m$ then i receives no work at all. For $i \neq m$ this yields the value $y_i := \frac{W}{s_m \cdot t_n}$, so that $w_i = 0$ if $u > y_i$. Consequently,

$\int_0^\infty w_i(b_{-i}, u)\, du < \infty$. For $i = m$ we can take $y_m := \frac{W}{s_{m-1} \cdot t_n}$.

Let us consider the function $w_i(b_{-i}, u)$ on $[b_i, y_i]$. The *schedule* LPT* is constant on any interval $u \in (2^l, 2^{l+1}]$. Inside the interval, w_i depends on the order of machine speeds, so w_i changes only in the points $u = b_h$ for some $h \neq i$. On any subinterval $(b_h, b_{h-1}) \subset (2^l, 2^{l+1}]$ the function $w_i(b_{-i}, u)$ is constant. The number of breaks in the function is at most $m + \log(y_i/b_i) \leq m + \log \frac{s_i \cdot W}{s_m \cdot t_n} \leq m + \log \frac{W}{t_n}$ so P_i is polytime computable. Moreover, the computation is essentially the same for all machines having the same rounded speed. For $i = m$ we have at most $m + \log \frac{s_m \cdot W}{s_{m-1} \cdot t_n}$ breakpoints. We obtained the following:

Theorem 3.12 *Let $\mathcal{M}$ be a mechanism using the monotone algorithm* LPT*, *and the payment scheme $\mathcal{P} = (P_1, \ldots, P_m)$ as given by (3.h). Then $\mathcal{P}$ is computable in polynomial time, moreover $\int_0^\infty w_i(b_{-i}, u)\, du < \infty$ for all machines i, so the mechanism admits voluntary participation.*

3.8.1 Frugality

Recall from Section 3.1.1, that Archer and Tardos call a mechanism **frugal**, if the sum of the payments does not exceed the total cost by more than a logarithmic factor. In particular, we saw that (unless the fastest speed may dominate arbitrarily) their randomized monotone algorithm is frugal, in that the ratio of the total payment and total cost is at most $\mathcal{O}(\frac{\sigma_m}{\sigma_{m-1}} \ln(\frac{t_1}{t_n} \cdot n))$.

Our results are similar to the frugality results of [5] (cf. Theorem 3.2 and Corollary 3.65). Due to the different approach using LPT, and some inconveniences with rounded machine speeds, we need a slightly more careful elaboration.

Lemma 3.64 *Let* LPT* *be the monotone algorithm used by the mechanism, and P_i denote the payment to machine i, as determined by (3.h).*
(i) If $i \neq m$, then either $P_i \leq b_i w_i [1 + 9 \cdot \log \frac{2y_i}{b_i}]$;
or $P_i \leq b_i w_i + b_{i+1} w_{i+1} \cdot \frac{9}{2} \cdot \log \frac{2y_i}{b_i}$.
(ii) If $i \neq 1$, then $P_i \leq b_i w_i [1 + (6 + \frac{3}{2} \frac{\sigma_i}{s_{i-1}}) \log \frac{2y_i}{b_i})]$.

Proof. The payment to machine i is given by formula (3.h). The term $b_i w_i$ equals the (assumed) finish time, so it compensates the machine for her cost, whereas the integral term stands for the profit. We need to upper bound $\int_{b_i}^\infty w_i(b_{-i}, u)\, du$ in terms of the finish time $b_i w_i$. Recall that if $1/\sigma \geq y_i$ then w_i disappears. Moreover, the *schedule* changes only if the rounded speed s changes. We will analyze the change in the schedule as s changes through $s_i, s_i/2, s_i/4, \ldots$, etc.

Let us consider the fixed schedule LPT with rounded input speeds and $s = s_i$, with work reordered according to step 3. of LPT*. In this schedule, $f_i = \frac{w_i}{s_i} \leq \frac{2w_i}{\sigma_i} = 2b_i w_i$. We fix this f_i value, a machine set L, and a job set $\mathcal{T}_L$. Let a machine h be in the set L, if $f_h \leq 2f_i$. It is easy to see that, $\{1, \ldots, i-1, i\} \subseteq L$. Let $\mathcal{T}_L$ be the set of all jobs on these machines, and I be the instance restricted to L and $\mathcal{T}_L$. Now $Lpt_I \leq 2f_i$, where Lpt_I is the makespan of I. In other words, $f_i = \alpha Lpt_I$, where $1/2 \leq \alpha \leq 1$. Suppose that t_j is the first job assigned to i. It is straightforward to show that all jobs with index $\geq j$ belong to $\mathcal{T}_L$.

Now we decrease s, and observe the change in the schedule of I. As s decreases, the (large) jobs *not* in $\mathcal{T}_L$ are still assigned to the same machines *not* in L. Therefore, machines in L always receive a subset of $\mathcal{T}_L$. Let $Opt(s)$ be the optimum makespan of I as a function of s. By Theorem 3.3, the work assigned to machine i is at most $w_i(s) \leq s \cdot 3/2 \cdot Opt(s)$.

If a machine $h^* \in L$ exists s.t. $h^* \neq i$ and $s_{h^*} \geq s_i$, then decreasing s can result in a new optimum of at most $Opt(s) \leq Lpt_I(1+\alpha) = f_i(1/\alpha+1) \leq 3f_i$, since in the worst case h^* receives all the jobs from i. So we obtain

$$w_i(s) \leq s \cdot \frac{3}{2} Opt(s) \leq \frac{3}{2} \cdot 3f_i \cdot s \leq \frac{9}{2} \cdot 2b_i w_i \cdot s = 9b_i w_i \cdot s,$$

and therefore

$$\int_{b_i}^{\infty} w_i(b_{-i}, u)\, du \leq 9b_i w_i \int_{b_i}^{y_i} 2^{\lfloor \log 1/u \rfloor}\, du \leq 9b_i w_i \log \frac{2y_i}{b_i}.$$

(i) Suppose that h^* does not exist. We replace h^* by $i+1$. Since h^* does not exist, $f_{i+1} > 2f_i \geq Lpt_I$. Our new restricted instance I' consists of $L \cup \{i+1\}$ and $\mathcal{T}_L \cup \{jobs\ on\ i+1\}$. If $Opt'(s)$ denotes the optimum of I', we get $Opt'(s) \leq f_{i+1} + f_i \leq 3f_{i+1}/2$ and $w_i(s) \leq \frac{3}{2} \cdot \frac{3}{2} f_{i+1} \cdot s$. Consequently,

$$\int_{b_i}^{\infty} w_i(b_{-i}, u)\, du \leq \frac{9}{4} f_{i+1} \cdot \log \frac{2y_i}{b_i} \leq \frac{9}{2} b_{i+1} w_{i+1} \cdot \log \frac{2y_i}{b_i}.$$

(ii) Recall that $i-1 \in L$. If we replace machine h^* with $i-1$, we get the following: $Opt(s) \leq f_i(1/\alpha + s_i/s_{i-1}) \leq f_i(2 + s_i/s_{i-1})$. Using $f_i = b_i w_i \sigma_i / s_i$, and $\sigma_i/s_i < 2$, we obtain $w_i(s) \leq \frac{3}{2} b_i w_i (4 + \sigma_i/s_{i-1}) \cdot s$ and

$$\int_{b_i}^{\infty} w_i(b_{-i}, u)\, du \leq b_i w_i [6 + \frac{3}{2} \frac{\sigma_i}{s_{i-1}}] \log \frac{2y_i}{b_i}. \quad \square$$

Corollary 3.65 *Let $r_1 := t_1/t_n$, and $r_2 := \sigma_m/\sigma_{m-1}$. The payment to machine m might be more than the cost of the machine by a factor of at most $\mathcal{O}(r_2 \log(r_2 n r_1))$. The total payment to all machines $i \neq m$ is at most $\mathcal{O}(\log(\frac{r_1 n}{r_2}))$ times more than the total cost of all the machines.*

3.9 Discussion

Monotone algorithms. There are plenty of questions left open concerning monotone algorithms, even if we restrict our attention to the $Q||C_{\max}$ problem. We know from [3], that if the number of machines, m, is constant, then there is a monotone FPTAS for this problem. In case m is part of the input, it would be nice to improve on our worst case ratio of 2.8 for deterministic, respectively on the ratio of 2 for randomized monotone algorithms [4]. We conjecture that it is possible to obtain a deterministic worst case ratio close to 2, e.g., by designing a monotone version of a better approximating algorithm – like the PTAS in [53] –, for 2-divisible machines. On the other hand, it seems to be more of a challenge either to find an efficient

monotone algorithm with approximation ratio below 2, or to prove some approximation lower bound for polynomial time monotone algorithms. Such a lower bound was proven for the problem $Q||\sum w_jC_j$ in [5] (cf. Section 3.1), by excluding the monotonicity of *any* allocation with approximation ratio better than $\frac{2}{\sqrt{3}}$. We cannot hope for this kind of solution for the problem $Q||C_{\max}$, since here even a certain type of *optimal* allocation is monotone.

One could be tempted to prove monotonicity *and* good approximation bound of LPT for c-divisible machines, where $c < 2$. However, it seems unlikely (we may, of course, be wrong) that radically simpler or shorter ways can be found to prove such a result, than those used in the thesis. Moreover, one cannot hope for a better c value than $c = \frac{3+\sqrt{17}}{4} \approx 1.78$, since it is shown in [2], that even for 2 machines, LPT is not monotone if $c < \frac{3+\sqrt{17}}{4}$. On the other hand, it might be feasible to improve on the latter bound, or in the best case, to show that LPT is not monotone on c-divisible machines, for $c < 2$.

Worst case ratio of LPT. For the classic LPT algorithm, we have shown a tight asymptotic approximation bound of $\frac{\sqrt{3}+1}{2} \approx 1.3660$ in case of one fast machine; and 'nearly' tight lower and upper bounds, $[1.3673, 1.4]$ for the same problem on 2-divisible machines. Notice that both Instances B and C in Section 3.7.1 apply three different machine speeds. It would be interesting to know whether the tight approximation ratio of $\frac{\sqrt{3}+1}{2}$ holds for machine speed vectors involving only two different speeds, i.e., vectors of the form $\langle 1, \ldots, 1, s, \ldots, s\rangle$.

In our instances providing approximation within ϵ distance to the lower bounds, the speed of the fast machine (and the ratio s_m/s_1) is $s_m = \mathcal{O}(2^r) = \mathcal{O}(1/\epsilon)$. The number of machines is $m = \mathcal{O}(2^r/\delta) = \mathcal{O}(1/\epsilon^2)$, since there are about as many blocks as jobs on the fast machine, and $\mathcal{O}(1/\delta)$ machines per block. These orders of magnitude are the same for Instances A, B, and C. However, Instance D of Section 3.7.2 contains $\mathcal{O}(2^r)$ copies of Instance A, thus altogether $m = \mathcal{O}(1/\epsilon^3)$ machines.

For relatively large ϵ it is easy to create instances with less machines. For example, if we just want to demonstrate $Lpt/Opt > 4/3$, then it is sufficient to modify Instance A by taking one 4-machine, ten blocks of six 1-machines each, and job sizes $x = 1.28$ and $y = 0.72$. We do not exclude, that – if LPT prefers faster machines in case of ties –, Instance B of Section 3.7.1 actually yields the basic construction for a tight bound on 2-divisible machines. However, proving such a tight bound – if at all possible –, seems to require a lengthy and technical elaboration. (The bound itself can be a bit higher than $(\sqrt{409}+29)/36$. When we optimized the smallest job size t in Instance B, we solved inequality (3.g). If we also calculate with the shrinkage above time 2, we obtain another inequality of degree 3, and one root of this will be the optimal job size t.)

Finally, let us summarize the results relative to the sort of tie-breaking used in LPT: The monotonicity and the upper bound proofs hold for both (from left or from right) kinds of tie-breaking. Except for Instance B, the lower bound instances are valid if ties are broken arbitrarily.

Bibliography

[1] F. Afrati, E. Bampis, A. Fishkin, K. Jansen, and C. Kenyon. Scheduling to minimize the average completion time of dedicated tasks. In *Proc. 20th Found. Softw. Techn. and Theor. C. S.*, volume 1974 of *LNCS*, pages 454–464, 2000.

[2] P. Ambrosio and V. Auletta. Deterministic monotone algorithms for scheduling on related machines. In *Proc. 2nd Wshop. Approx. and Online Alg. (WAOA)*, volume 3351 of *LNCS*, pages 267–280. Springer, 2004.

[3] N. Andelman, Y. Azar, and M. Sorani. Truthful approximation mechanisms for scheduling selfish related machines. In *Proc. 22nd Ann. Symp. on Theor. Asp. of Comp. Sci. (STACS)*, volume 3404 of *LNCS*, pages 69–82. Springer, 2005.

[4] A. Archer. *Mechanisms for Discrete Optimization with Rational Agents*. PhD thesis, Cornell University, 2004.

[5] A. Archer and É. Tardos. Truthful mechanisms for one-parameter agents. In *Proc. 42nd IEEE Symp. on Found. of Comp. Sci. (FOCS)*, pages 482–491, 2001.

[6] A. Archer and É. Tardos. Frugal path mechanisms. In *Proc. 13th SIAM Symp. on Disc. Algs. (SODA)*, pages 991–999, 2002.

[7] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proc. 33rd IEEE Symp. on Found. of Comp. Sci. (FOCS)*, pages 14–23, 1992.

[8] V. Auletta, R. De Prisco, P. Penna, and G. Persiano. Deterministic truthful approximation mechanisms for scheduling related machines. In *Proc. 21st Ann. Symp. on Theor. Asp. of Comp. Sci. (STACS)*, volume 2996 of *LNCS*, pages 608–619. Springer, 2004.

[9] J. Bar-Ilan and D. Peleg. Distributed resource allocation algorithms. In *Proc. 6th Intl. Wshop. on Distributed Algorithms*, pages 277–291, 1992.

[10] A. Bar-Noy, M. Bellare, M.M. Halldórsson, H. Shachnai, and T. Tamir. On chromatic sums and distributed resource allocation. *Information and Computation*, 140:183–202, 1998.

[11] A. Bar-Noy, M.M. Halldórsson, G. Kortsarz, R. Salman, and H. Shachnai. Sum multi-coloring of graphs. In *Proc. 7th Europ. Symp. on Algs. (ESA)*, volume 1643 of *LNCS*, pages 390–401. Springer, 1999.

[12] A. Bar-Noy, M.M. Halldórsson, G. Kortsarz, H. Shachnai, and R. Salman. Sum multicoloring of graphs. *Journal of Algorithms*, 37:422–450, 2000.

[13] A. Bar-Noy and G. Kortsarz. The minimum color-sum of bipartite graphs. *Journal of Algorithms*, 28:339–365, 1998.

[14] A. Bar-Noy, A. Mayer, B. Schieber, and M. Sudan. Guaranteeing fair service to persistent dependent tasks. In *Proc. 6th SIAM Symp. on Disc. Algs. (SODA)*, pages 243–252, 1995.

[15] M. Bell. Future directions in traffic signal control. *Transportation Research Part A*, 26:303–313, 1992.

[16] S. Bikhchandani, S. Chatterji, R. Lavi, A. Mu'alem, N. Nisan, and A. Sen. Weak monotonicity characterizes deterministic dominant-strategy implementation. *Econometrica*, 74(4):1109–1133, 2006.

[17] J. Błażewicz, M. Drozdowski, and K.H. Ecker. Management of resources in parallel systems. In J. Błażewicz et al., editor, *Handbook on parallel and distributed processing*, pages 281–297. Springer, Berlin, 2000.

[18] J. Błażewicz, K.H. Ecker, E. Pesch, G. Schmidt, and J. Węglarz. *Scheduling computer and manufacturing processes.* Springer, Berlin, 1996.

[19] H.L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2):1–45, 1998.

[20] P. Brucker. *Scheduling Algorithms. 2nd ed.* Springer, Heidelberg, 1998.

[21] P. Brucker and A. Krämer. Polynomial algorithms for resource-constrained and multiprocessor task scheduling problems. *European Journal of Operational Research*, 90:214–226, 1996.

[22] D. Bullock and C. Hendrickson. Roadway traffic control software. *IEEE Trans. Control Systems Technology*, 2:255–264, 1994.

[23] K. Chandy and J. Misra. The drinking philosophers problem. *ACM Trans. on Programming Languages and Systems*, 6:632–646, 1984.

[24] J. Chen, I. Cidon, and Y. Ofek. A local fairness algorithm for gigabit LANs/MANs with spatial reuse. *IEEE Journal on Selected Areas in Communication*, 11(8):1183–1192, 1993.

[25] Y. Cho and S. Sahni. Bounds for list schedules on uniform processors. *SIAM Journal on Computing*, 9(1):91–103, 1980.

[26] M. Choy and K. Singh. Efficient fault-tolerant algorithms in distributed systems. In *Proc. 24th Ann. ACM Symp. on the Theory of Comput. (STOC)*, pages 593–602, 1992.

[27] G. Christodoulou, E. Koutsoupias, and A. Vidali. A lower bound for scheduling mechanisms. In *Proc. 18th SIAM Symp. on Disc. Algs. (SODA)*, 2007.

[28] E. Clarke. Multipart pricing of public goods. *Public Choice*, 8:17–33, 1971.

[29] E.G. Coffman Jr., M.R. Garey, and D.S. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7(1):1–17, 1978.

[30] E.G. Coffman Jr., M.R. Garey, D.S. Johnson, and A.S. LaPaugh. Scheduling file transfers. *SIAM Journal on Computing*, 14(3):744–780, 1985.

[31] R.W. Conway, W.L. Maxwell, and L.W. Miller. *Theory of scheduling*. Addison-Wesley, 1967.

[32] E.W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Informatica*, 1:115–138, 1971.

[33] G. Dobson. Scheduling independent tasks on uniform processors. *SIAM Journal on Computing*, 13(4):705–716, 1984.

[34] M. Drozdowski. Scheduling multiprocesor tasks – an overview. *European Journal of Operational Research*, 94:215–230, 1996.

[35] P. Erdős, E. Kubicka, and A. Schwenk. Graphs that require many colors to achieve their chromatic sum. *Congressus Numerantium*, 71:17–28, 1990.

[36] U. Feige and J. Kilian. Zero knowledge and the chromatic number. *Journal of Computer and System Sciences*, 57(2):187–199, 1998.

[37] D.K. Friesen. Tighter bounds for LPT scheduling on uniform processors. *SIAM Journal on Computing*, 16(3):554–560, 1987.

[38] R. Gandhi, M.M. Halldórsson, G. Kortsarz, and H. Shachnai. Improved bounds for the sum multicoloring problem and scheduling dependent jobs with minsum criteria. In *Proc. 2nd Wshop. Approx. and Online Alg. (WAOA)*, volume 3351 of *LNCS*, pages 68–82. Springer, 2004.

[39] M.R. Garey and D.S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-completeness*. Freeman, San Francisco, 1979.

[40] K. Giaro, R. Janczewski, M. Kubale, and M. Malafiejski. A 27/26-approximation algorithm for the chromatic sum coloring of bipartite graphs. In *Proc. 5th Intl. Wshop. on Approx. Algs. for Comb. Opt.*, pages 135–145, 2002.

[41] M. Gonen. *Coloring Problems on Interval Graphs and Trees*. M.Sc. Thesis. School of Comp. Sci., The Open Univ, Tel Aviv, 2001.

[42] T. Gonzalez, O.H. Ibarra, and S. Sahni. Bounds for LPT schedules on uniform processors. *SIAM Journal on Computing*, 6(1):155–166, 1977.

[43] R.J. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnoy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Mathematics*, 5:287–326, 1979.

[44] R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[45] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429, 1969.

[46] T. Groves. Incentives in teams. *Econometrica*, 41(4):617–631, 1973.

[47] H. Hajiabolhassan, M.L. Mehrabadi, and R. Tusserkani. Minimal coloring and strength of graphs. *Discrete Mathematics*, 215(1-3):265–270, 2000.

[48] M.M. Halldórsson. Approximating weighted independent set and hereditary subset problems. In *Proc. 5th Intl. Computation and Combinatorics Conf. (COCOON)*, volume 1627 of *LNCS*, pages 261–270. Springer, 1999.

[49] M.M. Halldórsson and G. Kortsarz. Tools for multicoloring with applications to planar graphs and partial k-trees. *Journal of Algorithms*, 42(2):334–366, 2002.

[50] M.M. Halldórsson, G. Kortsarz, A. Proskurowski, R. Salman, H. Shachnai, and J. A. Telle. Multicoloring trees. *Inform. and Comput.*, 180(2):113–129, 2003.

[51] M.M. Halldórsson, G. Kortsarz, A. Proskurowski, R. Salman, H. Shachnai, and J.A. Telle. Multi-coloring trees. In *Proc. 5th Intl. Comput. and Combin. Conf. (COCOON), Tokyo*, volume 1627 of *LNCS*, pages 271–280. Springer, 1999.

[52] M.M. Halldórsson, G. Kortsarz, and H. Shachnai. Sum coloring interval and k-claw free graphs with application to scheduling dependent jobs. *Algorithmica*, 37(3):187–209, 2003.

[53] D.S. Hochbaum and D.B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM Journal on Computing*, 17(3):539–551, 1988.

[54] E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM*, 23:317–327, 1976.

[55] S. Irani and V. Leung. Scheduling with conflicts, and applications to traffic signal control. In *Proc. 7th SIAM Sym. Disc. Alg. (SODA)*, pages 85–94, 1996.

[56] K. Jansen. The optimum cost chromatic partition problem. In *Proc. 3rd Italian Conf. on Algs. and Complexity (CIAC)*, volume 1203 of *LNCS*, pages 25–36. Springer, 1997.

[57] K. Jansen. Approximation results for the optimum cost chromatic partition problem. *Journal of Algorithms*, 34(1):54–89, 2000.

[58] Y.A. Kim. Data migration to minimize the average completion time. In *Proc. 14th SIAM Symp. on Disc. Algs. (SODA)*, pages 97–98, 2003.

[59] A. Kovács. Sum-multicoloring on paths. In *Proc. 21st Ann. Symp. on Theo. Aspects of C. S. (STACS)*, volume 2996 of *LNCS*, pages 68–80. Springer, 2004.

[60] A. Kovács. Fast monotone 3-approximation algorithm for scheduling related machines. In *Proc. 13th Europ. Symp. on Algs. (ESA)*, volume 3669 of *LNCS*, pages 616–627. Springer, 2005.

[61] A. Kovács. Polynomial time preemptive sum-multicoloring on paths. In *Proc. 32nd Intl. Col. (ICALP)*, volume 3580 of *LNCS*, pages 840–852. Springer, 2005.

[62] A. Kovács. Tighter approximation bounds for LPT scheduling in two special cases. In *Proc. 12th Italian Conf. on Algs. and Complexity (CIAC)*, volume 3998 of *LNCS*, pages 187–198. Springer, 2006. Invited to the CIAC06 Special Issue of the Journal of Discrete Algorithms.

[63] L.G. Kroon, A. Sen, H. Deng, and A. Roy. The optimum cost chromatic partition problem for trees and interval graphs. In *Proc. Wshop. on Graph-theor. Concepts in Comp. Sci.*, volume 1197 of *LNCS*, pages 279–292, 1996.

[64] M. Kubale. Preemptive versus nonpreemptive scheduling of biprocessor tasks on dedicated processors. *European J. of Operational Res.*, 94(2):242–251, 1996.

[65] E. Kubicka. *The chromatic sum of a graph.* PhD thesis, Western Michigan University, 1989.

[66] E. Kubicka, G. Kubicki, and D. Kountanis. Approximation algorithms for the chromatic sum. In *Proc. of the First Great Lakes Comp. Sci. Conf.*, pages 15–21. Springer, 1989.

[67] E. Kubicka and A.J. Schwenk. An introduction to chromatic sums. In *Proc. of the ACM Comp. Sci. Conf.*, pages 15–21. Springer, 1989.

[68] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. Sequencing and scheduling: algorithms and complexity. In S.C. Graves et al., editor, *Logistics of production and inventory/ Handbooks in operations research and management science*, page 445. North-Holland, 1993.

[69] R. Li and L. Shi. An on-line algorithm for some uniform processor scheduling. *SIAM Journal on Computing*, 27(2):414–422, 1998.

[70] J.W.S. Liu and C.L. Liu. Bounds on scheduling algorithms for heterogeneous computing systems. In *Proc. Intern. Federation of Information Processing Societies*, pages 349–353, 1974.

[71] N.A. Lynch. Upper bounds for static resource allocation in a distributed system. *J. Comput. System Sci.*, 23:254–278, 1981.

[72] D. Marx. Chromatic sum and minimum sum multicoloring. List of papers: `http://www.cs.bme.hu/~dmarx/sum.html`.

[73] D. Marx. The complexity of tree multicolorings. In *Proc. 27th Intl. Symp. Math. Found. Comput. Sci. (MFCS)*, volume 2420 of *LNCS*, pages 532–542. Springer, 2002.

[74] D. Marx. Minimum sum multicoloring on the edges of trees. In *Proc. 1st Wshop. Approx. and Online Alg. (WAOA)*, volume 2909 of *LNCS*, pages 214–226. Springer, 2003.

[75] D. Marx. *Graph Coloring with Local and Global Constraints*. PhD thesis, Budapest University of Technology and Economics, 2004.

[76] D. Marx. Minimum sum multicoloring on the edges of planar graphs and partial k-trees. In *Proc. 2nd Wshop. Approx. and Online Alg. (WAOA)*, volume 3351 of *LNCS*, pages 9–22. Springer, 2004.

[77] D. Marx. The complexity of chromatic strength and chromatic edge strength. *Computational Complexity*, 14(4):308–340, 2006.

[78] P. Mireault, J.B. Orlin, and R.V. Vohra. A parametric worst case analysis of the LPT heuristic for two uniform machines. *Oper. Res.*, 45(1):116–125, 1997.

[79] R. Myerson. Optimal auction design. *Mathematics of Operations Research*, 6:58–73, 1981.

[80] S. Nicoloso, M. Sarrafzadeh, and X. Song. On the sum coloring problem on interval graphs. *Algorithmica*, 23(2):109–126, 1999.

[81] N. Nisan. Algorithms for selfish agents - mechanism design for distributed computation. In *Proc. 18th Symp. on Theor. Asp. of Comp. Sci. (STACS)*, volume 1563 of *LNCS*, pages 1–15. Springer, 1999.

[82] N. Nisan and A. Ronen. Algorithmic mechanism design. In *Proc. 31st Ann. ACM Symp. on Theory of Computing (STOC)*, pages 129–140, 1999.

[83] N. Nisan and A. Ronen. Computationally feasible VCG mechanisms. In *ACM Conf. on Electronic Commerce*, pages 242–252, 2000.

[84] C.H. Papadimitriou. Algorithms, games and the internet. In *Proc. 33rd Ann. ACM Symp. on Theory of Computing (STOC)*, pages 749–753, 2001.

[85] R.G. Parker. *Deterministic scheduling theory*. Chapman and Hall, London, 1995.

[86] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, New Jersey, 1995.

[87] M.R. Salvatipour. On sum coloring of graphs. *Discrete Applied Mathematics*, 127(3):477–488, 2003.

[88] K.J. Supowit. Finding a maximum planar subset of nets in a chanel. *IEEE Trans. Computer Aided Design*, 6(1):93–94, 1987.

[89] W. Vickrey. Counterspeculation, auctions and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

Summary

In the thesis we consider open problems from two distinct subareas of scheduling theory.

The first part provides an efficient algorithm for the preemptive sum multicoloring problem on path and cycle conflict graphs.

The input of a sum multicoloring, or SMC problem consists of a simple undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and integral weights $x(v)$ on each node $v \in \mathcal{V}$. The nodes represent jobs, each having size or time demand $x(v)$. The edges stand for pairwise conflicts between certain jobs, meaning that they cannot be processed at the same time.

The output is a proper multicoloring, i.e., one assigning a set $\Phi(v)$ of positive integers (colors) to each $v \in \mathcal{V}$ such that $|\Phi(v)| = x(v)$ and the sets assigned to adjacent vertices do not intersect. Thus, Φ determines a proper schedule of the jobs, where conflicting jobs never receive the same time-unit. Our objective is to minimize the average finish time of the jobs, or equivalently, the sum of finish times $\sum_{v \in \mathcal{V}} f(v)$, where $f(v) = \max \Phi(v)$.

In non-preemptive SMC (npSMC) the assigned sets $\Phi(v)$ must be contiguous, whereas in preemptive SMC (pSMC) the $\Phi(v)$ are arbitrary sets.

Although one of the classic motivations for studying sum (multi)coloring has been a problem in VLSI design, nowadays SMC finds its primary applications in scheduling settings, where the jobs need exclusive access to dedicated resources or machines, and they have to compete with other jobs for the use of these resources. Such scenarios occur, e.g., in dedicated multiprocessor scheduling, session management in local area networks, or file access control.

The SMC problem was introduced and studied on many different conflict graphs by Bar-Noy et al. [11], in 1999. Since even sum coloring (SC), the special case with unit time demands, was shown to be NP-hard on several graph classes [10, 13, 41, 49], recent research has mainly focused on the approximability and hardness of SC, pSMC, and npSMC [38, 40, 49, 52, 58, 74, 80].

On the other hand, it is natural to search for types of graphs, for which exact polynomial algorithms exist. Halldórsson et al. [50] studied the sum multicoloring problem on trees. For non-preemptive SMC they provided two efficient algorithms, which run in $\mathcal{O}(n^2)$ and $\mathcal{O}(np)$ time, respectively, where n is the number of vertices $|\mathcal{V}|$ and $p = \max_v x(v)$ is the maximum demand. For preemptive SMC, they gave a PTAS. The hardness of pSMC on trees and paths was posed as an open question already in an earlier version of the paper in 1999 [51]. One answer was given by Marx [73], who proved that even on binary trees pSMC is strongly NP-hard.

As another answer, in Chapter 2 of the thesis we provide the first exact polynomial algorithm on paths and cycles, with running time $\mathcal{O}(\min(n^2, n \log p))$. Thus, we identify a major gap between the solvability of the problem on binary trees and on graphs of maximum degree 2.

Although the investigated graphs are of extremely simple structure, even on these simple graph classes the problem has proved to be far from trivial. Our main result concerns the structure of optimal solutions, and is of combinatorial flavour. This result enables the algorithm to use a dynamic programming strategy for finding a solution in polyomial time.

The second part of the thesis presents a simple monotone 2.8-approximation algorithm for the classic scheduling problem $Q||C_{\max}$. Besides, it provides the exact approximation bound of the LPT algorithm for a special type of speed vector.

$Q||C_{\max}$ denotes the offline task scheduling problem on related machines. An instance of this problem is given by a machine speed vector $\langle s_1, \ldots, s_m \rangle$, and a job vector $\langle t_1, \ldots, t_n \rangle$, where t_j is the size of the jth job. The goal is to allocate the jobs to the machines, so that the overall finish time is minimized. In particular, the work w_i of machine i stands for the total job size assigned to i, the finish time of i is $f_i = w_i/s_i$, and the makespan to be minimized is $\max_{i=1}^{m} f_i$.

This is an NP-hard problem, but it has a PTAS [53]. The first approximation algorithm considered for $Q||C_{\max}$ was the 'Largest Processing Time first (LPT)' heuristic, which picks the jobs one by one in decreasing order of size, and always assigns the next job to a machine where it will have the smallest completion time [42].

In Chapter 3 of the thesis we are concerned with finding a so called monotone algorithm for $Q||C_{\max}$. A scheduling algorithm is monotone, if increasing the speed of any particular machine does not decrease the work assigned to that machine.

The motivation for the research on monotone algorithms originates in mechanism design. Traditional mechanism design focuses on voting and auction type problems in economics, and considers situations when players or agents might try to manipulate the system and lie in order to maximize their own profit. Mechanisms that are able to make payments to the players, so that a rational player will never find it in her self-interest to lie, are called truthful. The emergence of the Internet as a primary platform for distributed computation, made it necessary to consider similar socio-economic issues in theoretical computer science (cf. [5, 81, 84]). Nisan and Ronen [82] were the first to apply the standard tools of mechanism design to classic optimization problems like shortest paths, MST, and scheduling on unrelated machines.

Archer and Tardos [5] studied scheduling on related machines (as example for a more general setting), in a model where machines are owned by selfish agents, and the speed of each machine is private information to its agent. Payments should motivate, that the agents declare the true machine speeds to the scheduling mechanism. The authors showed that such payments exist only if the mechanism uses a monotone scheduling algorithm. They provided a randomized 3-approximation algorithm (later improved to 2-approximation [4]), and a mechanism that is truthful in expectation.

Auletta et al. [8] gave the first deterministic monotone – $(4 + \epsilon)$-approximation

– algorithm, which is polynomial if the number of machines m is constant. They used a modified version of Lpt, but conjectured that Lpt itself is monotone, if machine speeds are all integral powers of two (2-divisible). As the best previous results, Andelman et al. [3] presented a monotone FPTAS for the case of constant m, and a 5-approximation algorithm for arbitrary m.

In the thesis we prove the conjecture of Auletta et al., that Lpt is monotone on 2-divisible machines. By rounding arbitrary input speeds to powers of two, we obtain a simple deterministic monotone 2.8-approximation algorithm for $Q||C_{\max}$, polynomial in n and m. This is an improvement over the 5-approximation bound of Andelman et al. [3]. As compared to the algorithms of Archer and Tardos [4, 5], no randomization is needed, and a stronger definition of truthfulness is fulfilled. Like the above papers [3, 5, 8], we also describe advantageous characteristics of the payment function that complements our algorithm to a truthful mechanism.

Besides proving the monotonicity, it is nontrivial to prove the approximation bound (of 1.4) of Lpt on 2-divisible speeds. As an additional result, we obtain that on speed vectors of the form $\langle 1, 1, \ldots, 1, s > 1\rangle$, the worst case ratio of Lpt is $\frac{\sqrt{3}+1}{2} \approx 1.3660$. The best previous lower and upper bounds on this ratio were $\frac{4}{3} - \epsilon < Lpt/Opt \leq \frac{3}{2} - \frac{1}{2m}$. The latter bounds were found by Gonzalez, Ibarra, and Sahni in 1977 [42], who conjectured the actual worst case ratio to be 4/3.

Thus, the results in the thesis provide answers to three open questions or conjectures in scheduling theory. Our results appeared previously in [59, 60, 61, 62].

Zusammenfassung

In der vorliegenden Arbeit befassen wir uns mit Problemen aus zwei verschiedenen Teilgebieten der Scheduling-Theorie.

Im ersten Teil entwickeln wir einen effizienten Algorithmus für das sog. *preemptive Sum Multicoloring* Problem auf Pfaden und Kreisen als Konfliktgraphen.

Eine Eingabe für das Sum Multicoloring (oder kurz SMC) Problem besteht aus einem ungerichteten Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, und ganzzahligen, positiven Gewichten $x(v)$ auf allen Knoten $v \in \mathcal{V}$. Ein Knoten v repräsentiert einen Job mit Größe bzw. Zeitbedarf $x(v)$. Die Kanten stehen für paarweise Konflikte zwischen bestimmten Jobs in dem Sinne, daß sie nicht gleichzeitig verarbeitet werden können.

In einer korrekten Ausgabe für das Multicoloring Problem ist jedem Knoten $v \in \mathcal{V}$ eine Menge $\Phi(v)$ positiver ganzer Zahlen (Farben) so zugeordnet, daß $|\Phi(v)| = x(v)$ und sich die Mengen für benachbarte Graphknoten nicht überschneiden. Damit bestimmt Φ eine gültige Ausführungsreihenfolge der Jobs wobei in Konflikt stehenden Jobs nie die gleiche Zeiteinheit zugeteilt wird. Unsere Aufgabe besteht nun darin, die durchschnittliche Fertigstellungszeit der Jobs, oder – äquivalent – die Summe der Fertigstellungszeiten $\sum_{v \in \mathcal{V}} f(v)$ zu minimieren, wobei $f(v) = \max \Phi(v)$.

Im *non-preemptive SMC (npSMC)* Problem besteht jede mehrelementige Menge $\Phi(v)$ aus fortlaufenden ganzen Zahlen, wohingegen im *preemptive SMC (pSMC)* die Mengen $\Phi(v)$ Lücken aufweisen dürfen.

Die Untersuchung von Sum (Multi)coloring wurde ursprünglich über ein Problem im VLSI Entwurf motiviert. Weiterhin kann SMC vor allem in Scheduling-Szenarien angewandt werden, bei denen die Jobs exklusiven Zugriff auf bestimmte Resourcen oder Maschinen benötigen, und mit anderen Jobs um den Gebrauch dieser Resourcen konkurrieren. Solche Szenarien ergeben sich z.B. in den Bereichen *Dedicated Multiprocessor Scheduling*, *Session Management* in lokalen Netzwerken oder der Zugriffskontrolle auf Dateien.

Das SMC Problem wurde von Bar-Noy et al. in 1999 formuliert und auf verschiedenen Konfliktgraphen studiert [11]. Da sich sogar *Sum Coloring (SC)* – der Spezialfall mit einheitlichen Zeitanforderungen – auf mehreren Graphklassen als NP-schwer herausstellte [10, 13, 41, 49], hat sich die aktuellere Forschung vor allem auf die Approximierbarkeit und NP-Schwere von SC, pSMC und npSMC konzentriert [38, 40, 49, 52, 58, 74, 80].

Andererseits ist es natürlich nach Graphklassen zu suchen, für die sich das Problem in polynomieller Zeit exakt lösen läßt. Halldórsson et al. [50] untersuchten Sum Multicoloring auf Bäumen. Für non-preemptive SMC fanden sie zwei Algo-

rithmen, die in $\mathcal{O}(n^2)$ bzw. $\mathcal{O}(np)$ Zeit laufen, wobei n die Anzahl der Jobs $|\mathcal{V}|$ und $p = \max_v x(v)$ die maximale Zeitanforderung darstellen. Für preemptive SMC beschrieben sie ein PTAS. Die Frage nach der Komplexitätsklasse von pSMC auf Bäumen und Pfaden wurde schon in einer früheren Version des Artikels als offenes Problem gestellt [51].

Eine erste Antwort präsentierte Marx [73], indem er zeigte, daß pSMC sogar auf binären Bäumen NP-schwer ist. Wir liefern in Kapitel 2 einen weiteren Beitrag, in Form des ersten exakten Algorithmus für pSMC auf Pfaden und Kreisen mit polynomieller Laufzeit; er benötigt $\mathcal{O}(\min(n^2, n \log p))$ Zeit. Damit identifizieren wir einen bedeutenden Sprung zwischen der effizienten Lösbarkeit des Problems auf binären Bäumen einerseits und Graphen mit maximalem Grad zwei andererseits.

Obwohl die untersuchten Graphen von sehr einfacher Struktur sind, hat sich das pSMC Problem für sie schon als äußerst nichttrivial herausgestellt. Unser Hauptergebnis betrifft die Struktur optimaler Lösungen und ist kombinatorischer Natur. Dieses Resultat erlaubt den Einsatz dynamischer Programmierung um algorithmisch eine Lösung in polynomieller Zeit zu gewinnen.

Im zweiten Teil dieser Arbeit präsentieren wir einen einfachen monotonen Algorithmus mit Approximationsgüte 2.8 für das klassische Schedulingproblem $Q||C_{\max}$. Als Nebenprodukt ergibt sich die genaue Approximationsgüte des LPT Algorithmus für einen bestimmten Typ von Geschwindigkeitsvektoren.

$Q||C_{\max}$ bezeichnet das (off-line) Schedulingproblem mit abhängigen Maschinen (*uniform* or *related machines*). Eine Probleminstanz besteht aus einem Geschwindigkeitsvektor $\langle s_1, \ldots, s_m \rangle$ für die Maschinen und einem Jobvektor $\langle t_1, \ldots, t_n \rangle$, wobei t_j die Größe des j-ten Jobs darstellt. Die Aufgabe besteht darin, die Jobs so den Maschinen zuzuweisen, daß die maximale Maschinenlaufzeit minimiert wird. Das ist ein NP-schweres Problem, aber ein PTAS existiert [53].

Der erste diesbezügliche Approximationsalgorithmus war die *Largest Processing Time first* (LPT) Heuristik, welche die nach umgekehrter Größe sortierten Jobs so verteilt, daß der nächste zu platzierende Job auf derjenigen Maschine zu liegen kommt, die mit ihm den frühesten Fertigstellungszeitpunkt erzielt [42].

In Kapitel 3 beschäftigen wir uns mit sog. monotonen Algorithmen für $Q||C_{\max}$. Ein Scheduling-Algorithmus heißt monoton, wenn die Steigerung der Geschwindigkeit einer beliebigen Maschine nicht dazu führt, daß dieser Maschine weniger Arbeit zugewiesen wird.

Die Motivation zur Erforschung von monotonen Algorithmen entsprang dem *Mechanism Design*. Traditionelles Mechanism Design konzentriert sich auf Abstimmungs- und Auktionsprobleme im ökonomischen Umfeld und behandelt Situationen, in denen Spieler oder Agenten geneigt sind das System durch Lügen dahingehend zu manipulieren, daß sie ihren eigenen Gewinn maximieren. *Truthful* Mechanismen heißen jene, die durch ihre Auszahlungsfunktionen rationale Spieler dazu bringen, in ihrem eigenen Interesse nicht zu lügen. Die Verbreitung des Internets als Hauptplattform verteilter Berechnungen hat es nötig gemacht, ähnliche sozio-ökonomische Aspekte auch im Rahmen theoretischer Informatik zu betrachten (siehe z.B. [5, 81, 84]). Nisan und Ronen [82] waren die ersten, die die Standardwerkzeuge des Mecha-

nism Designs auf klassische Optimierungsprobleme wie kürzeste Wege, minimale Spannbäume und Schedulingprobleme auf unabhängigen Maschinen (*unrelated machines*) anwendeten.

Archer und Tardos [5] untersuchten Scheduling auf abhängigen Maschinen (als Beispiel eines allgemeineren Ansatzes), und zwar in einem Modell bei dem Maschinen eigennützigen Agenten gehören und die wahren Geschwindigkeiten der Maschinen nur den jeweiligen Besitzern bekannt sind. Die Auszahlungsfunktion sollte so beschaffen sein, daß die Agenten einen Anreiz haben, die tatsächlichen Maschinengeschwindigkeiten offenzulegen. Die Autoren zeigten, daß solche Auszahlungsfunktion nur dann existiert, wenn der Mechanismus einen monotonen Scheduling-Algorithmus benutzt. Sie entwickelten einen randomisierten Algorithmus mit Approximationsfaktor drei (später verbessert auf zwei [4]) sowie einen Mechanismus, der nach Erwartung truthful ist.

Auletta et al. entwarfen den ersten deterministischen monotonen – $(4+\epsilon)$-approximativen – Algorithmus, der in Polynomialzeit läuft falls die Anzahl der Maschinen, m, konstant ist [8]. Sie benutzten eine Variante von LPT, aber äußerten die Vermutung, daß LPT selbst monoton sei, vorausgesetzt daß alle Maschinengeschwindigkeiten ganzzahlige Potenzen von zwei sind (2-*divisible machines*).

Als bestes bisheriges Resultat veröffentlichten Andelman et al. [3] ein monotones FPTAS für konstantes m sowie einen Algorithmus mit Approximationsgüte fünf für beliebiges m.

In unserer Arbeit beweisen wir die Vermutung von Auletta et al., daß LPT im Falle von 2-divisible Maschinen monoton ist. Indem man für eine beliebige Eingabe die Maschinengeschwindigkeiten auf das jeweilige nächste Vielfache von zwei rundet, erhält man einen einfachen deterministischen monotonen 2.8-approximativen Algorithmus für $Q||C_{\max}$, der polynomielle Laufzeit in n und m aufweist. Dies ist eine deutliche Verbesserung gegenüber der Approximationsgüte von fünf im Algorithmus von Andelman et al. [3]. Verglichen mit dem Algorithmus von Archer und Tardos [4, 5] wird keine Randomisierung benötigt; weiterhin wird bei unserem Ansatz eine stärkere Definition von *truthful* Mechanismen unterstützt. Wie in den obigen Artikeln [3, 5, 8] beschreiben wir auch günstige Eigenschaften der Zahlungsfunktion, die gemeinsam mit unserem Algorithmus einen truthful Mechanismus darstellt.

Neben dem Beweis der Monotonie liefern wir einen nicht-trivialen Beweis für die Approximationsgüte (1.4) von LPT im Falle von 2-divisible Maschinen. Weiterhin zeigen wir, daß die Approximationsgüte von LPT auf Geschwindigkeitsvektoren der Form $\langle 1, 1, \ldots, 1, s > 1\rangle$ im schlimmsten Fall $\frac{\sqrt{3}+1}{2} \approx 1.3660$ ist. Die bisherigen besten unteren und oberen Schranken waren $\frac{4}{3} - \epsilon < Lpt/Opt \leq \frac{3}{2} - \frac{1}{2m}$. Letztere wurden 1977 von Gonzalez, Ibara und Sahni [42] bewiesen, die mutmaßten, daß die tatsächliche obere Schranke bei 4/3 läge.

Alles in allem, liefert diese Arbeit Antworten auf drei offene Fragen bzw. Vermutungen im Bereich der Scheduling-Theorie. Unsere Resultate erschienen in früheren Versionen unter [59, 60, 61, 62].

Curriculum Vitae

Name: Annamária Kovács

Birth: 1970 Budapest, Hungary

Studies:

- 1988 – 1990 Civil Engineering Faculty of the Technical University of Budapest
- 1990 – 1995 Mathematics Faculty of Eötvös Loránd University of Sciences (ELTE), Budapest;
- 1995 masters degree as mathematician
- 1996 masters degree as teacher of mathematics

Employment:

- 09/1995 – 07/2001 Computer and Automation Research Institute of the Hungarian Academy of Sciences
  - Oracle database web-enabled interface development; IBM Java development and support; Lotus Notes system administration
  - gave basic courses on computer usage; & on Java programming
- 1997 – 1999 teaching undergraduate courses in mathematics at the Budapest University of Economics (lectures and tutorials).
- 1998 teaching introductory graph theory and algorithms at the Electric Engineering Faculty at the Technical University of Budapest.

Ph.D. studies:

- 09/1999 entered the Informatics PhD Programme of ELTE Budapest

  Advisors: Prof. Dr. Katalin Friedl and Prof. Dr. Lajos Rónyai
- 09/2001 – 12/2001 PhD student with the Marie Curie Fellowship Programme at the Max Planck Institute for Informatics (MPI) Saarbrücken, Germany
- 12/2001 – 10/2003 maternity leave

- 10/2003 – PhD student at MPI, partially with support of the EU Marie Curie Fellowship Programme

  Advisor: Prof. Dr. Kurt Mehlhorn

Publications:

A. Kovács. Sum-multicoloring on paths. In *Proc. 21st Ann. Symp. on Theoretical Aspects of Computer Science (STACS)*, LNCS 2996, pp. 68–80, Springer, 2004. Springer.

A. Kovács. Polynomial time preemptive sum-multicoloring on paths. In *Proc. 32nd Intl. Colloquium on Automata, Languages and Programming (ICALP)*, LNCS 3580, pp. 840–852, Springer, 2005.

A. Kovács. Fast monotone 3-approximation algorithm for scheduling related machines. In *Proc. 13th European Symposium on Algorithms (ESA)*, LNCS 3669, pp. 616–627, Springer, 2005.

A. Kovács. Tighter approximation bounds for LPT scheduling in two special cases. In *Proc. 12th Italian Conference on Algorithms and Complexity (CIAC)*, LNCS 3998, pp. 187–198, Springer, 2006. Invited to the CIAC06 Special Issue of the Journal of Discrete Algorithms.