APPROXIMATION ALGORITHMS FOR SCHEDULING AND TWO-DIMENSIONAL PACKING PROBLEMS

Dissertation
zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)
der Technischen Fakultät der Christian-Albrechts-Universität zu Kiel,

vorgelegt von Dipl.-Inf. Ulrich Michael Schwarz

Kiel

2010

1. Gutachter: Prof. Dr. Klaus Jansen
2. Gutachter: Priv.-Doz. Dr. habil. Rob van Stee
Datum der Disputation: 1. Juli 2010
zum Druck genehmigt: 1. Juli 2010

Acknowledgements

Beyond the name on the title page, many need to mentioned which have influenced, explicitly or implicitly, this work. First and foremost, my thanks go to my colleagues, who shared the ups and downs:
Dr. Florian Diedrich, Lars Prädel, Dr. Ralf Thöle and Christina Otte;
to my advisor Prof. Dr. Klaus Jansen;
to my further co-authors Gabrielle Muratore, Fanny Pascual, Denis Trystram, and Gerhard Woeginger;
the students who have worked with me in teaching and research, particularly:
Rachid El-Araari, Tim Hartnack and Nick Prühs;
the colleagues in EU project AEOLUS, especially Ulf-Peter, who somehow made sure everything got done in time;
the reviewers at conferences and journals: anonymous, but not forgotten;
the students who have sat through my tutorials and the occasional lecture;
and finally, Eva, Pamela, Andreas, Jamy, Paul and Neska, for support that means more to me than words could express.

Kiel, March 2010.

Abstract

This dissertation thesis is concerned with two topics of combinatorial optimization: scheduling and geometrical packing problems.

Scheduling deals with the assignment of jobs to machines in a 'good' way, for suitable notions of good. Two particular problems are studied in depth: on the one hand, we consider the impact of machine failure on online scheduling, i.e. what are the consequences of the fact that in real life, machines do not work flawlessly around the clock, but need maintenance intervals or can break down? How do we need to adapt our algorithms to still obtain good overall schedules, and in what settings do we even have a chance to succeed?

Our second problem is of a more static nature: in some settings, not every job is permitted on all the machines. A classical example would be that of workers which needs special qualification to execute some jobs, or a certain minimum requirement of memory size of computers, etc. The problem in general is notoriously hard to tackle; we present improved approximation ratios for several special cases. In particular, we derive a polynomial-time approximation scheme for nested interval restrictions, which occur naturally in many practical applications.

Our final topic is two-dimensional geometric bin packing, the problem of packing rectangular objects into the minimum number of containers of identical size. (Figuratively speaking, we are arranging advertisements of fixed dimensions onto the minimum number of print pages.) It is known that no approximation ratio better than 2 is possible for this problem, unless $P = NP$; we present an algorithm that guarantees this ratio.

Zusammenfassung

Diese Promotionsschrift behandelt zwei Arten kombinatorischer Optimierungsprobleme: Ablaufplanungsprobleme und geometrische Packungsprobleme.

Ablaufplanungsprobleme handeln davon, eine Menge von Aufgaben, die Jobs, auf eine Menge von ausführenden Maschinen oder Arbeitern zu verteilen, so dass der entstehende Ablaufplan in geeignetem Sinne „gut“ ist. Wir betrachten hier insbesondere folgende zwei Probleme der Ablaufplanung: einerseits untersuchen wir den Einfluß von Maschinenausfällen auf die Online-Ablaufplanung: im wirklichen Leben sind Maschinen nicht fehler- und unterbrechungslos verfügbar. Wir geben eine teilweise Antwort auf die Frage, mit welchen Änderungen Algorithmen trotz unerwartet auftretender Maschinenausfälle gute Pläne erstellen können, und in welchen Fällen es prinzipiell nicht möglich ist, gute Ablaufpläne zu erstellen.

Unser zweites Ablaufplanungsproblem ist von statischerer Natur: in der praktischen Anwendung ist es häufig der Fall, dass nicht jede Maschine jeden Job ausführen kann. Ein einfaches Beispiel sind menschliche Arbeiter, die gewisse formale Qualifikationen für gewisse Jobs haben müssen. Diese Problem erweist sich als in voller Allgemeinheit bekannt hartnäckig; wir stellen hier Algorithmen für einige Spezialfälle vor. Insbesondere präsentieren wir ein polynomielles Approximationsschema für den wichtigen Fall verschachtelter Restriktionen, der in der Mitarbeiterplanung auf natürliche Weise auftritt.

Schlussendlich untersuchen wir das zweidimensionale geometrische *bin packing*-Problem. Fragestellung dieses Problem ist es, rechteckige Objekte in die minimale Anzahl von Containern gleicher Größe zu packen. Salopp gesprochen versuchen wir, eine vorgegebene Menge von Anzeigen mit vorgegebenen Abmessungen auf eine möglichst kleine Zahl von Druckseiten gleicher Größe zu platzieren. Es ist bekannt, dass dieses Problem keine Algorithmus mit Approximationsgüte besser als 2 erlaubt, es sei denn, $P = NP$; wir stellen einen Algorithmus mit Güte 2 vor.

Contents

1 Introduction

The present work contains some results of my research at the University of Kiel from 2006 to 2010. It can be divided into three major parts, which share a common theme of packing. The first two parts deal with on- and offline *scheduling*, the packing of jobs onto machines to obtain a 'good' (e.g., fast) execution. The third part deals with a two-dimensional rectangle packing problem. Before we go into the details, we recollect elementary facts about algorithms and approximability that we will use frequently throughout.

1.1 Computability, complexity and approximation algorithms

In this section, we will briefly recall the motivation and definitions needed to understand the analysis of algorithms in terms of complexity and approximation properties. Of course, this is not exhaustive by far; more detailed expositions can be found in the standard literature on algorithm design [AHU74, CLRS90], complexity theory [GJ79] and approximation theory [Vaz01, JM08].

1.1.1 Complexity

To formalize the notion of efficient algorithms that approximately solve problems, we need to define the constituent concepts. One basic distinction is that of *decision*

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[CLRS90] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 1990.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[Vaz01] V. V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., 2001.

[JM08] K. Jansen and M. Margraf. *Approximative Algorithmen und Nichtapproximierbarkeit*. de Gruyter, 2008.

problems and *optimization problems*: in both cases, a problem is a collection of *instances*, and each instance has a set of *solutions*. The decision problem is to find out (algorithmically) for an input instance I if its set of solutions $\mathrm{SOL}(I)$ is empty. For a minimization problem, we are additionally given a valuation function VAL that measures the cost of a solution, and the objective is to find a solution s that minimizes $\mathrm{VAL}(s)$. (A maximization problem considers VAL a profit and wants to maximize $\mathrm{VAL}(s)$.) In practice, our optimization problems will be such that $\mathrm{SOL}(I)$ is never empty here and $\mathrm{VAL}(s) \in]0, \infty[$ for all $s \in \mathrm{SOL}(I)$.

We denote with $\mathrm{OPT}(I)$, or OPT if I is clear from context, one fixed solution that attains the minimum for instance I; where no confusion can occur, we also call its value $\mathrm{OPT}(I)$. In the problems we study, it will be the case that $\mathrm{OPT}(I)$ is pseudopolynomially bounded in I.

An algorithm is a formalized procedure to derive an output result from some input. Classically, this is formalized in terms of an abstract machine model, such as the MMIX assembly language [Knu05] or Turing machines [Tur37]. We very briefly recall the definition of the latter, since Turing machines are essential to the definition of the complexity classes P and NP.

Definition 1.1.1. A nondeterministic *Turing machine* (NTM) consists of a finite control which can read and write to an infinite tape. Formally, it is defined by a tuple

$$\mathfrak{T} := (Q, q_0, \Delta, \Sigma, \Gamma, F) \tag{1.1}$$

consisting of the finite set of states Q, among which are the initial state q_0 and the set of accepting states $F \subseteq Q$, along with a finite input alphabet Σ, which we take to be $\{0, 1\}$ without loss of generality, a finite tape alphabet $\Gamma \supsetneq \Sigma$ including the *blank symbol* $\flat \notin \Sigma$, and a transition relation

$$\Delta \subseteq (Q \times \Gamma) \times (\Gamma \times \{-1, 0, +1\} \times Q), \tag{1.2}$$

[Knu05] D. E. Knuth. *MMIX - A RISC Computer for the New Millennium*, volume 1 of *The Art Of Computer Programming*. Addison-Wesley Longman, 2005.

[Tur37] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 1937.

where an element (q, a, a', m, q') denotes that upon being in state q and reading a on the tape, $\mathfrak{T}$ will replace a with a' in the current cell, move the tape head to the left, right or not at all for $m = -1, +1, 0$, respectively, and will then go into state q'.

If Δ is a function $\delta : (Q \times \Gamma) \to (\Gamma \times \{-1, 0, +1\} \times Q)$, we call $\mathfrak{T}$ *deterministic* (DTM).

We say that $\mathfrak{T}$ *accepts* a word $w \in \Sigma^*$ *in time* f for some function $f : \mathbb{N} \to \mathbb{N}$ if there is a sequence of at most $f(|w|)$ transitions starting from q_0 with only w on the tape to an accepting state. We denote with $L_f(\mathfrak{T})$, the *accepted language* of $\mathfrak{T}$, the set of words that are accepted in time f.

Finally, we define

$$\mathrm{P} := \{L \subseteq \Sigma^* : \exists p \in \mathrm{poly}(n), \mathrm{DTM}\ \mathfrak{T} : L = L_p(\mathfrak{T})\}$$
$$\mathrm{NP} := \{L \subseteq \Sigma^* : \exists p \in \mathrm{poly}(n), \mathrm{NTM}\ \mathfrak{T} : L = L_p(\mathfrak{T})\}$$

It is easy to see by definition that $\mathrm{P} \subseteq \mathrm{NP}$. The converse, $\mathrm{NP} \subseteq \mathrm{P}$, is unknown, but generally assumed to be untrue. Proving or disproving it is beyond the scope of this work.

Given two languages $L_1, L_2 \subseteq \Sigma^*$, L_1 is *polynomial-time transformable* to L_2 if there is a homomorphism $f : \Sigma^* \to \Sigma^*$, i.e. $f(w) \in L_2 \iff w \in L_1$, that can be computed deterministically in polynomial time. Intuitively, this means that L_1 is at most as difficult to decide as L_2, since it can be seen as a special case, up to a polynomial-time re-writing step given by f.

There are problems in NP that are NP-complete, i.e. maximally hard in terms of this polynomial-time reducibility: a polynomial-time DTM for such a problem can be used to derive polynomial-time DTMs for *all* problems in NP. Most naturally occuring combinatorial problems appear to be either in P or NP-complete, the latter ones being more interesting with regard to efficiently finding *sub-optimal* solutions. Among the literally hundreds of problems known to be NP-complete, we define some which are of particular importance for the present work:

Definition 1.1.2 (Partition; SP12 in [GJ79])**.** Given a multiset $\{a_1, \ldots, a_n\} \in \mathbb{N}^n$, is there a subset $S \subseteq \{1, \ldots, n\}$ such that $\sum\{a_i : i \in S\} = \sum\{a_i : i \in \{1, \ldots, n\} \setminus S\}$?

Definition 1.1.3 (3-Partition; SP15 in [GJ79])**.** Given a container size $B \in \mathbb{N}$ and

a multiset $\{a_1, \ldots, a_{3n}\} \in \mathbb{N}^{3n}$ such that $a_i \in]B/4, B/2[$ and $\sum_{i=1}^{3n} a_i = nB$, is there a partition of $\{1, \ldots, 3n\}$ into sets $S_1, \ldots, S_n$ such that $|S_1| = \cdots = |S_n| = 3$ and $\sum\{a_i : i \in S_1\} = \cdots = \sum\{a_i : i \in S_n\} = B$?

Definition 1.1.4 (Relaxed 3-Partition)**.** Given a container size $B \in \mathbb{N}$ and a multiset $\{a_1, \ldots, a_{3n}\} \in \mathbb{N}^{3n}$ such that $\sum_{i=1}^{3n} a_i = nB$, is there a partition of $\{1, \ldots, 3n\}$ into sets $S_1, \ldots, S_n$ such that $|S_1| = \cdots = |S_n| = 3$ and $\sum\{a_i : i \in S_1\} = \cdots = \sum\{a_i : i \in S_n\} = B$?

Theorem 1.1.5. *Partition is* NP*-complete; 3-Partition and Relaxed 3-Partition are* NP*-complete in the strong sense, i.e. even if the numbers are encoded in the unary alphabet* $\Sigma = \{1\}$.

1.1.2 Approximation algorithms

At first sight, decision problems seem to bear little relation to optimization problems. However, minimization problems give us associated decision problems in a natural way if we ask 'Is there a solution of value at most k?' (And similarly, we may ask for a solution of value at least k for maximization problems.) Clearly, if we can find optimal solutions, we can solve all associated decision problems by comparing the target value with the optimal value. (Recall that we generally assume the encoding length $|\mathrm{OPT}(I)| \in \mathrm{poly}(|I|)$, so this comparison can be done in polynomial time.) Unfortunately, the associated decision problems for the settings we study will mostly turn out to be NP-complete, which means that unless P = NP, we cannot find optimal solutions deterministically in polynomial time.

In light of the disappointing absence of fast exact algorithms, we consider polynomial-time algorithms which are not exact. Given a minimization problem and a polynomial-time algorithm ALG for it, we call ALG a γ*-approximation* if

$$\max_I \frac{\mathrm{ALG}(I)}{\mathrm{OPT}(I)} \leq \gamma\,, \tag{1.3}$$

where $\mathrm{ALG}(I)$ denotes the value given by the algorithm and $\mathrm{OPT}(I)$ the optimal

value of instance I. Somewhat weaker, ALG is an *asymptotic* γ-approximation if

$$\limsup_{\mathrm{OPT}(I)\to\infty} \frac{\mathrm{ALG}(I)}{\mathrm{OPT}(I)} \leq \gamma\,. \tag{1.4}$$

A *polynomial-time approximation scheme*, PTAS for short, is a family of algorithms $\{\mathrm{ALG}_\epsilon : \epsilon > 0\}$ such that ALG_ϵ is a polynmomial-time $(1+\epsilon)$-approximation. Note particularly that ϵ is not part of the input of the algorithm, so the running time may depend superpolynomially on ϵ. Considerably better is an *efficient* PTAS (EPTAS), where ALG_ϵ has a running time bounded by $f(1/\epsilon)\cdot\mathrm{poly}(n)$ for some arbitrary function f. If f is also polynomial, the scheme is called a *fully-polynomial time* approximation scheme (FPTAS).

One trick we will commonly use to design approximation algorithms is to turn the relation of optimization problem and associated decision problem around:

Definition 1.1.6. Given a minimization problem, a γ-*relaxed* algorithm for the associated decision problem is an algorithm that for a pair (I, v) of instance I and guessed value v either correctly determines that no solution of value at most v exists or returns a solution of value at most $\gamma \cdot v$.

Using such a relaxed algorithm, we can often create a γ-approximation algorithm for the optimization problem:

Theorem 1.1.7. *If* $\mathrm{OPT}(I) \in \mathbb{N}$ *for all instances* I*, and* $\mathrm{OPT}(I) \leq p(I)$ *for some pseudopolynomial function* p*, and there is a polynomial-time* γ*-relaxed algorithm for the associated decision problem, then there is a polynomial-time* γ*-approximation for the optimization problem.*

Proof. Let D_γ the relaxed algorithm. Then, we can obtain the wanted approximation algorithm as shown in Algorithm 1.1. As to the correctness, the algorithm maintains that a solution of $\gamma \cdot u$ exists and is known, while a solution of ℓ does not exist. In each iteration, the length of the interval, $u - \ell$, halves, so that after $\lceil \log(p(I)) \rceil$ iterations, we know that no solution of value ℓ exists, so $\mathrm{OPT} > \ell$, but we have a solution of value $\gamma u = \gamma(\ell + 1) \leq \gamma\mathrm{OPT}$. □

If the objective value is not known to be integral, we can still achieve the following result:

Algorithm 1.1: Approximation by relaxed decision

```
Input: polynomial-time algorithm D_γ, bounding polynom p, instance I
ℓ := 0;
u := 2^⌈log p(I)⌉ ;          /* upper bound, rounded up to next power of 2 */
s := D_γ(u) ;                                          /* trial solution */
while u − ℓ > 1 do
    m := (ℓ + u)/2;
    if D_γ(m) successful then
        u := m;
        s := D_γ(m);
    else
        ℓ := m;
return s;
```

Corollary 1.1.8. If there is a pseudopolynomial function p such that $1 \leq \mathrm{OPT}(I) \leq p(I)$ for all instances I, and there is a polynomial-time γ-relaxed algorithm for the associated decision problem, then there is a polynomial-time $(1+\epsilon)\gamma$-approximation for the optimization problem for every constant $\epsilon > 0$.

Proof. We proceed as in Algorithm 1.1, however, we continue for additional $\lceil \log \epsilon^{-1} \rceil$ iterations. Then, it holds that $u - \ell \leq 2^{-\lceil \log \epsilon^{-1} \rceil} \leq \epsilon \leq \epsilon\mathrm{OPT}$. As before, we know there is no solution of value ℓ, and we have a solution of value $\gamma \cdot u \leq \gamma(\ell + \epsilon) \leq \gamma(\mathrm{OPT} + \epsilon\mathrm{OPT})$. □

1.1.3 Online algorithms

In Chapter 2, we will be studying *online* algorithms, i.e. algorithms that are not given the entire input at the beginning. Since we are mostly interested in online algorithms for scheduling problems, this manifests itself in two ways: jobs may arrive and machines may join and leave/fail while the algorithm is working and has already irrevocably assigned some jobs – entirely or partially – to machines. The usual way to think of this setting is to consider the online aspect to be dictated by an adversarial player that tries to sabotage the algorithm as much as possible.

The quality of algorithms in this setting is commonly measured by their *competit-*

ive ratio, which compares the result given by the online algorithm with the best possible *offline* result, i.e. one that could be achieved in retrospect, now knowing the decisions the adversary has made.

The flexible nature of the setting makes the algorithms slightly more difficult to formalize. For convenience, we write them in an event-oriented fashion, i.e. the algorithm is called whenever the outside situation changes. In addition, we allow the algorithm to generate a polynomial number of events itself, which will streamline the description. As to the running time, we will not give explicit numbers but mostly study the running time per event, and for setup calculations – after all, we must allow the algorithms at least constant time per event, and the number of events such as machine failures is outside our control.

1.2 Outline of this thesis

The remainder of this thesis is structured as follows: in Chapter 2, we consider scheduling problems in settings with online and semi-online machine failures. Some of the results of this chapter have been previously published in [DS07] and [Sch08].

In Chapter 3, scheduling on unrelated machines is studied. Particularly, we give improved approximation algorithms for three special cases of Scheduling with Interval Assignment Restrictions. One of the results given there was previously published in [MSW10].

In Chapter 4, we present a 2-approximation for the two-dimensional geometric bin packing problem. This result was previously published in shorter form in [JPS09].

[DS07] F. Diedrich and U. M. Schwarz. A framework for scheduling with online availability. In *Proc. Euro-Par*, 2007.

[Sch08] U. M. Schwarz. Online scheduling on semi-related machines. *Information Processing Letters*, September 2008.

[MSW10] G. Muratore, U. M. Schwarz, and G. J. Woeginger. Parallel machine scheduling with nested job assignment restrictions. *Operations Research Letters*, 2010.

[JPS09] K. Jansen, L. Prädel, and U. M. Schwarz. Two for one: Tight approximation of 2d bin packing. In *Proc. WADS*, 2009.

Finally, we conclude with open questions and future research directions.

2 Online Scheduling with Machine Unavailability

2.1 Introduction

In this chapter, we present scheduling algorithms for settings in which machine availability is not guaranteed. That this setting has become much more important in recent years reflects a change in the way large distributed computation is performed: initially, large-scale computing jobs would be executed on specialized parallel 'supercomputers' which operate in batch mode. It is a reasonably realistic assumption that such a computer always runs at its full power or is totally off-line, so that the amount of processing power available is constant. (Maintenance times would be inserted between the batch jobs; and a catastrophic failure during execution of a job is not catered for.)

The availability of reasonably fast global networks and fast cheap computers has changed the picture: it is now feasible to spread tasks among more than one site, and in particular, an organization that has to manage large-scale computation may want to invite private people to let their computers work on part of the problem. Well-known examples of this are the SETI@home project, which analyses extra-terrestrial radio signals, GIMPS, a project looking for large Mersenne prime numbers, and various projects such as the World Community Grid that perform simulations of chemical reactions for medical purposes. Such a structure requires new scheduling algorithms: computing power is no longer a commodity that is bought with, say, 99.999% availability, but donated because of the nature of the project. In such a setting, it has to be accepted that donations can be withdrawn with little or no advance warning, for example because the donating user needs the machine's processing power for their own local purposes. In particular, these changes are not under the scheduler's control.

2.1.1 Historical overview

In classical scheduling, dynamic machine unavailability has at best played a minor role; however, unreliable machines have been considered as far back as 1975 [Ull75] in the offline setting for the makespan objective. It is known from Eyraud-Duboid et al. [EDMT07] that without any further restriction and without preemptions, no constant approximation ratio is possible for the makespan objective on identical machines; we show a similar construction for the online setting with preemptions on related machines as Lemma 2.3.5 on p. 41. Similar constructions have been done by Fu et al. [FHZ09] for the weighted sum of completion times. Hence, it is common to assume that at least one machine is always available, and the offline results listed subsequently are all in this setting. The best results are given by Diedrich et al. [DJPT07] who give a PTAS if the number of machines m is considered constant; in [DJ09], Diedrich and Jansen give a $3/2$-approximation if the number of machines is not constant. Both results are tight: for m constant, the problem is strongly NP-hard by reduction of 3-Partition; for m part of the input, no algorithm with approximation ratio $3/2 - \epsilon$ exists unless $\mathrm{P} = \mathrm{NP}$ [DJ09].

Semi-online adversarial variants of the makespan problem were studied by Sanlaville [San95] as well as Albers and Schmidt [AS01]. In the semi-online setting, the next point in time when machine availability may change is known. The discrete time step setting also considered in the following is a special case (i.e.

[Ull75] J. D. Ullman. NP-complete scheduling problems. *Journal of Computer and System Sciences*, 1975.

[EDMT07] L. Eyraud-Dubois, G. Mounie, and D. Trystram. Analysis of scheduling algorithms with reservations. In *Proc. IPDPS*, 2007.

[FHZ09] B. Fu, Y. Huo, and H. Zhao. Exponential inapproximability and FPTAS for scheduling with availability constraints. *Theoretical Computer Science*, 2009.

[DJPT07] F. Diedrich, K. Jansen, F. Pascual, and D. Trystram. Approximation algorithms for scheduling with reservations. In *Proc. HiPC*, 2007.

[DJ09] F. Diedrich and K. Jansen. Improved approximation algorithms for scheduling with fixed jobs. In *Proc. SODA*, 2009.

[San95] E. Sanlaville. Nearly on line scheduling of preemptive independent tasks. *Discrete Applied Mathematics*, 1995.

[AS01] S. Albers and G. Schmidt. Scheduling with unexpected machine breakdowns. *Discrete Applied Mathematics*, 2001.

we assume that at time $t+1$, availability changes) that is closely linked to the unit execution time model. Sanlaville and Liu [LS95] have shown that *longest remaining processing time* (LRPT) is an optimal strategy for minimizing the makespan even if there are certain forms of precedence constraints on the jobs.

Albers and Schmidt [AS01] also give results on the true online setting which are obtained by imposing a "guessed" discretization of time.

The general notion of solving an online problem by re-using offline solutions, which is central to the algorithms we propose, was used by Hall et al. [HSW96] and earlier by Shmoys et al. [SWW95], where $\sum w_j C_j$ and makespan objectives, respectively, with online job arrivals were approximated using corresponding or related offline algorithms.

2.1.2 New results

In this chapter, we consider three settings that give different abilities to the adversary. We start by a very simple setting where preemptions and machine failures can only happen in discrete timesteps. We consider both makespan and average completion time objectives: for the average completion time objective, we give general inapproximability results and identify special cases in which the SRPT heuristic continues to be optimal. We then present a meta-heuristic MIMIC which can be used to transfer approximation results from offline settings to the online setting with failure. In particular, this heuristic can handle release times of jobs and any completion-time based objective, as long as an offline approximation algorithm is known. Under a probabilistic model of machine failure, we can then bound the quality of MIMIC in terms of machine reliability and quality of the underlying algorithm.

In the second setting, we strengthen the algorithm by allowing *look-ahead*, i.e. the algorithm is aware of the next change in machine availability, and arbitrary

---

[LS95] Z. Liu and E. Sanlaville. Preemptive scheduling with variable profile, precedence constraints and due dates. *Discrete Applied Mathematics*, 1995.

[HSW96] L. A. Hall, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line algorithms. In *Proc. SODA*, 1996.

[SWW95] D. B. Shmoys, J. Wein, and D. P. Williamson. Scheduling parallel machines on-line. *SIAM Journal on Computing*, 1995.

preemptions. We show that in this adapted setting, MIMIC can also handle precendence constraints without further degradation of approximation. However, these results – and the general approach taken by MIMIC – cannot be translated to related machines in a meaningful way. Nevertheless, we give an algorithm LAFM that is optimal for the problem of minimizing the makespan on related machines with failures, even when jobs have non-zero release times.

Our third setting features a stronger adversary than the second: we now do away with the possibility of look-ahead. We show that our algorithm LAFM can be modified to SALAFM which yields a solution of value OPT $+ \epsilon$ for arbitrary $\epsilon > 0$, as long as some machine is always available.

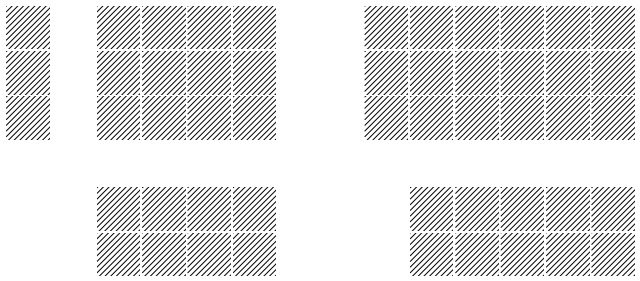
2.2 Scheduling in discrete time steps

We will consider the problem of scheduling n jobs $1, \ldots, n$, where a job i has processing time p_i which is known *a priori*. We denote the completion time of job i in schedule σ with $C_i(\sigma)$ and drop the schedule where it is clear from context. Since we are allowed to move jobs between machines anyway, it is sufficient to consider the number of machines present at any time, not the identity of the machines. We denote this number of machines available for a given time t as $m(t)$ and the total number of machines $m = \max_{t \in \mathbb{N}} m(t)$. The SRPT algorithm simply schedules the jobs of shortest remaining processing time in every step, preempting running jobs if necessary.

2.2.1 Lower bounds

Theorem 2.2.1. *There is* no *online algorithm for* P,*fail* $\mid p_j \in \{1,2\} \mid \sum_j C_j$ *with competitive ratio* $2 - \epsilon$ *for any* $\epsilon > 0$.

We consider instances of the following structure: for m even, there are $k = m/2$ jobs of length 1 and k jobs of length 2. During the first time step, $m/2$ machines are available. The adversary will then choose from two different machine availability patterns A, B, depending on the number j of short jobs executed during the first step.



(a) Pattern A (b) Pattern B

Figure 2.1: The two patterns of the adversary

Let us first describe these patterns, which are also sketched in Figure 2.1. In pattern A, all m machines are available during the second time step, and only one machine is available from the third step on. In pattern B, only one machine is available from the second step onwards. The intuition is as follows: if j is large, i.e. the algorithm executes many small jobs in the first step, the adversary chooses pattern A. In this case, there will be j idle machines during the second step. The optimal solution would delay all short jobs to time 2 for a loss of j, however, all jobs complete by time 2. If j is small, there are many jobs left and we try to delay as many of them as possible.

Let us first show some auxiliary results that help bound the quality of any algorithm:

Remark 2.2.2. For one machine, the optimal sum of completion times is given by scheduling the jobs in order of non-descending length.

In particular, for a jobs of length 1 and b jobs of length 2, a sum of completion times of

$$\sum C_j = a(a+1)/2 + ab + b(b+1) \tag{2.1}$$

is achieved.

Proof. The general result is folklore by now and can for example be seen as a special case of McNaughton's result [McN59] that SRPT is optimal even on m machines.

[McN59] R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 1959.

As to the second claim, it remains to observe that we have jobs terminating at times $1, 2, \ldots, a, a+2, a+4, \ldots, a+2b$, so the total sum is

$$\sum C_j = \sum_{i=1}^{a} i + \sum_{i=1}^{b} (a+2i) = \frac{a(a+1)}{2} + ab + 2\frac{b(b+1)}{2}, \tag{2.2}$$

as claimed. □

Lemma 2.2.3. *If the adversary chooses pattern A, and j short jobs have been completed by the algorithm ALG in the first step, we have*

$$ALG \geq A_k(j) := 4k + \frac{(j-1)j}{2}. \tag{2.3}$$

Proof. By setting, j short jobs terminate at time 1, so at most $k-j$ long jobs were started at time 1, and all available jobs can run at time 2. Of these jobs, $k-j$ are short. Hence, at the end of time 2, at best all short jobs and $k-j$ long jobs have terminated, leaving at least j long jobs with 1 unit of remaining processing time each, which are executed at time $3, 4, \ldots, 2+j$. In total, the sum of completion times is at least

$$j \cdot 1 + (k-j) \cdot 2 + (k-j) \cdot 2 + \sum_{i=1}^{j} (2+i) = -j + 4k + \sum_{i=1}^{j} i$$

as claimed. Note in particular that $A_k(j)$ is increasing in j, hence its minimum is $A_k(0) = 4k$. □

Corollary 2.2.4. If the adversary chooses A, it holds that $\mathrm{OPT} = 4k$.

Proof. We know that $\mathrm{OPT} \leq 4k = A_k(0)$, because $A_k(0)$ is achieved by executing only large jobs in the first step. Assume that $\mathrm{OPT} < 4k$ holds. Since we have $2k$ jobs, this means there is some number j of (short by neccessity) jobs that terminate at time 1. As reasoned above, this means that there are at least j jobs that have not terminated by the end of step 2. Hence, for each job that terminates at time 1, there is a job that terminates at time 3 or later, which contradicts the assumption that the average completion time is strictly less than 2. □

Lemma 2.2.5. *If the adversary chooses B, and j short jobs have been completed by ALG in the first step, we have*

$$ALG \geq B_k(j) := 2k^2 + 3k - 2kj + j^2\,, \tag{2.4}$$

which is decreasing in j.

Proof. In this case, j jobs terminate at time 1. At this time, of the $2k-j$ remaining jobs, at least j jobs are still long and at most $2(k-j)$ are short. By applying Remark 2.2.2 with an offset of $+1$ for all $2k-j$ remaining jobs, we get a total sum of

$$\sum C_j \leq j+(2k-j)+(k-j)(2k-2j+1)+2(k-j)j+j(j+1) = 2k^2+3k-2kj+j^2\,. \tag{2.5}$$

As to the monotonicity, we note that for $j < k$, we have

$$B_k(j+1) - B_k(j) = -2k(j+1) + (j+1)^2 + 2kj - j^2 = -2k + 2j + 1 \leq -1\,, \tag{2.6}$$

as required. □

Corollary 2.2.6. If the adversary chooses B, it holds that $\mathrm{OPT} \leq k^2 + 3k$.

From this, we can immediately conclude:

Corollary 2.2.7. If the algorithm executes j short jobs during the first step, the adversary can force a competitive ratio of at least

$$\max\{A_k(j)/A_k(0), B_k(j)/B_k(k)\}\,. \tag{2.7}$$

Considering all values of j and k, this implies:

Corollary 2.2.8. The competitive ratio of any online algorithm is bounded from below by

$$\sup_{k>0} \min_{1\leq j\leq k} \max\{A_k(j)/A_k(0), B_k(j)/B_k(k)\} \tag{2.8}$$

Note that for any k, $A_k(j)/A_k(0)$ becomes 1 for $j = 0$ and is increasing in j; similarly, $B_k(j)/B_k(k)$ equals 1 for $j = k$ and is decreasing in j. This means that to find the minimal maximum it is sufficient to solve the quadratic equation

$A_k(j)/A_k(0) = B_k(j)/B_k(k)$ for j, and a solution must exist. Using

$$A_k(j)B_k(k) = (k^2+3k)(j^2/2-j/2+4k) = (k/2)\cdot((k+3)j^2-(k+3)j+8k^2+24k) \tag{2.9}$$

and

$$B_k(j)A_k(0) = (j^2 - 2kj + 3k + 2k^2)(4k) = (k/2)\cdot(8j^2 - 16kj + 16k^2 + 24k) \tag{2.10}$$

and simplifying, we need to solve

$$(k+3)j^2 - (k+3)j + 8k^2 = 8j^2 - 16kj + 16k^2\,,$$

i.e.

$$(k-5)j^2 + (15k-3)j - 8k^2 = 0\,.$$

For $k \leq 5$, the values are shown in Table 2.1; for $k \geq 6$ we can exploit that a quadratic equation $ax^2 + bx + c$ is solved by

$$x_{1,2} = \Big\{ -\frac{1}{2a}\big(-b \pm \sqrt{b^2 - 4c^2}\big) \Big\}$$

to obtain the solution

$$j_{1,2} = \Big\{ \frac{3 - 15k \pm \sqrt{32k^3 + 65k^2 - 90k + 9}}{2k - 10} \Big\}. \tag{2.11}$$

Abbreviating the radicand $\Delta := \Delta(k) = 32k^3 + 65k^2 - 90k + 9$, we note that for $k \geq 6$, we have

$$\begin{aligned} \Delta(k) &= 32k^3 + 65k^2 - 90k + 9 \\ &\geq (32\cdot 6)k^2 + 50k^2 + (15\cdot 6)k - 90k + 9 > 242k^2 \geq (15k)^2 > 0\,, \end{aligned} \tag{2.12}$$

so both solutions are real-valued and the numerator in (2.11) is positive for the 'plus' branch and negative for the 'minus' branch. The denominator $2k - 10$ is positive for all $k > 5$, so the 'minus' branch of (2.11) is negative for $k > 5$, while the 'plus' branch is positive. Setting $j^* := \frac{3-15k+\sqrt{\Delta}}{2k-10} \in \Theta(\sqrt{k})$ and substituting this

Table 2.1: $A_k(j)/A_k(0)$ and $B_k(j)/B_k(k)$ for $k \le 5$; lower bound shown bold

(a) Values of $A_k(j)/A_k(0)$

	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$k=2$	1	1	9/8			
$k=3$	1	1	**13/12**	15/12		
$k=4$	1	1	17/16	19/16	22/16	
$k=5$	1	1	21/20	**23/20**	26/20	30/20

(b) Values of $B_k(j)/B_k(k)$

	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$k=2$	14/10	**11/10**	1			
$k=3$	27/18	22/18	19/18	1		
$k=4$	44/28	37/28	**32/28**	29/28	1	
$k=5$	65/40	56/40	49/40	44/40	41/40	1

choice into $A_k(j)/A_k(0)$, we obtain a lower bound of

$$\begin{aligned} A_k(j^*)/A_k(0) &= \frac{4k + j^*(j^*-1)/2}{4k} = 1 + \frac{(j^*)^2 - j^*}{8k} \\ &= 1 + \Big(\frac{(3-15k)^2 + (6-30k)\sqrt{\Delta} + \Delta}{32k^3 + \Theta(k^2)} \\ &\qquad - \frac{3 - 15k + \sqrt{32k^3 + 65k^2 - 90k + 9}}{16k^2 - \Theta(k)} \Big) \\ &= 1 + \frac{32k^3 + O(k^{2.5})}{32k^3 + \Theta(k^2)} - o(1) \\ &\to 2\,. \end{aligned} \tag{2.13}$$

This proves Theorem 2.2.1; Figure 2.2 shows the values of the lower bound for some small values of k.

As mentioned above, a classical result by McNaughton [McN59] is that SRPT is optimal even on multiple machines as long as the completion times are unweighted. This is of course not possible in our setting by Theorem 2.2.1; but as the following example demonstrates, SRPT does not even have constant competitive ratio.

Theorem 2.2.9. *For* Pm,*fail* | *pmtn* | $\sum_j C_j$, *the competitive ratio of SRPT is* $\Omega(n)$.

Proof. For $m \in \mathbb{N}$, m even, consider m machines and m small jobs with $p_1 = \cdots = p_m = 1$ and $m/2$ large jobs with $p_{m+1} = \cdots = p_{m+m/2} = 2$. Clearly, $n = \Theta(m)$. We

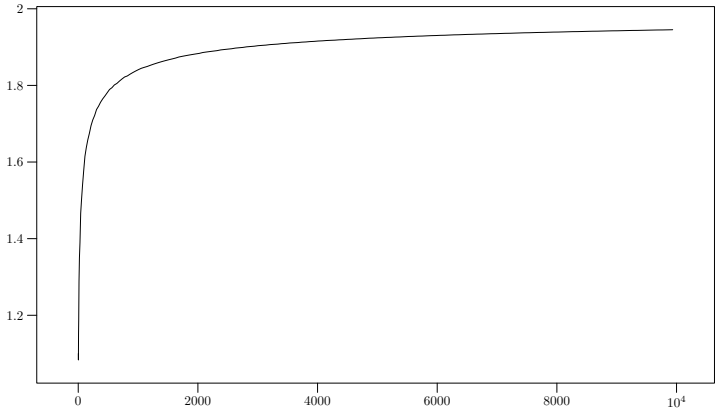
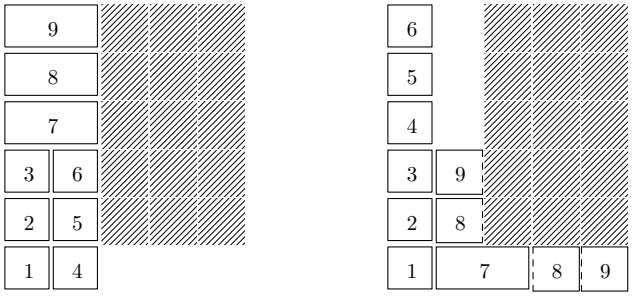


Figure 2.2: Lower bounds on competitive ratio for small k



(a) Optimal schedule (b) SRPT schedule

Figure 2.3: Optimal and SRPT schedule for $m = 6$ machines.

set $m(1) = m(2) = m$ and $m(t) = 1$ for every $t > 2$.

As shown in Figure 2.3, SRPT generates a schedule σ_2 by starting the m small jobs at time 1 resulting in $C_j(\sigma_2) = 1$ for each $j \in \{1, \ldots, m\}$. At time 2 all of the $m/2$ large jobs are started; however, they cannot be finished at time 2 but as time proceeds, each of them gets executed in a successive time step. This means that $C_{m+j}(\sigma_2) = 2 + j$ holds for each $j \in \{1, \ldots, m/2\}$. In total, we obtain $\sum C_j = m + \sum_{j=1}^{m/2}(2 + j) = \Omega(m^2)$.

A better schedule will start all long jobs at time 1 and finishes all jobs by time 2, for $\sum C_j \leq 3m$. □

2.2.2 SRPT for special availability patterns

Throughout this section, we assume the following availability pattern which has been previously studied for min-max objectives [SS98]:

Definition 2.2.10. Let $m : \mathbb{N} \to \mathbb{N}$ the machine availability function; m forms an *increasing zig-zag pattern* iff the following condition holds:

$$\forall t \in \mathbb{N} : m(t) \geq \max_{t' \leq t} m(t') - 1 \,.$$

Intuitively, we may imagine that machines may join at any time and that only one of the machines is unreliable. An example is shown in Figure 2.4.

Lemma 2.2.11. *For every schedule σ, we can find a schedule σ' such that $p_i < p_j$ implies $C_i(\sigma') \leq C_j(\sigma')$ for each $i, j \in \{1, \ldots, n\}$ and $\sum_{j=1}^{n} C_j(\sigma') \leq \sum_{j=1}^{n} C_j(\sigma)$.*

Proof. Fix a schedule σ. Let $i, j \in \{1, \ldots, n\}$ with $p_i < p_j$ but $C_i > C_j$, as sketched in Figure 2.5.

Let I_i, I_j be the sets of times in which i, j are executed in σ, respectively. We have $0 < |I_i \setminus I_j| < |I_j \setminus I_i|$ since $p_i < p_j$, $C_i > C_j$. Let $g : I_i \setminus I_j \to I_j \setminus I_i$ be an injective mapping; construct a schedule σ' from σ in the following way: for all $t \in I_i \setminus I_j$ exchange the execution of job i at time t with the execution of job j at time $g(t)$.

[SS98] E. Sanlaville and G. Schmidt. Machine scheduling with availability constraints. *Acta Informatica*, 1998.

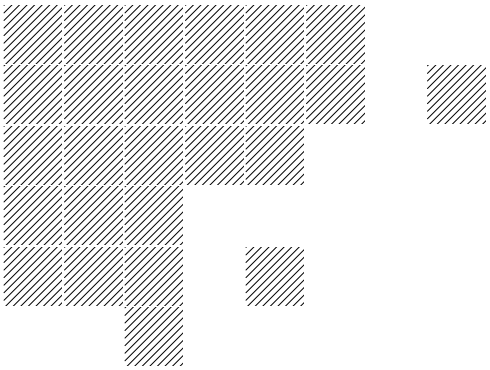
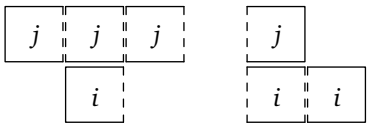
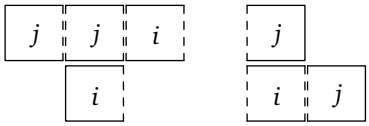


Figure 2.4: Increasing zig-zag availability pattern



(a) before reordering



(b) after reordering

Figure 2.5: Reordering jobs in Lemma 2.2.11

Then we have $C_k(\sigma) = C_k(\sigma')$ for every $k \neq i, j$, furthermore $C_i(\sigma) = C_j(\sigma')$ and $C_j(\sigma) \geq C_i(\sigma')$. Iterating the construction yields the claim. □

Theorem 2.2.12. *SRPT is an optimal algorithm if machine availabilities form an increasing zig-zag pattern.*

Proof. Assume a counterexample I with jobs $1, \ldots, n$ and m such that $\sum_{j=1}^{n} p_j$ is minimal. Fix an optimal schedule σ_{OPT} and an SRPT schedule σ_{ALG} such that the set D of jobs that run at time 1 in only one of $\sigma_{\mathrm{OPT}}, \sigma_{\mathrm{ALG}}$ is of minimal size.

If $|D| = 0$, then σ_{OPT} and σ_{ALG} coincide at time 1 up to permutation of machines. In this case, denote with C the set of jobs running at time 1. (This set is the same for σ_{OPT} and σ_{ALG}.) By definition of SRPT, $C \neq \emptyset$. We define a new instance I' by setting

$$\forall j = 1, \ldots, n : p'_j := \begin{cases} p_j - 1, & j \in C, \\ p_j, & j \notin C \end{cases}$$

$$\forall t \in \mathbb{N} : m'(t) := m(t+1).$$

Every solution to I' of value s induces a solution to I that has value $s + n$ by running the jobs of C in step 1; on the other hand, every solution to I of value s that runs C in the first step induces a solution of value $s - n$ to I'. In particular, $\mathrm{OPT}(I) = \mathrm{OPT}(I') + n$ and it is easy to see that $\mathrm{SRPT}(I) = \mathrm{SRPT}(I') + n$. (In this case, C is a selection of the shortest jobs in I, and after decreasing them, they will still be the shortest jobs in I'.) Since $\sum_{j=1}^{n} p'_i < \sum_{j=1}^{n} p_i$, we conclude that I' is not a counterexample, so $\mathrm{OPT}(I) = \mathrm{OPT}(I') + n = \mathrm{SRPT}(I') + n = \mathrm{SRPT}(I)$, so I is not a counterexample, either, which contradicts our assumption.

Hence, $D \neq \emptyset$.

We will now argue that there must be some job run by σ_{OPT} that is not run by σ_{ALG} at time 1 and vice versa and then show that we can exchange these jobs in σ_{OPT} without increasing the objective function value, leading to a counterexample of smaller $|D|$.

Assume that all jobs run by σ_{OPT} also run in σ_{ALG}. Since $|D| > 0$, there is some job in σ_{ALG} that is not in σ_{OPT}, hence σ_{OPT} contains an idle machine. Hence, all n

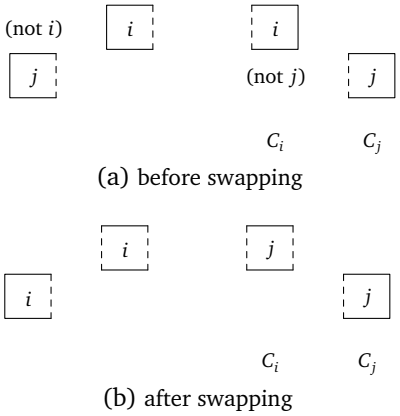


Figure 2.6: Case 1 in Theorem 2.2.12

available jobs must run in σ_{OPT} at time 1 by optimality, a contradiction to $|D| > 0$.

Thus there is a job j run by σ_{OPT} which is not run by σ_{ALG}. Since not all n jobs can run in σ_{ALG} at time 1 and SRPT is greedy, there must be a different job i which is run in σ_{ALG}, but not in σ_{OPT}. By definition of SRPT, we may assume $p_i < p_j$, and $C_i(\sigma_{\text{OPT}}) \leq C_j(\sigma_{\text{OPT}})$ by Lemma 2.2.11.

We will now show that it is always possible to modify σ_{OPT} to execute job i at time 1 instead of job j. Preferring job i will decrease its completion time by at least 1, so it is sufficient to show that the total sum of completion times of the other jobs is increased by at most 1. This would then yield a counterexample of smaller $|D|$, and iteratively, we arrive at $D = \emptyset$.

Case 1: if job j does not run at time C_i in σ_{OPT}, we have $C_j > C_i$ and we can execute job i at time 1 and job j at time C_i, cf. Figure 2.6. This does not increase the completion time C_j, and any other job's completion time remains unchanged.

Case 2: The following construction is sketched in Figure 2.7. In this case, job j does run at time C_i. We will execute job i at time 1 and job j at time $C_j + 1$ for a total change of $\sum C_j$ of at most 0. This can trivially be done if there is an idle machine in σ_{OPT} at time $C_j + 1$. Otherwise, there are $m(C_j + 1)$ jobs running at that time. We still have an idle machine at time C_i, freed up by moving J_i to time 1, and want to displace one of the $m(C_j + 1)$ jobs into this space. We note that we may not choose jobs that are already running at time C_i. There are at most $m(C_i) - 2$ such jobs, since we know jobs i and j are running at that time. By the

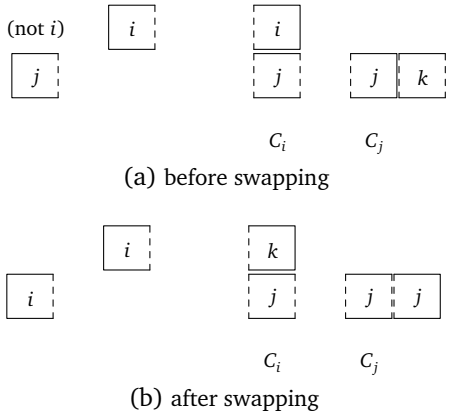


Figure 2.7: Case 2 in Theorem 2.2.12

increasing zig-zag condition and $C_i \leq C_j < C_j + 1$, we know that

$$m(C_j + 1) \geq m(C_i) - 1 > m(C_i) - 2\,,$$

so at least one job, say job k, is not excluded. Since no part of k is delayed, C_k does not increase. □

By letting the length of the individual timestep go to 0, we obtain

Corollary 2.2.13. SRPT is optimal for increasing zig-zag pattern even for non-integral failure times without lookahead.

It would be interesting to extend this result to the setting with release times. It is possible to generalize Lemma 2.2.11 to this setting: if we denote with $p_{i@t}$ the remaining processing time of i at time t (under some schedule σ which is clear from context) with the understanding that $p_{i@t}$ is undefined for $t < r_i$, i.e. before the job is released, we can show by the same proof as for Lemma 2.2.11:

Lemma 2.2.14. *For every schedule σ, there is a schedule σ' that is at least as good as σ (in terms of $\sum_j C_j$) and satisfies*

$$p_{i@t} < p_{j@t} \implies C_i \leq C_j \tag{2.14}$$

for all i, j, t where defined.

However, SRPT is in fact not optimal in the presence of release times, as the following example shows, the key problem being that the application of Lemma 2.2.11

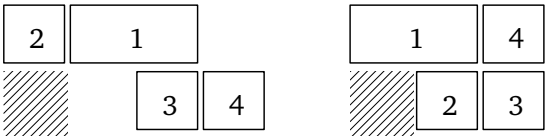


Figure 2.8: An example that SRPT is suboptimal in the presence of release times.

(a) SRPT schedule of value 11

(b) Optimal schedule of value 10

increases the makespan of the schedule:

Example 2.2.15. Consider four jobs with $p_1 = 2$, $p_2 = p_3 = p_4 = 1$, $r_1 = r_2 = 0$, $r_2 = r_3 = 2$. For the machines, set $m(1) = 1$ and $m(t) = 2$ for $t > 1$. As shown in Figure 2.8, the SRPT schedule has total value $1 + 3 + 3 + 4 = 11$, while an optimal schedule has value $2 + 2 + 3 + 3 = 10$.

2.2.3 Algorithm MIMIC

The basic idea of algorithm MIMIC is to use an offline approximation for reliable machines and re-use this given schedule as far as possible. More precisely, let us assume that we already have an α-approximate schedule σ for the offline case for an objective in $\{\sum w_j C_j, \sum C_j, C_{\max}\}$. We will first convert the schedule into a queue Q in the following way; we note that this is for expository reasons and not needed in the implementation.

(2.15) For any time t and any machine $i \in \{1, \dots, m\}$, the job running at time t on machine i in schedule σ is at position $(t-1)m + i$ in the queue.

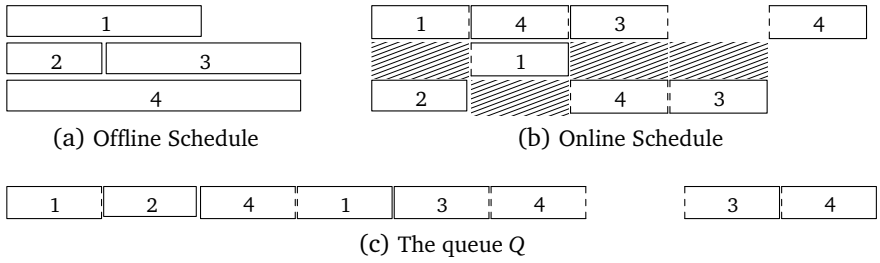
Note that this means "idle" positions may occur in the queue; this is not exploited. We can now use the queue in our online scheduling algorithm in Algorithm 2.1.

Remark 2.2.16. In the generated schedule, no job runs in parallel to itself.

Proof. We assume without loss of generality that there are no redundant preemptions in the offline schedule σ, i.e. if a job j runs at time t as well as at time $t+1$, it remains on the same machine. Note that this is trivial if the offline schedule is non-preemptive; if it is preemptive, we can simply reorder the jobs accordingly. Hence, two entries in the queue corresponding to the same job must be at least m

Algorithm 2.1: Algorithm MIMIC for independent jobs

Setup Calculate the schedule queue Q ;
Upon *time t* **do**
  Let $m(t)$ the number of available machines for the next step;
  Preempt all currently running jobs;
  Remove the first $\max\{m(t), |Q|\}$ jobs from Q and schedule them;
  Wait until time $t+1$;



(a) Offline Schedule (b) Online Schedule

(c) The queue Q

Figure 2.9: Example of algorithm MIMIC's behaviour

positions apart. Since at no time in the online schedule, more than m machines are available, no two entries of the same job can be eligible simultaneously. □

Example 2.2.17. Figure 2.9 shows an offline schedule without machine failure, the corresponding queue Q, and an online schedule generated by MIMIC in case of machine failure.

To bound the loss we incur, we now take a different view upon machine failure: instead of imagining failed machines, we consider that "failure blocks" are inserted into the queue. Since there is a one-to-one correspondence of machine/time positions and queue positions given by (2.15), this is equivalent to machine failures. We recall an elementary probabilistic fact:

Remark 2.2.18 (Expected run length). If in every timestep, every machine fails independently with probability f, the expected number of failure blocks immediately in front of each non-failure block is $f/(1-f)$.

We can now bound how long the expected completion of a single job is delayed in the online schedule σ':

Lemma 2.2.19. *For any job j, we have* $E[C_j(\sigma')] \leq \frac{1}{1-f} C_j(\sigma) + 1$.

Table 2.2: Selection of known offline results that can be used by MIMIC.

Setting	Source		ax. ratio
$P \mid pmtn \mid \sum_j C_j$	McNaughton	[McN59]	1
$P \mid\mid \sum_j w_j C_j$	Kawaguchi and Kyan	[KK86]	$(1+\sqrt{2})/2$
$P \mid r_j, pmtn \mid \sum_j w_j C_j$	Afrati et al.	[ABC$^+$99]	PTAS
$P \mid r_j, prec, pmtn \mid \sum_j w_j C_j$	Hall et al.	[HSW96]	3

Proof. We note that since there are always m machines in the offline setting, there cannot be any blocks corresponding to j in the queue after position $mC_j(\sigma)$ before failure blocks are inserted. This means that after random insertion of the failure blocks, the expected position of the last block of job j is at most $\big(1 + f/(1-f)\big)mC_j(\sigma)$. In light of (2.15), this yields

$$E[C_j(\sigma')] = \lceil \frac{1}{m}(mC_j(\sigma)\frac{1}{1-f}) \rceil \leq \frac{1}{1-f}C_j(\sigma) + 1\,,$$

which proves the claim. □

Theorem 2.2.20. *MIMIC has asymptotic approximation ratio* $1/(1-f)$ *for unweighted sum of completion times and* $(1+\epsilon)/(1-f)$ *for sum of weighted completion times with release dates.*

This is achieved by exploiting known offline results for different settings (cf. Table 2.2). We should note in particular that since machine failure at most delays a job, our model is applicable to settings with non-zero release dates. We list the result of Kawaguchi & Kyan [KK86] mainly because it is obtained by a very simple *largest ratio first* heuristic, as opposed to the more sophisticated methods of Afrati et al. [ABC$^+$99], which gives it a very low computational complexity.

We note that our results stated so far cannot be simply used if there are general precedence constraints, as the following example shows:

[KK86] T. Kawaguchi and S. Kyan. Worst case bound of an LRF schedule for the mean weighted flow-time problem. *SIAM Journal on Computing*, 1986.

[ABC$^+$99] F. N. Afrati, E. Bampis, C. Chekuri, D. R. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proc. FOCS*, 1999.

(a) valid offline schedule

(b) invalid online schedule generated by MIMIC

Figure 2.10: MIMIC fails for general precedence constraints

Example 2.2.21. Consider four jobs $1, \ldots, 4$ of unit execution time such that $\{1,2\} \prec \{3,4\}$. The queue 1234 corresponds to an optimal offline schedule. If a failure occurs during the first time step, we have $C_2 = 2$ and MIMIC schedules one of jobs 3 and 4 in parallel to job 2, as shown in Figure 2.10.

The main problem is that jobs 2 and 3 have a distance of $1 < m = 2$ in the queue, so they may be scheduled for the same time step online even though there is a precedence constraint on them. Conversely, if the distance is at least m, they are never scheduled for the same time step.

Since our setting allows free migration of a job from one machine to another, we can sometimes avoid this situation: if the precedence constraints form an *in-forest*, i.e. every job has at most one direct successor, we can rearrange the jobs in the following way: if, in the offline schedule, a job j is first started at some time t, and $1, \ldots, k, k \geq 1$ are those of job j's direct predecessors that run at time $t-1$, w.l.o.g. on machines $1, \ldots, k$, we assign job j to machine k. This ensures that the distance in the queue from job j to job k and hence also to job $1, \ldots, k-1$ is at least m. This construction is always possible, because j will be the only job to depend immediately on job k.

If we have general precedence constraints, we cannot guarantee that all jobs are sufficiently segregated from their predecessors, as seen above. As we will study in the next section, this can be remedied if we allow preemptions at arbitrary points.

2.3 Scheduling with limited lookahead

In this section, we will adapt the idea of algorithm MIMIC—reusing an offline approximation—to the more general semi-online setting by methods similar to

Prasanna & Musicus' continuous analysis [PM96]. In the semi-online setting, changes of machine availability and preemptions may occur at any time whatsoever, however, we know in advance the next point in time when a change of machine availability will take place. We can use this knowledge to better convert an offline schedule into an online schedule, using the algorithm MIMIC′ given as Algorithm 2.2: during each interval of constant machine availability, we calculate the area $m(t)\delta$ we can schedule. This area will be used up in time $m(t)\delta/m$ in the offline schedule. We take the job fractions as executed in the offline schedule and schedule them online with McNaughton's wrap-around rule [McN59]. Precedence constraints can be handled by suitable insertion of artificial interruptions. An example of this correspondence is shown in Figure 2.11.

Algorithm 2.2: Algorithm MIMIC′

Setup Calculate offline schedule σ_{offline};
Set $t_{\text{offline}} := 0$;
Upon *time* t **do**
  Let δ such that the next event is at time $t+\delta$;
  Let $m(t)$ the number of machines available in the interval $[t, t+\delta[$;
  Set $\delta_{\text{offline}} = \min\{m(t)\delta/m, \min\{C_j(\sigma_{\text{offline}}) - t_{\text{offline}} | t_{\text{offline}} \le C_j(\sigma_{\text{offline}})\}\}$;
  Set $\delta_{\text{online}} = m\delta_{\text{offline}}/m(t)$;
  Schedule all job fractions that run in the interval $[t_{\text{offline}}, t_{\text{offline}} + \delta_{\text{offline}}[$ in σ_{offline} in the online interval $[t, t+\delta_{\text{online}}[$ using McNaughton's rule;
  Set $t_{\text{offline}} := t_{\text{offline}} + \delta_{\text{offline}}$;
  Wait until time $t+\delta_{\text{online}}$;

Since at time $C_j(\sigma_{\text{offline}})$, a total area of $mC_j(\sigma_{\text{offline}})$ is completed, we have the following bound on the online completion times $C_j(\sigma_{\text{online}})$:

$$\int_0^{C_j(\sigma_{\text{online}})} m(t)dt \le mC_j(\sigma_{\text{offline}}). \tag{2.16}$$

If we set $\forall t : E[m(t)] = (1-f)m$ to approximate our independent failure setting above, equation (2.16) simplifies to $C_j(\sigma)(1-f)m \le mC_j(\sigma_{\text{offline}})$, which again

[PM96] G. N. S. Prasanna and B. R. Musicus. The optimal control approach to generalized multiprocesor scheduling. *Algorithmica*, 1996.

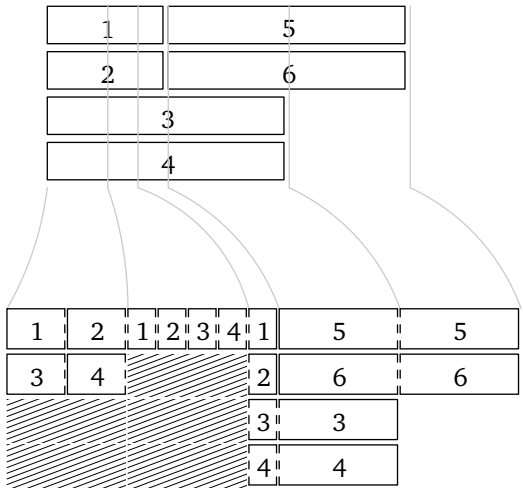


Figure 2.11: Example of MIMIC′ behaviour. Top: offline pre-generated schedule, bottom: online schedule generated

yields a $1/(1-f)$-approximation as in Lemma 2.2.19, thus we obtain the following result.

Theorem 2.3.1. *Algorithm MIMIC' non-asymptotically matches the approximation rates of MIMIC for the continuous semi-online model.*

Note that the key property of algorithm MIMIC is the following: for every online point in time t, there is an offline time t' such that MIMIC has executed in $[0,t]$ exactly those job fractions that are executed in $[0,t']$ in the offline schedule.

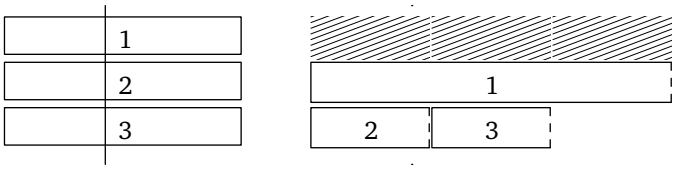
Observing that an important piece of this result is the possibility to generate a new fractional schedule optimally, it seems natural to study a generalization to related machines. This is motivated by the following result:

Theorem 2.3.2 (Liu and Yang [LY74])**.** *The heuristic LRPTFM (Longest Processing Time on the Fastest Machine) minimizes the makespan (in the absence of machine failures), and the optimal makespan is*

$$\max\left\{\frac{\sum_{j=1}^{n} p_j}{\sum_{i=1}^{m} s_i}, \frac{\sum_{j=1}^{k} p_j}{\sum_{i=1}^{k} s_i} : k = 1,\dots,m\right\}, \tag{2.17}$$

where we suppose $p_1 \geq p_2 \geq \dots p_n$ *and* $s_1 \geq s_2 \geq \dots s_m$.

[LY74] J. W. S. Liu and A.-T. Yang. Optimal scheduling of independent tasks on heterogeneous computing systems. In *ACM 74: Proceedings of the 1974 annual conference*, 1974.



(a) Offline schedule (b) Online schedule

Figure 2.12: MIMIC on related machines will leave machines idle

This is not sufficient, however: the newly-generated schedule should also use all machines. This is not the case, as the following example shows:

Example 2.3.3. Consider three jobs of lengths $6, 2, 2$, respectively, and three machines of speeds $3, 1, 1$. There is an obvious schedule of makespan 2. Consider now a failure of the fast machine in the interval $[0, 1]$. Since the remaining machines have speed 1 only, MIMIC can at most schedule the offline interval $[0, 1/3]$ as shown in Figure 2.12 and will consequently leave one of the remaining machines idle even though no machine is idle in the original schedule.

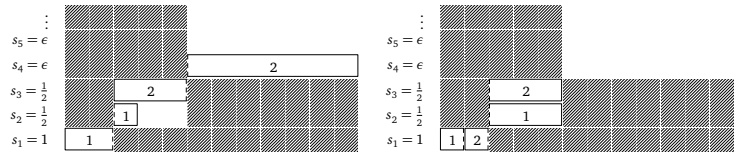
We remark that by Theorem 2.3.2, we can bound the error in concrete cases and make a *a posteriori* evaluation of the schedule, but since the results depend on the combination of machine speeds and job lengths, a concise closed expression like Theorem 2.2.20 or (2.16) giving the expected loss in approximation quality is unlikely to exist.

However, we can still show results for the makespan objective, which we will devote the rest of the section to.

Albers and Schmidt already proved that without preemptions, the problem with online failures is essentially untreatable [AS01]:

Lemma 2.3.4. P,*fail* | *pmtn* | $C_{\max}$ *with online failure does not admit algorithms with constant competitive ratio.*

To circumvent this problem, they propose to force that at least one machine should be available at any given point in time. (Given that we are allowed preemption and migration without penalty, without loss of generality this is always the same machine.) Unfortunately, this additional restriction is not strong enough for the case of related machines, even when we always have at least a constant amount of processing power available:



(a) Algorithm's schedule (b) Optimal schedule

Figure 2.13: Inapproximability without lookahead

Lemma 2.3.5. *For any constant $S > 0$, the problem* Q,*fail* | *pmtn* | $C_{\max}$ *with online failure does not admit algorithms with constant competitive ratio, even when at any point in time, the total speed of available machines is at least S.*

Proof. Fix any $S > 0$ and assume for sake of contradiction that there is an algorithm A that has competitive ratio $c \geq 1$. Consider the following instance adapted from [AS01]: initially, there are two jobs 1 and 2, both of length S, and one machine M_1 of speed S. Denote with t_0 the time at which algorithm A first preempts or terminates a job. By symmetry, we suppose without loss of generality that this is job 1 and note that by definition, only job 1 runs during the interval $[0, t_0[$.

If $t_0 > 2c \geq 2$, then A is not c-competitive because the overall makespan will be at least $t_0 + 1 \geq 2c + 1$, whereas an optimal schedule will start job 1 at time 0 and job 2 at time 1 for a makespan of 2.

In the interval $[t_0, 2[$, which might be empty, machine M_1 becomes unavailable and is replaced by two machines M_2, M_3 of speeds $s_2 = s_3 = S/2$, which are available until time 2. This is sufficient for an optimal schedule, which will schedule job 1 on M_1 in the interval $[0, t_0/2)$, job 2 on M_1 in the interval $[t_0/2, t_0)$ and both jobs in the interval $[t_0, 2)$, cf. Figure 2.13. At time 2, either job will have remaining time $q_1 = q_2 = S - (t_0/2) \cdot S - (2 - t_0) \cdot S/2 = 0$. However, under the schedule generated by A, job 2 has remaining time at least

$$S - (2 - t_0) \cdot S/2 = S \cdot t_0/2 > 0\,, \tag{2.18}$$

because by definition, it runs only in the interval $[t_0, 2)$ and on at most one of the two machines at any time.

At time 2, machines M_2 and M_3 become unavailable and are replaced by $k := 2\lceil 2c/t_0 \rceil + 1$ machines of speed S/k. Clearly, A can only use one of these machines,

and needs it for at least

$$\frac{S \cdot t_0/2}{S/k} = k \cdot t_0/2 = (2\lceil 2c/t_0 \rceil + 1) \cdot t_0/2 \geq 2c + t_0/2 > 2c \tag{2.19}$$

units of time, which means its total makespan is at least $2c + 2$, which means that A is at best $c + 1$-competitive. □

The crucial point here is that in the related-machine case, a machine that is present might still be so slow as to be nearly useless. Hence, we have to make the somewhat stronger assumption that the machines are not arbitrarily slow, i.e. by rescaling, we assume that $s_i \geq 1$ for all machines i with $s_i > 0$. For notational simplicity, we will also assume that the number of machines is always at least the number of jobs, with extra machines having speed 0, and that there is always a job of length 0.

2.3.1 The LRPTFM heuristic

One important subroutine of our algorithms is a modification of the *Longest Remaining Processing Time on the Fastest Machine* (LRPTFM) heuristic. As the name suggests, this algorithm maintains at every point in time that for every $k \in \{1, \ldots, n\}$, the k jobs with the longest remaining processing times are currently executed on the k fastest machines. We note that as stated, this is not an algorithm in the strictest sense of the word, since it usually does not terminate: it generates an infinite number of preemptions without advancing the schedule, as the following example shows:

Example 2.3.6. Consider two jobs of the same length $p_1 = p_2$, and one available machine. Without loss of generality, LRPTFM breaks the tie in favour of job 1 and starts executing it. After an infinitesimal amount of time ι, job 1's remaining processing time becomes smaller than that of job 2, so it is preempted in favour of job 2. After 2ι, job 2 will again become shorter than job 1 and so on.

We can only interpret this schedule as a relaxation in the sense that a machine is allowed to execute multiple jobs concurrently, dividing its speed between them arbitrarily, and a job is allowed to be executed fractionally on multiple machines

at the same time, as long as the fractions add up to 1 over all machines. The importance of the heuristic lies in the following classical result:

Theorem 2.3.7 (Liu and Yang [LY74])**.** *LRPTFM minimizes the makespan (in the absence of machine failures), and the optimal makespan is*

$$\max\left\{\frac{\sum_{j=1}^{n} p_j}{\sum_{i=1}^{m} s_i}, \frac{\sum_{j=1}^{k} p_j}{\sum_{i=1}^{k} s_i} : k = 1, \ldots, m\right\}, \tag{2.20}$$

where we suppose $p_1 \geq p_2 \geq \ldots p_n$ *and* $s_1 \geq s_2 \geq \ldots s_m$.

Such a schedule is obviously not reasonably displayed as the usual Gantt chart, we will therefore use a different graphical display in the following: namely, we plot, for every job, its remaining processing time vs. time, which we will call a RPT diagram. The slope of the line then indicates the speed of the machine a job is running on, and LRPTFM's infinite number of preemptions corresponds to a slope which is not naturally occuring among the machines' speeds.

Example 2.3.8. Consider the instance shown in Figure 2.14: two machines are available, one of speed $s_1 = 2$ and one of speed $s_2 = 1$. The initial processing times are 5, 4, 1.5 and 1, respectively. At first, the faster machine is assigned to the longest job and the slower to the job of length 4. At time 1, both these jobs have remaining length 3 and start sharing both machines, resulting in an effective speed of 1.5 per machine. At time 2, the longest three jobs now all have length 1.5. At this point, all four jobs share the machines evenly until they all finish at the same time.

With this in mind, we can extend a partial schedule in the way given as Algorithm 2.3. The clusters defined there are sets of jobs that have the same remaining processing time, i.e. they share a line in LRPTFM's RPT diagram. The schedule is continued until 'two lines meet'. Note that at this point, we do not yet consider machine failures, this will be done by suitable choice of the parameter $\nu_{\max}$.

Example 2.3.9. Continuing Example 2.3.8, Figure 2.15 shows the schedule generated by Algorithm 2.3 for the same instance. The dashed lines show the fractional schedule; note that the solid lines, corresponding to the proper schedule, intersect in all places where the slope of the dashed lines change.

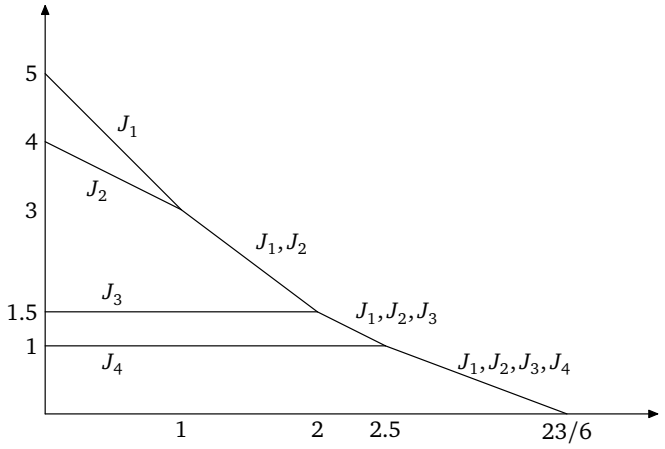


Figure 2.14: Plotting remaining processing time vs. time for LRPT.

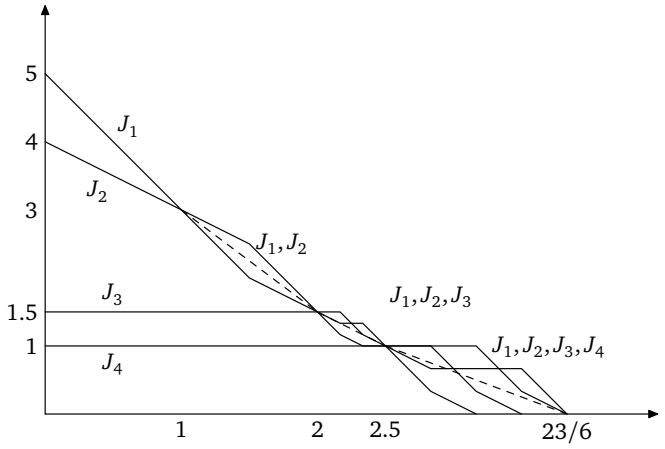


Figure 2.15: Schedule generated by Algorithm 2.3

Algorithm 2.3: Discrete Longest Remaining Processing Time on the Fastest Machine

Input: Jobs with remaining processing times $q_1 \geq \cdots \geq q_n = 0$,
machines with speeds $s_1 \geq \ldots s_m \geq 1$, $s_{m+1} = \cdots = s_n = 0$
maximal schedule span $v_{\max}$
Output: $v > 0$ and a schedule spanning the next v units of time

// Define the clusters and their length
$C_0 := \emptyset$, $\rho_0 := \infty$, $k := 0$;
for $i := 1, \ldots, n$ **do**
  if $q_i = \rho_k$ **then**
    // Continue the current cluster
    $C_k := C_k \cup \{i\}$;
  else
    // Start a new cluster
    $\sigma_k := i - 1$;
    $k := k + 1$;
    $C_k := \{i\}$, $\rho_k := q_i$;

// Find average speed per job in cluster
for $i := 1, \ldots, k-1$ **do**
  $\bar{s}_i := \sum_{j=\sigma_{i-1}+1}^{\sigma_i} s_j / |C_i|$;

// Special case: last cluster has $\rho_k = 0$, finished already
$\bar{s}_k := 0$;
// Find the maximal schedule span length
$v := \min\left\{v_{\max}, \frac{\rho_{i-1} - \rho_i}{\bar{s}_{i-1} - \bar{s}_i} : i = 2, \ldots, k\right\}$;
// Assign the jobs; cluster k is finished already
for $i := 1, \ldots, k-1$ **do**
  for $j := 1, \ldots, |C_i|$ **do**
    for $\ell := 1, \ldots, |C_i|$ **do**
      assign job $\sigma_i + j$ to machine $\sigma_i + 1 + ((j + \ell - 1) \bmod |C_i|)$ in interval $[(\ell - 1) \cdot v/|C_i|, \ell \cdot v/|C_i|[$;

Correctness and usefulness of this procedure is given by the following auxiliary results:

Lemma 2.3.10. *The value v returned by Algorithm 2.3 is positive and finite as long as $q_1 > 0$, $s_1 > 0$ and $v_{\max} > 0$.*

Proof. Note that all values $\rho_i - \rho_{i-1}$ are strictly positive by definition, and since

$$\bar{s}_i = \left(\sum_{j=\sigma_{i-1}+1}^{\sigma_i} s_j \right) / |C_i| \geq s_{\sigma_i} \geq s_{\sigma_i+1} \geq \left(\sum_{j=\sigma_i+1}^{\sigma_{i+1}} s_j \right) / |C_{i+1}|, \tag{2.21}$$

all values $\bar{s}_i - \bar{s}_{i-1}$ are non-negative. □

Lemma 2.3.11. *Let $q_1 \geq \ldots \geq q_n \geq 0$ the remaining processing times Algorithm 2.3 is called with at some time t, $v > 0$ the span it returns, and $C_1, \ldots, C_k$ the non-empty clusters it generates with corresponding lengths $\rho_1 > \cdots > \rho_k = 0$.*

Then, the following statements hold:

1. *At time $t + v$, all jobs in cluster C_i have the same remaining processing time for all $i \in \{1, \ldots, k\}$, which we denote ρ'_i.*
2. *Executing LRPTFM would also result in having all jobs from C_i remaining processing time ρ'_i at time $t + v$, for all $i \in \{1, \ldots, k\}$.*
3. *At time $t + v$, $\rho'_1 \geq \ldots \geq \rho'_k$.*

Proof. Fix some cluster $i \in \{1, \ldots, k\}$. The assignment step in Algorithm 2.3 divides the interval $[t, t + v[$ into $|C_i|$ subintervals of equal length, and every job j of C_i is assigned one subinterval on every machine, so clearly its remaining processing time at the end is

$$q'_j = q_j - \sum_{\ell=1}^{|C_i|} \frac{v}{|C_i|} s_{\sigma_{i-1}+\ell} = \rho_i - v\bar{s}_i \tag{2.22}$$

and this is independent of the choice of j since all jobs in C_i have the same value $q_j = \rho_i$. This proves the first claim.

As to the second claim, note that LRPTFM will by design assign (machines of total fractional) speed $\bar{s}_i$ to every job in cluster C_i, for all i, until some clusters

merge, i.e. their lines in the RPT diagram meet. Consider two clusters i, i' such that $i < i'$. The corresponding lines in the diagram are given by

$$\begin{aligned} y_i(t+v') &= \rho_i - v' \cdot \bar{s}_i \\ y_{i'}(t+v') &= \rho_{i'} - v' \cdot \bar{s}_{i'} . \end{aligned} \tag{2.23}$$

Note that by design, $\bar{s}_i \geq \bar{s}_{i'}$. Solving (2.23) for equality, we obtain an intersection point at

$$v' = (\rho_i - \rho_{i'})/(\bar{s}_i - \bar{s}_{i'}) . \tag{2.24}$$

Since $\rho_i > \rho_{i'}$ and $\bar{s}_i \geq \bar{s}_{i'}$, this value is positive, but possibly infinite.

We claim that $v' \geq v$, the value returned by Algorithm 2.3, for all choices of i, i'. From this, the second and third claim of the theorem will follow, since in the overall interval $[t, t+v[$, both algorithms assign $\bar{s}_i$ to every job in cluster C_i and LRPTFM will maintain the order of the clusters. Assume for sake of contradiction $v' < v$ for some choice of i, i' such that $i < i'$ and $i' - i$ is minimal. By definition of v and (2.24), we know $i' \neq i+1$. At time t, we know $\rho_i > \rho_{i+1} > \rho_{i'}$ by precondition. By choice of v', at time $t+v'$, we have $\rho_i - v'\bar{s}_i = \rho_{i'} - v'\bar{s}_{i'}$. By definition of v and $v' < v$, we still have $\rho_i - v'\bar{s}_i > \rho_{i+1} - v'\bar{s}_{i+1}$, so $\rho_{i'} - v'\bar{s}_{i'} > \rho_{i+1} - v'\bar{s}_{i+1}$. But this means that the pair $i+1, i'$ is also an intersecting pair, and $i' - (i+1) < i' - i$, a contradiction. □

The cornerstone to optimality is captured in the next lemma, which compares execution of Algorithm 2.3, or, equivalently by the previous discussion, LRPTFM, at some time t with any other schedule, in particular an optimal one. Let us first fix notation: we will denote with $q_1 \geq \ldots \geq q_n \geq 0$ the remaining processing times that Algorithm 2.3 starts out with, $\sigma_1 \leq \ldots \leq \sigma_k$ the values it fixes, v the length it returns and $q'_1 \geq \ldots \geq q'_n \geq 0$ the remaining processing times at time $t+v$. By the third part of Lemma 2.3.11, we can see this does not involve re-indexing of the jobs.

The other algorithm starts off at time t with possibly different remaining processing times $q^*_1 \geq \ldots \geq q^*_n \geq 0$, since it may have made different choices earlier on. It will also possibly have different remaining processing times $q^{*\prime}_1 \geq \ldots \geq q^{*\prime}_n \geq 0$ at time $t+v$. Note that here, the permutation of the jobs that sorts them is

possibly different at beginning and end and also different from the one used for Algorithm 2.3. With this terminology, we claim:

Lemma 2.3.12. *Suppose that*

$$\sum_{j=1}^{\sigma_i} q_j \leq \sum_{j=1}^{\sigma_i} q_j^* \tag{2.25}$$

for all $i \in \{1,\dots,k\}$ *with* $q_{\sigma_i} > 0$ *and let* $s_1 \geq \dots s_n$ *the machine speeds, adding dummy machines of speed* 0 *if needed. Then at time* $t+v$*, it holds that*

$$\sum_{j=1}^{\sigma_i} q_j' \leq \sum_{j=1}^{\sigma_i} q_j^{*\prime}\,. \tag{2.26}$$

Proof. Let $i \in \{1,\dots,k\}$. By Lemma 2.3.11 and design of v, we may study the behaviour of LRPTFM instead of that of Algorithm 2.3. By design, we know that

$$\sum_{j=1}^{\sigma_i} q_j' = \sum_{j=1}^{\sigma_i} q_j - v\sum_{j=1}^{\sigma_i} s_j\,. \tag{2.27}$$

For this, it is crucial to note that $q'_{\sigma_i} > 0$ at the beginning of the interval, so all jobs benefit from their machines. On the other hand, we know that

$$\sum_{j=1}^{\sigma_i} q_j^{*\prime} \geq \sum_{j=1}^{\sigma_i} q_j^* - v\sum_{j=1}^{\sigma_i} s_j\,, \tag{2.28}$$

since at any point in time, *no* algorithm can allocate more than $\sum_{j=1}^{\sigma_i} s_j$ processing power to σ_i machines. Plugging this into the precondition yields

$$\sum_{j=1}^{\sigma_i} q_j' = \sum_{j=1}^{\sigma_i} q_j - v\sum_{j=1}^{\sigma_i} s_j \leq \sum_{j=1}^{\sigma_i} q_j^* - v\sum_{j=1}^{\sigma_i} s_j \leq \sum_{j=1}^{\sigma_i} q_j^{*\prime}\,, \tag{2.29}$$

as claimed. □

The same invariant is maintained by release of new jobs:

Lemma 2.3.13. *Let $p \geq 0$, and q, q^* as defined above. Define*

$$q'_i = \begin{cases} q_i & \text{if } q_i > p \\ p & \text{if } q_{i-1} > p \geq q_i \\ q_{i-1} & \text{if } q_{i-1} \leq p \end{cases} \qquad q^{*\prime}_i = \begin{cases} {q^*}_i & \text{if } q^*_i > p \\ p & \text{if } q^*_{i-1} > p \geq q^*_i \\ {q^*}_{i-1} & \text{if } q^*_{i-1} \leq p\,, \end{cases}$$

i.e. the new value p is inserted into the lists in the proper position. Let $k \in \{1, \ldots, n\}$ such that $\sum_{i=1}^{k} q_i \leq \sum_{i=1}^{k} q^*_i$. Then,

$$\begin{cases} \sum_{i=1}^{k} q'_i \leq \sum_{i=1}^{k} q^{*\prime}_i & \text{if } p < q_k \\ \sum_{i=1}^{k+1} q'_i \leq \sum_{i=1}^{k+1} q^{*\prime}_i & \text{if } p \geq q_k\,. \end{cases} \tag{2.30}$$

Proof. First observe that $q'_i \geq q_i$ and $q^{*\prime}_i \geq q^*_i$ for all $i \in \{1, \ldots, n\}$. Consider the case $p < q_k$. Then, we immediately obtain

$$\sum_{i=1}^{k} q'_i = \sum_{i=1}^{k} q_i \leq \sum_{i=1}^{k} q^*_i \leq \sum_{i=1}^{k} q^{*\prime}_i\,. \tag{2.31}$$

If $p \geq q_k$, then we obtain

$$\sum_{i=1}^{k+1} q'_i = \sum_{i=1}^{k} q_i + p \leq \sum_{i=1}^{k} q^*_i + p\,. \tag{2.32}$$

If $p < q^*_{k+1}$, then by definition $\sum_{i=1}^{k+1} q^{*\prime}_i = \sum_{i=1}^{k+1} q^*_i$ and hence

$$\sum_{i=1}^{k} q^*_i + p < \sum_{i=1}^{k+1} q^*_i = \sum_{i=1}^{k+1} q^*_i\,. \tag{2.33}$$

Otherwise, $p \geq q^*_{k+1}$, and then, $\sum_{i=1}^{k+1} q^{*\prime}_i = \sum_{i=1}^{k} q^*_i + p$. In either case, the claim follows. □

From the previous discussion, we conclude

Theorem 2.3.14. *Algorithm 2.4 is optimal for* $\mathrm{Q}, fail \mid r_j \mid C_{\max}$ *with lookahead.*

Algorithm 2.4: Look-Ahead LAFM

while *unfinished jobs exist* **do**
  Query for the time g until the next machine change or job release;
  Call Algorithm 2.3 with the currently available jobs and machines and $v_{\max} = g$;
  Execute the schedule returned;

Proof. We iteratively apply Lemma 2.3.12: the precondition (2.25) is true initially, and it is left invariant by Lemma 2.3.12. It also remains true if machines join or fail, because the invariant does not involve the machines at all. By Lemma 2.3.13, it remains true under insertion of jobs.

Finally, we show that the overall makespan of Algorithm 2.4 is optimal. Let t a time the algorithm is called, such that $t < \mathrm{OPT} \leq t + v$, with v again given by the algorithm, and let $C_1, \ldots, C_i$ the clusters with non-zero remaining processing time at time t. LAFM will not complete any jobs in the interval (because then its cluster would merge with the cluster that has remaining processing time 0 from the start), so all remaining jobs are completed by LAFM exactly at time $t + v$. On the other hand, this means that all $|C_1| + \cdots + |C_i|$ machines used by LAFM are busy until time $t + v$. But this means that the sum of remaining processing times

$$\sum_{j=1}^{|C_1|+\cdots+|C_i|} q_i = v \sum_{j=1}^{|C_1|+\cdots+|C_i|} s_i \tag{2.34}$$

and we conclude by Lemma 2.3.12

$$\sum_{j=1}^{|C_1|+\cdots+|C_i|} q_i^* \geq v \sum_{j=1}^{|C_1|+\cdots+|C_i|} s_i \tag{2.35}$$

so no schedule is able to complete all these jobs in strictly less than v time, and in particularly, they are not able to have a smaller makespan than Algorithm 2.4. □

2.4 Scheduling without lookahead

We have seen in the previous discussion that we can obtain optimality if the adversary is not unlimited in its power to spring surprises on us. The crucial point is that we can arrange machines to be shared perfectly between jobs at certain well-known times with a finite number of preemptions. As shown in the example of Lemma 2.3.5, we will not be able to exactly share machines exactly if these times are not known. In particular, all three parts of Lemma 2.3.11 are no longer true if we cannot guarantee that our schedule runs unmodified and uninterrupted in the time interval $[t, t+v[$.

In this section, we show that we can still bound the loss in accuracy when we do not have this advance knowledge. Algorithm 2.5 below is almost identical to Algorithm 2.4, however the proof of correctness is more involved.

Algorithm 2.5: Speed-adaptive LAFM (SALAFM)

Input: Accuracy parameter δ
Upon *job release or machine failure/join* **do**
  Preempt all jobs currently running;
  Let g the next time a job is released;
  Call Algorithm 2.3 with the currently available jobs and machines and $v_{\max} = \min\{\delta/s_1, g\}$;

The result we achieve is the following:

Theorem 2.4.1. *The makespan C^{SALAFM} returned by Algorithm 2.5 is bounded by $C^{\text{OPT}} + n(n+3)\delta/2$, where n is the number of jobs scheduled and $\delta > 0$ is an arbitrary constant.*

Corollary 2.4.2. If the number of jobs n is known in advance, the additive error can be made arbitrarily small by setting $\delta := 2\epsilon/(n^2+3n)$.

To prove Theorem 2.4.1, we have to take into account that when the schedule is interrupted, the machines were not shared exactly as needed. We can, however, still show the following somewhat weaker result:

Lemma 2.4.3. *Let q_i, q_j the remaining processing times of two jobs at the time the schedule was last extended, v the length returned by Algorithm 2.3, and q_i', q_j' the*

processing times at the point the schedule was interrupted. Then,

$$|q'_i - q'_j| \leq \max\{\delta, |q_i - q_j|\}\,, \tag{2.36}$$

and also

$$q_i - q'_i \leq \delta\,. \tag{2.37}$$

Proof. Let us assume without loss of generality that $q_i \geq q_j$, and if $q_i = q_j$, that $q'_i \geq q'_j$. The claim (2.37) is true by definition of $v_{\max}$, since $q'_i \geq q_i - v_{\max} s_1 \geq q_i - \delta$.

As to (2.36), the claim is immediately true if $q_i = q_j$ since $q'_i \leq q_i$, so $q'_i - q'_j \leq q_i - q'_j \leq q_i - q_j + \delta = \delta$.

If $q_i > q_j$, then the jobs belong to different clusters, and by (2.21), at any point in time, the machine that executes job i is at least as fast as the one executing job j, so $q'_i - q'_j \leq q_i - q_j$. It remains to show that $q'_i - q'_j \geq -\delta$, but this is obvious since $q'_i - q'_j \geq (q_i - \delta) - q_j = (q_i - q_j) - \delta$ and $q_i \geq q_j$. □

Based on this lemma, it will prove helpful to relax our notion of clusters in the following way:

Definition 2.4.4. Let $q_1 \geq \cdots \geq q_n \geq 0$. For $\delta \geq 0$, the *partition into δ-relaxed clusters* $C_1, \ldots, C_k$ is defined by

$$C_1 = \{q_1, \ldots, q_{\sigma_1}\}, C_2 = \{q_{\sigma_1+1}, \ldots, q_{\sigma_2}\}, \ldots$$

such that $|q_{i+1} - q_i| > \delta$ if and only if $i \in \{\sigma_1, \sigma_2, \ldots\}$.

Equivalently, we can define that two values q_i and q_{i+1} are *closely adjacent* if $q_{i+1} - q_i \leq \delta$. The classes of the transitive reflexive symmetric closure of this relation are then exactly the δ-relaxed clusters.

The following properties will help bound the error the algorithm accumulates by only tracking δ-relaxed clusters.

Lemma 2.4.5. *If C is a δ-relaxed cluster, then*

$$\max C - \min C \leq (|C| - 1)\delta \tag{2.38}$$

and

$$\sum C \leq |C| \min C + \frac{(|C|-1)|C|}{2}\delta\,. \tag{2.39}$$

Proof. Let $\sigma = |C|$ and $q_1 \geq \ldots \geq q_\sigma$ the elements of C. In particular, $\max C = q_1$ and $\min C = q_\sigma$. We observe that by definition, $q_{\sigma-i} \leq \min C + i\delta$ for $i \in \{0,\ldots,|C|-1\}$, i.e. $q_\sigma \leq \min C$, $q_{\sigma-1} \leq \min C + \delta$ and so on. In particular, we obtain

$$\max C - \min C \;=\; q_1 - \min C \;\leq\; \min C + (\sigma-1)\delta - \min C \;=\; (|C|-1)\delta\,,$$

which proves (2.38), and

$$\begin{aligned}\sum C = \sum_{i=0}^{\sigma-1} q_{\sigma-i} &\leq \sum_{i=0}^{\sigma-1}(\min C + i\delta) \\ &= \sigma \min C + \delta \sum_{i=0}^{\sigma-1} i = |C| \min C + \delta\frac{|C|(|C|-1)}{2}\,,\end{aligned}$$

which shows (2.39). □

Recall that we introduced a dummy job in our algorithm that already starts off with remaining processing time 0. This job now becomes crucial, because its existence implies

$$\min C > 0 \implies \min C > \delta \tag{2.40}$$

for all δ-relaxed clusters C.

By Lemma 2.4.3, it is easy to see that over the course of time, Algorithm 2.5 will merge, but never split δ-relaxed clusters. Using the same notation as for Lemma 2.3.12, but denoting with $\sigma_1,\ldots$ the boundaries of δ-relaxed clusters instead of clusters, we can show:

Lemma 2.4.6. *Suppose that*

$$\sum_{j=1}^{\sigma_i} q_j \leq \sum_{j=1}^{\sigma_i} q_j^* \tag{2.41}$$

for all $i \in \{1,\ldots,k\}$ such that $\min C_i > 0$ and let $s_1 \geq \ldots \geq s_n$ the machine speeds, adding dummy machines of speed 0 if needed. Then at any time $\tau \in [t, t+v]$, it

holds that

$$\sum_{j=1}^{\sigma_i} q'_j \leq \sum_{j=1}^{\sigma_i} q^{*\prime}_j \,. \tag{2.42}$$

Proof. The proof is identical to that of Lemma 2.3.12, the essence being again that the longest σ_i jobs will get exclusive use of the fastest σ_i machines, and no algorithm can put more processing power into them. □

To prove Theorem 2.4.1, we again proceed inductively. By Lemma 2.4.6 and Lemma 2.3.13, the invariant (2.41) is maintained at all points when Algorithm 2.3 is called as long as $\min C_i > 0$. It remains to consider the final stages of the algorithm's execution. Let t the last time that subroutine Algorithm 2.3 was called such that $\min C_1 > 0$. (Recall that C_1 is the δ-relaxed cluster with the longest jobs.) By Lemma 2.4.6, $t \leq \mathrm{OPT}$, because every algorithm must have at least $\min C_1 > 0$ in total remaining processing time. As observed above, we then have $\min C_1 > \delta$. By Lemma 2.4.3 and maximality of t, $\min C_1 \leq 2\delta$. In particular, all q_i that are not in C_1 are bounded from above by $\min C_1 - \delta \leq \delta$. We can now bound the sum of remaining processing times as

$$\begin{aligned}
\sum_{j=1}^{n} q_j &= \sum_{j=1}^{\sigma_1} q_j + \sum_{j=\sigma_1+1}^{n} q_j \\
&\leq \sigma_1(2\delta) + \frac{\sigma_1(\sigma_1-1)}{2}\delta + (n-\sigma_1)\delta \\
&= (\sigma_1 + n + \frac{\sigma_1(\sigma_1-1)}{2})\delta \\
&\leq \frac{n(n+3)}{2}\delta
\end{aligned} \tag{2.43}$$

by using (2.39) and $\sigma_1 \leq n$, which proves Theorem 2.4.1 since we are guaranteed a machine of speed 1 at all times.

3 Offline Scheduling on Unrelated Machines

3.1 Introduction and historical remarks

In this chapter, we study scheduling problems on *unrelated* machines. In this setting, the execution time of every job may vary arbitrarily from machine to machine, i.e. the instance is given by an arbitrary function

$$p : \{1, \dots, m\} \times \{1, \dots, n\} \to \mathbb{N}, (i, j) \mapsto p_{ij} \,. \tag{3.1}$$

This lack of structure makes the problem notoriously difficult to handle: every job can draw from the set of machines a subset which it likes or dislikes. Since the number of such subsets is exponential in the number of machines, it is difficult to group different jobs together to reduce the state space if there is no constant bound on the number of machines. (Conversely, if the number of machines is assumed constant, such approaches work very well, as noted below.) In fact, unterstanding general makespan minization on unrelated machines is generally considered one of the open puzzles [Woe02, Open problem 4] in scheduling theory.

From the results that are known so far, it appears that this set structure is the main difficulty posed by this problem: the strongest inapproximability result, the nonexistance of a $3/2 - \epsilon$ approximation for any $\epsilon > 0$ unless P = NP, can already be achieved if all job lengths are drawn from the set $\{1, 2\}$ as shown by Lenstra, Shmoys and Tardos [LST90]. In the same paper, they present an approximation algorithm for $R \mid\mid C_{\max}$ with additive error $\max_{i,j} p_{ij}$, in particular, this can easily be turned into a 2-approximation. This is still asymptotically the best bound known today. The only improvement in the general case was given by Shchepin and

[Woe02] G. J. Woeginger. Open problems in the theory of scheduling. *Bulletin of the EATCS*, 2002.

[LST90] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 1990.

Vakhania [SV05], who reduce the additive error to $\frac{m-1}{m} \cdot \max_{i,j} p_{ij}$ and hence the multiplicative error to $2 - 1/m$.

One important setting that is considered to gain more insight into approximability of the set structure is that of *assignment restrictions*: it is often the case that the time a job takes to perform does not so much depend on some aptitude of the worker who performs it, as in the general unrelated scheduling problem, but on a binary criterion: a worker is qualified to do a certain job (for example, he has received the appropriate training), or he isn't, but the job will take any qualified worker the same amount of time. Hence, we can define

$$p_{ij} = \begin{cases} p_j & \text{if machine } i \text{ is capable of performing job } j, \\ \infty & \text{otherwise.} \end{cases} \tag{3.2}$$

where p_j is the length of the job j.

We denote with $M|_j$ the set $\{i : i \in \{1, \ldots, m\}, p_{ij} < \infty\}$ of machines on which job j is admissible; correspondingly, we refer to this problem as $\mathrm{P} \mid M_j \mid C_{\max}$ in three-field notation.

Despite removing almost all complexity due to job lengths, this problem is still hard: as Ebenlendr et al. [EKS08] have shown, there is no $3/2 - \epsilon$-approximation for Scheduling with Assignment Restrictions unless P = NP, even when for every job j, $|M|_j| \leq 2$ and all p_j are drawn from $\{1, 2\}$.

On the positive side, Ebenlendr et al. give a 7/4-approximation for this case. If the assignment graph additionally is a tree, Lee et al. [LLP09] give a fully polynomial approximation scheme.

Also, results exist on other additional constraints on the structure of the sets M_j: Ou et al. [OLL08] have given a PTAS for the case that the machine sets are

[SV05] E. V. Shchepin and N. Vakhania. An optimal rounding gives a better approximation for scheduling unrelated machines. *Operations Research Letters*, 2005.

[EKS08] T. Ebenlendr, M. Krcal, and J. Sgall. Graph balancing: a special case of scheduling unrelated parallel machines. In *Proc. SODA*, 2008.

[LLP09] K. Lee, J. Y.-T. Leung, and M. Pinedo. A note on graph balancing problems with restrictions. *Information Processing Letters*, 2009.

[OLL08] J. Ou, J. Y.-T. Leung, and C.-L. Li. Scheduling parallel machines with inclusive processing

totally ordered by $\subseteq$ improving over a previous 3/2-approximation by Glass and Kellerer [GK07]; this PTAS was improved to include non-zero release times of the jobs by Li and Wang [LW10]. For the strictly more general case of *nested* sets, i.e. $M_j \cap M_{j'} \in \{M_j, M_{j'}, \emptyset\}$, Huo and Leung [HL10] give a 7/4-approximation.

In contrast, if all jobs have the same length, the problem can be solved exactly in polynomial time, even for arbitrary M_j. One such algorithm based on matching techniques is given by Lin and Li [LL04], but the result as such can already be concluded from the result by Shchepin and Vakhania, or in fact, as we show in Theorem 3.2.6 below, from the result by Lenstra, Shmoys and Tardos and Plotkin, Shmoys and Tardos [PST95].

The situation is much easier if the number of machines m is assumed constant: here, classical scaling and dynamic programming approaches yield fully-polynomial time approximation schemes; the currently fastest result for Rm || $C_{\max}$ being an FPTAS with running time $\mathcal{O}(n) + 2^{\mathcal{O}(m \log(m/\epsilon))}$ by Jansen and Mastrolilli [JM09].

3.2 Scheduling with Assignment Restrictions

In the following, we will concentrate on a special structure of the machines, namely that of *interval* assignment restrictions.

In the setting with interval assignment restrictions, there exists a permutation π of the machines such that for every job j, there are values α_j and ω_j such that

set restrictions. *Naval Research Logistics*, 2008.

[GK07] C. A. Glass and H. Kellerer. Parallel machine scheduling with job assignment restrictions. *Naval Research Logistics*, 2007.

[LW10] C.-L. Li and X. Wang. Scheduling parallel machines with inclusive processing set restrictions and job release times. *European Journal of Operational Research*, 2010.

[HL10] Y. Huo and J. Y.-T. Leung. Parallel machine scheduling with nested processing set restrictions. *European Journal of Operational Research*, 2010.

[LL04] Y. Lin and W. Li. Parallel machine scheduling of machine-dependent jobs with unit-length. *European Journal of Operational Research*, 2004.

[PST95] S. A. Plotkin, D. B. Shmoys, and E. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 1995.

[JM09] K. Jansen and M. Mastrolilli. Scheduling unrelated parallel machines: linear programming strikes back. Submitted, 2009.

$M|_j = \{\alpha_j, \ldots, \omega_j - 1\}$.

First of all, we mention for completeness the following result which is implicit in a result first proven by Booth and Lueker [BL76]:

Lemma 3.2.1. *Given the family of sets $\{M|_j : j \in \{1, \ldots, n\}\}$, we can find in linear time a matching permutation π and the values α_j, ω_j.*

Hence, we will in the following always assume that the machines are already re-ordered such that π is the identity permutation, and that the intervals are given by their endpoints, i.e. consider an instance I of the Scheduling with Interval Assignment Restrictions problem given as a list of n tuples

$$(p_j, \alpha_j, \omega_j) \in \mathbb{N} \times \{1, \ldots, m\} \times \{2, \ldots, m+1\}; j = 1, \ldots, n$$

signifying that the jth job has length p_j and is admissible on machines $[\alpha_j, \omega_j) = \{\alpha_j, \alpha_j + 1, \ldots, \omega_j - 1\}$. Clearly, we may assume without loss of generality that every machine is useful in the sense that at least one job is admissible on it; in particular, we have $m = \max\{\omega_j : j = 1, \ldots, n\} - 1$ and $1 = \min\{\omega_j : j = 1, \ldots, n\}$.

We will often need a relaxation of the intervals given by the instance: we call a subset $[k, l) = \{k, k+1, \ldots, l-1\}$ of the machines a *pseudo-interval* if $\{k, l\} \subseteq \{\alpha_j, \omega_j : j = 1, \ldots, n\}$. Note that for notational simplicity we do not require k to be an α value or l an ω value, and that the number of pseudo-intervals in the instance is bounded by $(2n)^2 \in \mathcal{O}(n^2)$. Given some subset X of the jobs, we can define for every subset M' of the machines the set

$$X|_{M'} := \{j \in X : M|_j \subseteq M'\}, \tag{3.3}$$

the set of jobs in X that is not admissible anywhere outside M'. For intervals, we will write

$$X|_{[k,l)} := \{j \in X : k \leq \alpha_j \wedge \omega_j \leq l\}. \tag{3.4}$$

As a simple but important insight, we note

[BL76] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *Journal of Computer and System Sciences*, 1976.

Remark 3.2.2. For any non-negative function $v : J \to \mathbb{R}$, the aggregated function

$$f : 2^M \to \mathbb{N} : M' \mapsto \sum_{M|_j \subseteq M'} v(j) \tag{3.5}$$

is *supermodular*, i.e. for two sets $A, B \subseteq M$, we have

$$f(A \cup B) \geq f(A) + f(B) - f(A \cap B)\,. \tag{3.6}$$

In the same way as Scheduling with Assignment Restrictions, Scheduling with Interval Assignment Restrictions is strongly NP-hard as a generalization of scheduling on identical machines P $||$ $C_{\max}$, thus it does not allow an FPTAS. Horowitz and Sahni [HS76] have shown that there exists a PTAS for P $||$ $C_{\max}$. In Subsection 3.2.2, we will show that a PTAS is still possible for some special cases; however, proving or disproving the existence of a PTAS for Scheduling with Interval Assignment Restrictions is beyond the scope of this work.

In the encoding scheme given above, the instance size is not polynomial in the number of machines. Hence, we first show that the number of machines is, without loss of generality, polynomially bounded:

Lemma 3.2.3. *Given an instance I, we can find in time polynomial in n at most n disjoint subintervals of $\{1, \ldots, m\}$ such that each subinterval has length polynomial in n and it is neccessary and sufficient to solve the subinstances induced by the subintervals.*

Proof. Consider the list of endpoints of intervals given by the jobs in I sorted in ascending order $1 = a_1 \leq \ldots \leq a_{2n} = m + 1$, and the at most $2n + 1$ non-empty minimal pseudo-intervals $[a_k, a_{k+1})$ induced by them. We can find in time polynomial in n the number of jobs $n_k \leq n$ that are feasible in the interval $[a_k, a_{k+1})$. For every k such that the length of the interval $a_{k+1} - a_k \geq n_k$, we can match the machines to the jobs in an arbitrary way. Since every machine gets only one job, the load of these machines is always bounded by OPT, and the rest of the instance splits in a natural way into two indepedendent subinstances on the machines

[HS76] E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM*, 1976.

$\{1,\dots,a_k-1\}$ and $\{a_{k+1},\dots,m\}$. After iterating this $\mathcal{O}(n)$ times, all remaining pseudo-intervals have $0 \le a_{k+1}-a_k < n_k \le n$, which means the total number of machines in all of them is at most $\mathcal{O}(n^2)$. □

As a further general result that will prove useful for all algorithms presented here, we show that the crucial aspect of the problem is the assignment of *job sizes* to machines, and the jobs themselves are then easily scheduled by the algorithm 'Least Flexible First' given as Algorithm 3.1. The following two lemmas are a generalization of a technical result given by Lin and Li [LL04].

Lemma 3.2.4. *Let X be a set of jobs of the same size, and $a : \{1,\dots,m\} \to \mathbb{N}$ a function. The following statements are equivalent:*

1. *The* Least Flexible First *heuristic generates a feasible schedule that assigns at most $a(i)$ jobs to machine i, for $i \in \{1,\dots,m\}$.*
2. *There is a feasible schedule that assigns at most $a(i)$ jobs to machine i, for $i \in \{1,\dots,m\}$.*
3. *For all pseudo-intervals $[k,l)$, it holds that*

$$\sum_{i\in[k,l)} a(i) \ge \left|X|_{[k,l)}\right|. \tag{3.7}$$

Algorithm 3.1: Algorithm 'Least Flexible First'

Input: Function a, set X of jobs
Output: A schedule of X
for $i = 1,\dots,m$ **do**
  Let $A := \{j : \alpha_j \ge i, \omega_j < i, j \text{ unscheduled}\}$ be the set of available jobs;
  if $|A| \le a(i)$ **then**
    Schedule all jobs in A on machine i;
  else
    Sort A by non-decreasing ω_j values;
    Schedule the first $a(i)$ jobs from the sorted list on machine i;

Proof. The implication '1 $\implies$ 2' is trivially true. If 2 holds, certainly all jobs of $X|_{[k,l)}$ are scheduled within $[k,l)$ for all pseudo-intervals $[k,l)$, so 3 holds and '2 $\implies$ 3' is true.

Assume now for sake of contradiction that 3 holds for some X and a, but 1 fails. This means that there is some machine l such that machines $\{1,\dots,l-1\}$ have been feasibly assigned to, but there remains some job j with $\omega_j = l$ that is not yet scheduled. We will iteratively prove the existance of a machine $k \leq \alpha_j$ such that all jobs scheduled on $[k,l)$ belong to $X|_{[k,l)}$ and all machines $i \in [k,l)$ have $a(i)$ jobs, i.e. they are full, which contradicts 3 since j also belongs to $X|_{[k,l)}$.

In our first iteration, let us start with $k = \alpha_j$. By design of the algorithm, all jobs j' scheduled in $[k,l)$ have $\omega_{j'} \leq l$, because otherwise j would have been scheduled in their place. Consider now the value $k' = \min\{\alpha_{j'} : j' \text{ scheduled in } [k,l)\}$. Two cases can occur: if $k' \geq k$, all jobs in $[k,l)$ belong to $X|_{[k,l)}$ and we are done. Otherwise, we have $k' < k$. Then, all machines in $[k',k)$ are full, because otherwise, some job j' with $\alpha_{j'} = k'$ would not have been put into the interval $[k,l)$. For all jobs j'' in $[k',k)$, we also know by design of the algorithm that $\omega_{j''} \leq \omega_{j'} \leq \omega_j$. We now set $k := k'$ and iterate. After at most m iterations, the process terminates with no job in the interval $[k,l)$ being admissible outside $[k,l)$. □

In other words, Lemma 3.2.4 states that in the case of intervals, the neccessary and sufficient condition of Hall's Marriage Theorem can be checked in polynomial time since all constraints not corresponding to intervals are redundant, and also that a matching of the jobs to the spaces on the machines can easily be found. Clearly, we can generate a feasible schedule for an entire instance by using Lemma 3.2.4 for all sizes simultaneously.

If we allow *fractional* schedules, where a job might be partially executed on more than one machine, we can essentially replace the distinct jobs by their total area and consider the limit as the allowed size of the fragments approaches 0. Then, we obtain:

Corollary 3.2.5. Let X be a subset of jobs and $a : \{1,\dots,m\} \to \mathbb{R}$ a function. The following are equivalent:

1. The fractional *Least Flexible First* heuristic generates a feasible fractional schedule that assigns at most load $a(i)$ to machine i for all $i \in \{1,\dots,m\}$.

2. There is a feasible fractional schedule that assigns at most load $a(i)$ to machine i for all $i \in \{1, \dots, m\}$.

3. For all pseudo-intervals $[k, l)$, it holds that

$$\sum_{i \in [k,l)} a(i) \geq \sum_{j \in X|_{[k,l)}} p_j \,. \tag{3.8}$$

Finally, before we turn to the details of the algorithms for the specific cases, we note that in all cases the optimal makespan is bounded from above by $n \cdot \max\{p_j : j \in \{1, \dots, n\}\}$, i.e. it is pseudopolynomial in the input size, and since the p_j are integral, so is the optimal makespan, so it is sufficient to give relaxed decision algorithms to obtain approximation algorithms by Theorem 1.1.7.

3.2.1 A $(2 - 2/p_{\max})$-approximation for assignment on intervals

In the following, we will give intermediate results that serve to highlight the techniques used to obtain our overall results. To start off, we show

Theorem 3.2.6. *There is a $(2 - 1/p_{\max})$-approximation for Scheduling on Unrelated Machines.*

We should point out that the approximation ratio for general unrelated machine scheduling in Theorem 3.2.6 was previously obtained by Gairing et al. [GLMM04]; however, their proof does not rely on a linear programming formulation. The result itself, albeit being a simple extension of the classical approach, appears not to be widely known, for example, it is mentioned as an open question in [BM10].

Proof. Denote with $p_{ij} \in \{1, \dots, p_{\max}\}$ the processing time of job j on machine i. By Theorem 1.1.7, we may assume that we there exists a schedule of length C. In

[GLMM04] M. Gairing, T. Lücking, M. Mavronicolas, and B. Monien. Computing nash equilibria for scheduling on restricted parallel links. In *Proc. STOC*, 2004.

[BM10] P. Biró and E. McDermid. Matching with sizes (or scheduling with processing set restrictions). Technical Report TR-2010-307, University of Glasgow, 2010.

particular, this means that the LP relaxation of Lenstra, Shmoys and Tardos [LST90]

$$\begin{aligned} \sum_{j:p_{ij}\leq C} y_{ij}p_{ij} &\leq C \quad \forall i \in \{1,\dots,m\} \\ \sum_{i:p_{ij}\leq C} y_{ij} &\geq 1 \quad \forall j \in \{1,\dots,n\} \\ y_{ij} &\geq 0 \quad \forall i \in \{1,\dots,m\}, j \in \{1,\dots,n\} \end{aligned} \tag{3.9}$$

has a solution. Since this formulation only involves variables with $p_{ij} \leq C$, we may assume $p_{\max} \leq C$. Any such solution y^* induces a residual bipartite fractional assignment graph as follows: the vertex set consists of the disjoint union of jobs and machines, and there is an edge of weight y^*_{ij} from job j to machine i iff $y^*_{ij} \in]0,1[$. Note that we can discard from this graph all jobs that are already integrally assigned. Then, the total weight of all edges incident to a job equals 1. Using the standard cyclic shifting argument of Plotkin, Shmoys and Tardos [PST95], we can modify the solution y^* so that this assignment graph is a forest without violating feasibility.

It is easy to see that in the resulting forest, all leaves are machines: since the total weight of incident edges is still 1 for every job, it is either integrally matched to a machine or non-integrally matched to at least two machines.

The rounding can then be done by iteratively assigning to every leaf machine the one job it is connected to and then removing both machine and job. The resulting ratio then follows from the following claim:

$$\sum_{j:y^*_{ij}=1} p_{ij} \leq C - 1 \tag{3.10}$$

for all leaf machines i. Assume this were not the case, i.e. there is some leaf machine i with $\sum_{j:y^*_{ij}=1} p_{ij} > C - 1$. Since the p_{ij} are integral, this means the sum is at least $C - 1 + 1 = C$. However, there is also one j' with $1 > y^*_{ij'} > 0$ since i is a leaf machine. Since all processing times are positive, this means the total load of machine i is then $\sum_{j:y^*_{ij}=1} p_{ij} + y^*_{ij'}p_{ij'} > C$, in contradiction to the fact that y^* solves the LP (3.9). With this bound, we obtain that after rounding, the load of every machine is bounded by $C - 1 + p_{\max}$. Since $p_{\max} \leq C$, the approximation

ratio of the algorithm is then at worst

$$\frac{C-1+p_{\max}}{C} = \frac{C+p_{\max}(1-1/p_{\max})}{C} \leq \frac{C+C(1-1/p_{\max})}{C} = 2-1/p_{\max}, \tag{3.11}$$

as claimed. □

As we can observe, the critical point to obtain better approximation bounds is to find an assignment of jobs which have size close to $p_{\max}$. As a first step, we show that this is possible if such large jobs have a special structure:

Theorem 3.2.7. *There is a $2-2/p_{\max}$-approximation for Scheduling with Interval Assignment Restrictions.*

We devote the rest of the subsection to the proof of Theorem 3.2.7, combining the rounding technique of Theorem 3.2.6 with a two-step approximation. Again, assume we are given C such that a schedule of length C exists. Note that in the assignment case, we can immediately assume $C \geq p_{\max} = \max_{j=1,\dots,n} p_j$ since the length of a job is fixed across its feasible machines. Denote with J_l the set of 'large' jobs of length $p_{\max}$ and with J_s the set of all other, 'small', jobs. By Lemma 3.2.4, we have for each pseudo-interval $[k,\ell)$ a lower bound on the number of jobs of J_l that are assigned within this interval. On the other hand, we know that at most $\lfloor C/p_{\max} \rfloor$ of these jobs can be on a single machine. Finally, there is also an upper bound given by all the jobs in J_s by Corollary 3.2.5. In total, we can give the following IP formulation, where x_i counts the number of jobs of size $p_{\max}$ assigned to machine i:

$$\begin{aligned}
\sum_{i=k}^{\ell-1} x_i &\geq \left|J_l|_{[k,l)}\right| \quad \forall k,\ell \in \{1,\dots,m+1\} \\
\sum_{i=k}^{\ell-1} x_i &\leq \frac{l-k-\sum_{j\in J_s|_{[k,l)}} p_j}{p_{\max}} \quad \forall k,\ell \in \{1,\dots,m+1\} \\
x_i &\leq \lfloor C/p_{\max} \rfloor \quad \forall i \in \{1,\dots,m\} \\
x_i &\in \mathbb{N} \quad \forall i \in \{1,\dots,m\}
\end{aligned} \tag{3.12}$$

Clearly, solutions exist because one integral solution is induced by the unknown schedule of length C. We can find some feasible fractional solution, which will

yield an integral assignment of large jobs:

Lemma 3.2.8. *All basic solutions of the LP relaxation of* (3.12) *are integral.*

To show this, we recall a classical result about totally unimodular matrices:

Theorem 3.2.9 (Ghouila-Houri [GH62])**.** *A matrix $A \in \{-1,0,1\}^{n\times m}$ with column vectors $A^{(1)},\dots,A^{(m)}$ is totally unimodular if and only if every subset $S \subseteq \{1,\dots,m\}$ can be partitioned into S^+ and S^- such that*

$$\sum_{i\in S^+} A^{(i)} - \sum_{i\in S^-} A^{(i)} \in \{-1,0,1\}^n . \tag{3.13}$$

Proof of Lemma 3.2.8. Let S a subset of the columns of (3.12). Note that by design of the LP, S then corresponds to a subset of machines, which we also denote with S. Denote the elements of $S = \{a_1,\dots,a_{|S|}\}$ with $a_1 < a_2 < \cdots < a_{|S|}$.

We set $S^+ := \{a_i : i \in \{1,\dots,|S|\}, i \text{ odd}\}$ and $S^- := \{a_i : i \in \{1,\dots,|S|\}, i \text{ even}\}$ and claim that this partition satisfies the condition of Theorem 3.2.9. To see this, consider an arbitrary pseudo-interval $[k,l)$ and the two corresponding constraints in (3.12). If $|S\cap[k,l)|$ is even, then by design of S^+ and S^-, $|S^+\cap[k,l)| = |S^-\cap[k,l)|$. If $|S\cap[k,l)|$ is odd, then either $\min(S\cap[k,l))$ is odd and $|S^+\cap[k,l)| = |S^-\cap[k,l)|+1$, or it is even and $|S^+\cap[k,l)| = |S^-\cap[k,l)|-1$. In either case, since the coefficients in the constraints are all 1 or all -1, we obtain that

$$\Big\{ \sum_{i\in S^+\cap[k,l)} 1 - \sum_{i\in S^+\cap[k,l)} 1,\ \sum_{i\in S^+\cap[k,l)} (-1) - \sum_{i\in S^+\cap[k,l)} (-1) \Big\} \subset \{-1,0,1\} . \tag{3.14}$$

Finally note that for the constraints $x_i \le \lceil C/p_{\max}\rceil$, the sum will trivially be 0, 1 or -1 for $i \notin S$, $i \in S^+$, and $i \in S^-$, respectively. The claim now follows from Theorem 3.2.9. □

Hence, we can obtain in polynomial time an integral solution x^* to (3.12). Based on this assignment of large jobs, we can now formulate a Lenstra/Shmoys/Tardos-

[GH62] A. Ghouila-Houri. Characterisation des matrices totalement unimodulaires. *Comptes Rendus de l'Académie des sciences*, 1962.

style LP for the remaining jobs:

$$\begin{aligned} \sum_{j \in J_s} y_{ij} p_j &\leq C - x_i^* p_{\max} \quad \forall i \in \{1, \dots, m\} \\ \sum_{i=\alpha_j}^{\omega_j - 1} y_{ij} &\geq 1 \quad \forall j \in J_s \\ y_{ij} &\geq 0 \quad \forall i \in \{1, \dots, m\}, j \in J_s \end{aligned} \tag{3.15}$$

It is not obvious that this formulation has any connection to the original problem, since we might not have the 'right' assignment of the jobs in J_l. The following claim shows that any assignment is 'good enough'.

Lemma 3.2.10. *If x^* is a feasible solution to* (3.12)*, then there is a feasible solution y^* to* (3.15)*, and x^*, y^* induce a fractional schedule of length at most C.*

Proof. Assume that there is no feasible solution to (3.15). We now consider the following relaxation of the LP:

$$\begin{aligned} \sum_{j \in J_s} y_{ij} p_j &\leq C - x_i^* p_{\max} + \lambda \quad \forall i \in \{1, \dots, m\} \\ \sum_{i=\alpha_j}^{\omega_j - 1} y_{ij} &\geq 1 \quad \forall j \in J_s \\ y_{ij} &\geq 0 \quad \forall i \in \{1, \dots, m\}, j \in J_s \end{aligned} \tag{3.16}$$

which allows us to exceed the load bound on all machines by λ. Clearly, for sufficiently large λ, e.g. $\lambda = C$, (3.16) will have feasible solutions. In the following, let $\lambda > 0$ minimal such that feasible solutions exist, and fix a feasible solution that minimizes the number of machines for which the load constraint becomes tight.

Let i a machine with tight load constraint, and set $L = \{i\}$. We will iteratively grow L, but maintain the following properties:

1. L is an interval.
2. All machines in L have a tight load constraint.

Clearly, both properties hold for $L = \{i\}$. Suppose now that there is a job j such that $\sum\{x_{ij} : i \in L\} > 0$ and $\emptyset \neq L \cap [\alpha_j, \omega_j) \neq [\alpha_j, \omega_j)$. Then, we will set

$L' := L \cup [\alpha_j, \omega_j)$. Since the union of two non-disjoint intervals is again a single interval, the new set L' still maintains property 1.

Assume that one of the newly-added machines i' does not have a tight load constraint, and let $i'' \in L$ with $x_{i''j} > 0$. By induction, i'' has a tight load constraint. By increasing $x_{i'j}$ and decreasing $x_{i''j}$ by a sufficiently small amount, both i' and i'' will get an untight load constraint, which violates the assumption that we already chose a solution with a minimal number of tight constraints. Hence, all machines in L' will have a tight load constraint, so L' still has property 2. In total, we can set $L := L'$ and iterate.

After at most $|J_s| \leq n$ iterations, this process terminates, and all jobs in J_s that are at least partially assigned inside L are assigned totally inside L. In particular, this means that

$$\sum_{j \in J_s : [\alpha_j, \omega_j) \subseteq L} p_j + \sum_{i \in L} x_i^* p_{\max} = |L| \cdot (C + \lambda) > |L| \cdot C\,, \tag{3.17}$$

which means that the interval L violated its upper bound in (3.12), which is a contradiction. □

We can now prove Theorem 3.2.7 by proceeding in the same way as in the proof of Theorem 3.2.6: we turn the fractional assignment y^* into a forest and assign jobs to leaf machines. Note, however, that the maximum size of a fractional job is at most $p_{\max} - 1$, since larger jobs are already assigned integrally. This means the total length of the resulting schedule is at most $C + p_{\max} - 2$, and we then obtain a ratio of

$$\frac{C + p_{\max} - 2}{C} = \frac{C + p_{\max}(1 - 2/p_{\max})}{C} \leq 2 - 2/p_{\max}\,, \tag{3.18}$$

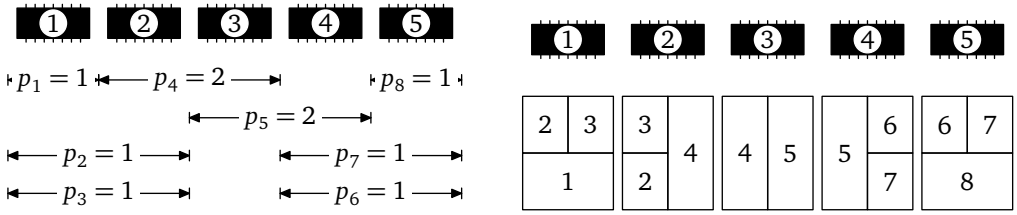
as claimed.

Noting that we only use the bound

$$p_{2\max} := \max\{p_j : j \in \{1, \ldots, n\}, p_j \neq p_{\max}\} \leq p_{\max} - 1\,, \tag{3.19}$$

we even obtain the following result which is stronger in special cases:

Corollary 3.2.11. There is an algorithm for Restricted Assignment on Intervals with additive error at most $p_{2\max} - 1$.



(a) Jobs of the instance

(b) A fractional schedule of length 2

Figure 3.1: An instance with integrality gap 3/2.

Corollary 3.2.12. There is a polynomial-time exact algorithm for Restricted Assignment on Intervals if all $p_j \in \{p, 1\}$ for some $p \in \mathbb{N}$.

The next logical step towards better approximation results would be to consider a stronger LP formulation. However, the canonical choice, a *configuration LP*, already introduces a large gap in very limited circumstances:

Example 3.2.13. Consider an instance consisting of eight jobs, all of which draw their lengths from $\{1, 2\}$, as shown in Figure 3.1. Note that by Corollary 3.2.12, such instances can be solved exactly in polynomial time. It is easily verified that there is no feasible schedule of length 2 since either the set of jobs $\{1, 2, 3, 4\}$ of total length 5 must be assigned entirely to machines 1 and 2 or else the set of jobs $\{5, 6, 7, 8\}$ must be assigned entirely to machines 4 and 5.

However, in the configuration ILP for makespan 2, there are only two configurations: one large job, or two small jobs. An LP relaxation may mix both on a single machine and will arrive at a feasible fractional assignment of jobs as shown in Figure 3.1.

Hence, we will in the following concentrate on cases in which dynamic programming approaches can be successfully used to obtain approximation ratios better than 3/2.

3.2.2 Approximation schemes for special interval structures

In this section, we will show that two more restricted settings admit a polynomial-time approximation scheme. More precisely, we show:

Theorem 3.2.14. *There is a PTAS for Scheduling with Interval Assignment Restric-*

tions if for any two jobs j, j', we have

$$\alpha_j < \alpha_{j'} \implies (\omega_j \geq \omega_{j'} \vee \omega_j \leq \alpha_{j'}). \tag{3.20}$$

(We call such an instance nested*.)*

Theorem 3.2.15. *There is a PTAS for Scheduling with Interval Assignment Restrictions if for any two jobs j, j', we have*

$$\alpha_j < \alpha_{j'} \implies \omega_j \geq \omega_{j'}. \tag{3.21}$$

(We call such an instance compatible*.)*

Especially the former setting occurs very naturally in practical applications, if we consider a system of "specialisations": over time, a (human) worker will acquire deeper skills in the field he works in, and will tend to specialise on a certain sub-field. This does not mean, of course, that he suddenly forgets everything he has previously learned, but he will probably not become more proficient outside his most specialized field. Assuming that there will usually be several fields that one can focus on, there is a natural branching process, where nodes of the tree represent skills, and the path from the root to a node represents the education needed to obtain this skill. Fixing an arbitrary order of the children of each node, it is easy to see that the workers can be arranged in a way that they have nested interval assignment restrictions.

In building up to Theorem 3.2.14 and Theorem 3.2.15, we will first present a dynamic programming formulation for the problem $\mathrm{P} \mid\mid C_{\max}$. This formulation is slightly weaker than the one presented by Hochbaum and Shmoys for identical machines [HS87] in terms of the running time that can be achieved, but as we demonstrate, it can be extended to some kinds of interval assignment restrictions. In the following, we consider a fixed, but arbitrary accuracy ϵ; our algorithms will generate a schedule of length at most $(1+4\epsilon)\mathrm{OPT}$. Similar to above, we can perform binary search over the interval $[0, n \max_j p_j]$, so it is sufficient to find a

[HS87] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 1987.

schedule of length $(1+4\epsilon)C$ provided that a schedule of length C exists, where C is the guessed makespan.

Recall that we denote with p_j the length of job j. Our dynamic program will work on an instance I' with rounded job lengths p'_j that are defined as follows:

$$p'_j := \begin{cases} \epsilon(1+\epsilon)^{\lceil \log_{1+\epsilon}(\epsilon^{-1}\cdot p_j/C)\rceil} C & p_j > \epsilon C \\ p_j & p_j \leq \epsilon C, \end{cases} \tag{3.22}$$

i.e. jobs whose sizes are larger than ϵC are rounded up to the next value of the form $\epsilon(1+\epsilon)^k C$ for integral k, while smaller jobs are unchanged. We call the corresponding sets of jobs $J_\epsilon, J_1, \ldots$. The error introduced by this rounding is not too large:

Lemma 3.2.16. *If there is a schedule of length C of the original instance, there is a schedule of length at most $(1+\epsilon)C$ of the rounded instance.*

Proof. Consider any machine of a schedule of length C, and let J denote the jobs on it. Hence, it holds that

$$\sum_{j\in J} p_j \leq C\,, \tag{3.23}$$

so after rounding, we have

$$\sum_{j\in J} p'_j \leq \sum_{j\in J} \epsilon(1+\epsilon)^{\lceil \log_{1+\epsilon}(\epsilon^{-1}\cdot p_j/C)\rceil} C \leq \sum_{j\in J}(1+\epsilon)p_j \leq (1+\epsilon)C\,, \tag{3.24}$$

as desired. □

Since the lengths of jobs are at most increased and the intervals remain unchanged, we obtain by replacing rounded jobs with their original counterparts:

Remark 3.2.17. If there is a schedule of length $(1+4\epsilon)C$ in the rounded instance, there is a schedule of length at most $(1+4\epsilon)C$ in the original instance.

The crucial effect of this rounding is that the number of exponents k that can occur is bounded: the largest exponent is $q := \lceil \log_{1+\epsilon} \epsilon^{-1} \rceil$, which is constant with regards to n, and the smallest exponent is 1.

The situation for small jobs is not as easy: to solve the problem by dynamic programming, we can assign space for small jobs to machines only in integral multiples of some value, which we set to be ϵC. We will in the folllowing call such a space of ϵC a slot for convenience and note that a small job may be split over up to two slots. This discreteness has consequences in terms of the neccessary and sufficient condition of Corollary 3.2.5: for every interval $[k,l)$, we can without loss of generality round up the number of slots needed to the next integral value. However, this may be misleading, since the resulting constraints are not supermodular anymore, i.e. they need not satisfy Remark 3.2.2, as the following example shows:

Example 3.2.18. Consider two machines, each of which has $\epsilon C/2$ in small jobs. By rounding the constraints over the intervals, we can conclude that we must have one slot on the first machine, one on the second, and one in total.

Note however that we round up at most ϵC per machine, so we can move from the rounded instance I' to an instance I'' by replacing J_ϵ with slot-sized jobs of size ϵC such that for every interval $[k,l)$, the number of these jobs required is

$$\lfloor (\sum_{j \in J_\epsilon|_{[k,l)}} p_j)/(\epsilon C) \rfloor + (l-k). \tag{3.25}$$

Note in particular that the new constraints force every machine to have at least one slot for small jobs.

The deviation made by this change is captured in the following statements:

Lemma 3.2.19. *If I' has a solution of at most length $(1+\epsilon)C$, then I'' has a solution of length at most $(1+3\epsilon)C$.*

If I'' has a solution of length at most $(1+3\epsilon)C$, then I' has a solution of length at most $(1+3\epsilon)C$ which is fractional with regards to the small jobs.

Proof. Consider a solution to I' of length at most $(1+\epsilon)C$. We add, unconditionally, to every machine one slot. This increases the makespan by ϵC, and satisfies all constraints (3.25) by area, but the total size of small jobs on a machine is not yet an integral multiple of ϵC. Rounding it up increases the length of the schedule again by at most ϵC, for a total length of $(1+3\epsilon)C$.

As to the second part, it is sufficient to note that the solution to I'' will satisfy the area constraints of Corollary 3.2.5 for every interval $[k,l)$, since the area it allocates is at least

$$\left(\lfloor \sum_{j\in J_\epsilon|_{[k,l)}} p_j/(\epsilon C)\rfloor + (l-k)\right)(\epsilon C) \geq \left(\lfloor \sum_{j\in J_\epsilon|_{[k,l)}} p_j/(\epsilon C)\rfloor + 1\right)(\epsilon C) \geq \sum_{j\in J_\epsilon|_{[k,l)}} p_j\,,$$

as required. □

The final step is then to convert the fractional solution back to an integral solution, again only making a small error in the process. Formally:

Lemma 3.2.20. *Given a feasible solution of length at most $(1+3\epsilon)C$ which is fractional in the small jobs, we can construct a feasible (and entirely integral) solution of length at most $(1+4\epsilon)C$.*

Proof. By Corollary 3.2.5, we can use the Least Flexible First heuristic Algorithm 3.1 to generate the fractional assignment. Note that on every machine, there are then at most two jobs that are added fractionally: the first and the last small job added by the algorithm. We now assign the 'last' job entirely, which increases the load by at most ϵ and results in a totally integral solution. □

We call a $(q+1)$-tuple $\vec{c} = (n_\epsilon, n_1, \ldots, n_q) \in (\{0,\ldots,n\})^{q+1} =: \mathscr{C}$ a *configuration* and think of a configuration as assigning $n_\epsilon \cdot \epsilon C$ slots for small jobs, n_1 spaces for jobs of rounded size $\epsilon(1+\epsilon)^1 C$, and so on. The *size* of the configuration is defined as

$$\text{size}(\vec{c}) := n_\epsilon \epsilon C + \sum_{k=1}^{q} n_k \epsilon (1+\epsilon)^k C \tag{3.26}$$

and denotes the load a machine would have if it is assigned jobs according to the configuration c.

The previous discussion shows that the main problem in devising a PTAS is to find a feasible assignment of configurations to machines. Since the number of configurations is bounded by $|\mathscr{C}| \leq (n+1)^{q+1}$, i.e. polynomial in n, such an assignment can be found by dynamic programming, but we must be careful to obey the assignment restrictions.

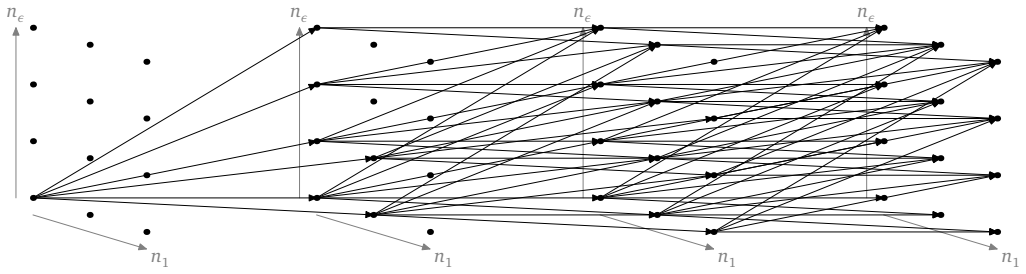


Figure 3.2: The state space digraph

In its most general form, we can think of the dynamic program as finding a path through a suitable state space. Formally, our state space is given by the vertex set $V := \{1, \dots, m+1\} \times (\{0, \dots, n\})^{q+1}$. The first component identifies a prefix of the machines in the interval order, while the second corresponds to a subset of the jobs of the rounded instance. Our intent is that there should be a directed path from $(i, \vec{c})$ to $(i', \vec{c}')$ if and only if we can schedule a subset of jobs as indicated by $\vec{c}' - \vec{c}$ on the machines $\{i, \dots, i'-1\}$. To solve the overall problem, we must then check if there exists a path from $(1, \vec{0})$ to $(m+1, \vec{J})$, where $\vec{J}$ corresponds to the full set of jobs. Formally, we define

$$\begin{aligned} V &:= \{1, \dots, m+1\} \times \mathscr{C}, \\ E &:= \{\big((i, \vec{c}), (i+1, \vec{c}')\big) : i \in \{1, \dots, m\}, \vec{c}, \vec{c}' \in \mathscr{C}, \text{size}(\vec{c}' - \vec{c}) \leq (1+3\epsilon)C\} \end{aligned} \tag{3.27}$$

Example 3.2.21. Focussing on the dynamic programming, we consider an instance with 5 jobs and 3 machines and a guessed makespan of 4. There are $n_1 = 2$ jobs of size 3 and $n_\epsilon = 3$ jobs of size 1. Possible configurations are then: no jobs at all, one large job, one large and one small job, or one, two or three small jobs. Note that these simple sizes are chosen for expository reasons and do not correspond to a rounded instance. Note also that this instance could be solved to optimality by Corollary 3.2.11.

A graphical example of the resulting digraph is given in Figure 3.2, where we already restrict ourselves to arcs that can be reached from the start vertex $(1, \vec{0})$.

Note that the construction of (3.27) is more general than the one by Hochbaum and Shmoys which can just count how often each configuration occurs, but cannot take the assignment restrictions into account. We propose two methods to reflect the restrictions in the dynamic program: one is by a pruning step of the state

digraph structure, leading to Theorem 3.2.15, the other is by a iterative compression routine and will lead to Theorem 3.2.14.

Pruning the state space

Observe the state space in the formulation (3.27) is a worst-case estimate and many of the nodes cannot occur on a path. In particular, we know that some jobs *must* occur on certain machines, while some jobs *cannot*. To reflect this, we define the *lower and upper envelope* of the state space by

$$\ell(i) := J|_{[1,i-1)} \tag{3.28}$$

$$u(i) := (J \setminus J|_{[i,m+1)}) \tag{3.29}$$

for all $i \in \{0,\dots,m\}$; by a slight abuse of notation, we also call the corresponding configurations $\ell(i)$ and $u(i)$, and remark:

Lemma 3.2.22. *No node $(i,\vec{c})$ with $\vec{c} < \ell(i)$ occurs on a feasible $(1,\vec{0})$-$(m+1,\vec{J})$-path; and if feasible $(1,\vec{0})$-$(m+1,\vec{J})$-paths exist, there exists such a path that does not include any node $(i,\vec{c})$ such that $\vec{c} > u(i)$.*

Proof. The claim follows directly from Lemma 3.2.4 if we fix one endpoint of the pseudo-interval to be either $k = 1$ or $l = m+1$: all jobs in $\ell(i)$ must already be scheduled on machines $1\dots,i-1$, and all jobs not in $u(i)$ can only be scheduled on machines $i,\dots,m$, so they cannot be feasibly scheduled on machines $1\dots,i-1$ ever. □

Hence, we can remove all such nodes and all incident arcs from the state space in (3.27).

Example 3.2.23. Continuing with the instance of Example 3.2.21 on p. 73, we now assume the following interval values, which are also shown in Figure 3.4.

j	1	2	3	4	5
p_j	3	3	1	1	1
α_j	1	2	1	2	3
ω_j	3	4	3	3	4

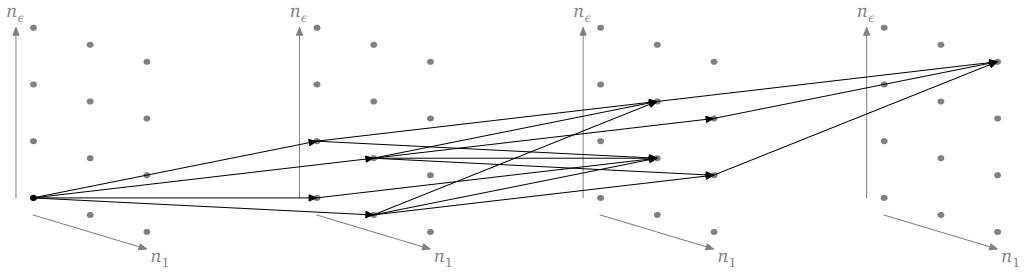
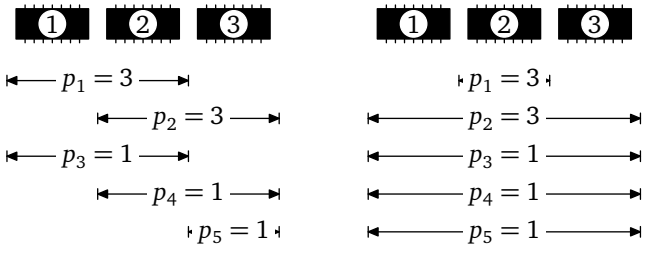


Figure 3.3: A compatible state digraph with useless nodes pruned



(a) Compatible intervals (b) Incompatible intervals

Figure 3.4: Intervals corresponding to the instances of Example 3.2.23

Note that these are compatible in the sense previously defined. Then, Figure 3.3 shows the corresponding pruned state space, where unneeded nodes are shaded.

Note that if we had the following interval values instead:

j	1	2	3	4	5
p_j	3	3	1	1	1
α_j	2	1	1	1	1
ω_j	3	4	4	4	4

the resulting state space is as shown in Figure 3.5, and there is a feasible path given by the configurations $(0,1),(3,0),(0,1)$; however, this path does not correspond to

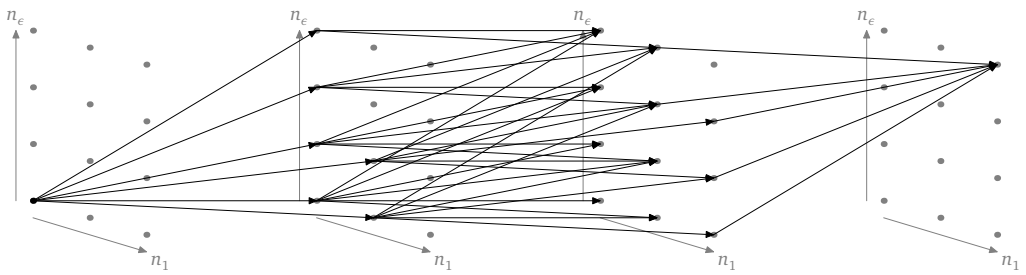


Figure 3.5: An *in*compatible state digraph with useless nodes pruned

a feasible solution of the problem since it assigns the large jobs to the first and last machine.

Somewhat surprisingly, this weaker condition of having a path in the pruned state digraph is still already sufficient for compatible intervals:

Lemma 3.2.24. *If $\alpha_j < \alpha_{j'}$ implies $\omega_j \leq \omega_{j'}$, then* every *path*

$$(1, \vec{0}) = (1, c_1), \ldots, (m+1, c_{m+1}) = (m+1, \vec{J}) \tag{3.30}$$

in the pruned state space digraph induces a feasible assignment of configurations.

Proof. We only need to show that the condition of Lemma 3.2.4 on p. 60 is satisfied. To this end, we fix an arbitrary pseudo-interval $[k, l)$ and one of the size classes and claim that the sum of relevant configurations, $\vec{c_l} - \vec{c_k}$, allocates enough spaces for the rounded jobs in $J''|_{[k,l)}$. If $J''|_{[k,l)}$ is empty, the claim is trivially true. Otherwise, there is some job j with $k \leq \alpha_j < \omega_j \leq l$. In this case, *all* jobs j' with $\alpha_{j'} < k$ have $\omega_{j'} \leq \omega_j \leq l$ by compatibility. In particular, we have $\ell(l) \geq u(k) + J|_{[k,l)}$. We also know by pruning that $\vec{c_l} \geq \ell(l)$ and $\vec{c_k} \leq u(k)$, so in total, we can conclude

$$\vec{c_l} - \vec{c_k} \geq \ell(l) - u(k) \geq u(k) + J|_{[k,l)} - u(k) = J|_{[k,l)}, \tag{3.31}$$

as desired. □

As a corollary, we obtain Theorem 3.2.15, which is summed up in Algorithm 3.2.

Compressing the state space

Another observation can be made regarding the routine that will be used to find a path in the state space. Usual approaches would traverse the state space from one end to the other, which has the advantage that the entire graph need not be kept in memory at the same time, but only two successive layers. However, such algorithms have difficulties satisfying the necessary and sufficient conditions of Lemma 3.2.4 and Corollary 3.2.5. Instead, we can proceed bottom-up, first starting to solve the 'smallest' sub-instances and replacing parts of the state space with

Algorithm 3.2: PTAS for compatible intervals

Round jobs with $p_j > \epsilon C$ to one of q large sizes;
Replace small jobs by their slot requirements;
for $i = 1, \ldots, m+1$ **do**
  Calculate the values $\ell(i)$, $u(i)$;
  Create all configuration nodes $(i, \vec{c})$ with $\ell(i) \leq c \leq u(i)$;
  if $i>1$ **then**
    Create all arcs from $(i-1, \vec{c})$ to $(i, \vec{c'})$ such that size$(\vec{c'} - \vec{c}) \leq (1+3\epsilon)C$;
Find a path from $(1, \vec{0})$ to $(m+1, \vec{J})$;
Return the configurations given by the edges of this path;

a representative selection of solutions to the sub-instance. In the following, we formalize this notion:

Lemma 3.2.25. *In a nested instance, for all jobs j, j', we have*

$$M|_j \cap M|_{j'} \in \{\emptyset, M|_j, M|_{j'}\}. \tag{3.32}$$

Proof. Recall that by definition, $\alpha_j < \alpha_{j'} \implies (\omega_j \geq \omega_{j'} \vee \omega_j \leq \alpha_{j'})$. This means the claim is trivially true if $\alpha_j \geq \alpha_{j'}$.

Otherwise, by nestedness, either $\omega_j \geq \omega_{j'}$, so $M|_j \supset M|_{j'}$, or $\omega_j \leq \alpha_{j'}$, in which case $M|_j$ and $M|_{j'}$ are disjoint. □

Corollary 3.2.26. There exist *minimal* jobs, i.e. $j_0 \in J$ such that $M|_{j_0} \cap M|_j \in \{\emptyset, M|_{j_0}\}$ for all $j' \in J$.

If j_0 is such a minimal job, we can define a shorter state graph by defining a new edgeset

$$E' := \{((\alpha_{j_0}, \vec{c}), (\omega_{j_0}, \vec{c'})) : \vec{c'} - \vec{c} \geq J|_{[\alpha_{j_0}, \omega_{j_0})} \text{ and there is a path from } (\alpha_{j_0}, \vec{c}) \text{ to } (\omega_{j_0}), \vec{c'})\} \tag{3.33}$$

along with one path per edge with that property. E' then replaces all vertices in layers $\alpha_{j_0}+1, \ldots, \omega_{j_0}-1$ along with all incident edges. Clearly, this can be done in polynomial time since the number of configurations is polynomially bounded

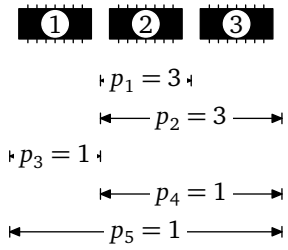


Figure 3.6: An instance with nested interval structure

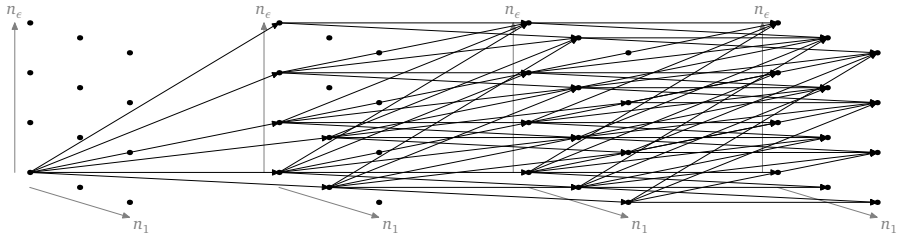
in n, and it does not change the set of states in layer m that can be reached. By iterating this process, we can encode the restrictions of all jobs into the edges, until we arrive at a state graph that does not have additional assignment restrictions. We find a path in this graph in the conventional way and obtain a solution for the original instance by replacing edges with the witnessing paths we stored on creation.

This proves Theorem 3.2.14.

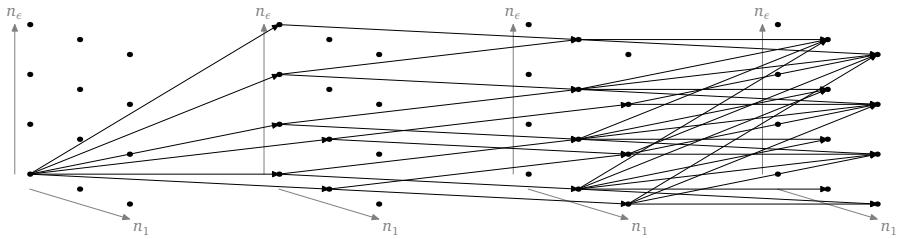
Example 3.2.27. Continuing with the instance of Example 3.2.21 on p. 73, we now assume the following interval values also depicted in Figure 3.6:

j	1	2	3	4	5
p_j	3	3	1	1	1
α_j	2	2	1	2	1
ω_j	3	4	2	4	4

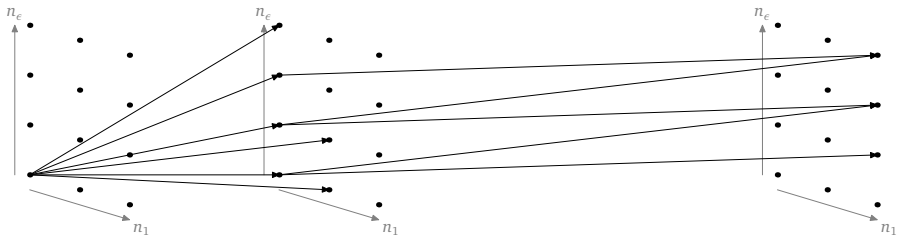
Observe that these are intervals are nested. In Figure 3.7, we see the successive compression steps taken. In the first step, we note that only the configurations $(1,1)$ and $(0,1)$ are feasible on machine 2 due to job 1. In the second step, we compress the interval $[2,4)$, then the interval $[1,2)$. Since there is an arc from $(1,(0,0))$ to $(4,(3,2))$ in the final graph, feasible solutions exist. By backtracking, we can find that one assigns the large jobs to machines 2 and 3 and one small job to each machine.



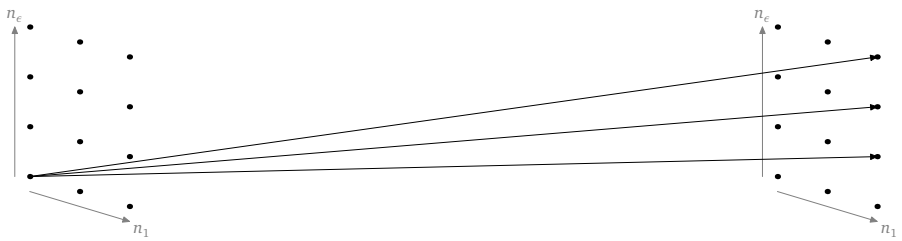
(a) Initial digraph



(b) After collapsing $[2,3)$



(c) After collapsing $[2,4)$



(d) After collapsing $[1,4)$

Figure 3.7: Compression steps in Example 3.2.27

4 A 2-approximation for 2D Bin Packing

4.1 Introduction

Two-dimensional geometric bin packing, both with and without rotations, is one of the very classical problems in combinatorial optimization and its study has begun several decades ago. This is not only due to its theoretical appeal, but also to a large number of applications, ranging from print and web layout [FN04] (putting all ads and articles onto the minimum number of pages) to office planning (putting a fixed number of office cubicles into a small number of floors), to transportation problems (packing goods into the minimum number of standard-sized containers) and VLSI design [HM84].

It is easy to see that two-dimensional bin packing without rotation (2DBP) is strongly NP-hard as a generalization of its one-dimensional counterpart, hence the main focus is on algorithms with provable approximation quality. In fact, even the following stronger statement is well-known to hold:

Lemma 4.1.1. *2DBP is* $(2-\epsilon)$*-inapproximable for all* $\epsilon > 0$*, unless* P = NP.

Proof. Assume we are given a polynomial-time algorithm A with ratio < 2. In particular, A will yield a packing into strictly less than two bins if a packing into one bin exists. Since the number of bins is discrete, it solves all these instances optimally. We now reduce 3-Partition to 2DBP in the following way: given the list of numbers $a_1, \ldots, a_{3n}$ and target sum $B = \sum_{i=1}^{3n} a_i/n$, we create $3n$ items r_i of width $w_i = a_i/B$ and height $h_i = 1/n$ and consider the number of bins A packs them in. Clearly, three items r_i, r_j, r_k fit next to each other if and only if $a_i + a_j + a_k \leq B$,

[FN04] A. Freund and J. Naor. Approximating the advertisement placement problem. *Journal of Scheduling*, 2004.

[HM84] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in robotics and VLSI. In *Proc. STACS*, 1984.

and n such layers can be packed in every bin, so if the 3-Partition instance has a solution, the 2DBP instance will have a packing into one bin.

On the other hand, if A finds a packing into one bin, this bin is filled completely since the total area of items is

$$\sum_{i=1}^{3n} a_i/B \cdot 1/n = \frac{1}{Bn} \sum_{i=1}^{3n} a_i = 1\,. \tag{4.1}$$

In particular, the packing consists of n layers of items, and each layer is filled completely. Since we have $a_i/(4B) < w_i < a_i/(2B)$ for all i, a layer can only be filled completely by exactly three items. Hence, the partition of the items into layers induces a 3-partition of $a_1, \ldots, a_{3n}$. □

The best previous result for the non-rotational case was a 3-approximation by Zhang [Zha05]; very recently, Harren and van Stee have given another 3-approximation with an improved running time of $O(n \log n)$ [HvS10]. Independently from the present work, they also found a 2-approximation [HvS09].

For the case that rotation by 90° is allowed, Harren and van Stee have recently given a 2-approximation in [HvS08]. Since it is also NP-complete to decide whether a set of squares fits into a single bin [LTW$^+$90], this is best possible unless P = NP.

As to asymptotical approximation ratio, Bansal and Sviridenko showed in [BS04] that 2DBP does not admit an asymptotical PTAS. Caprara gave an algorithm with

---

[Zha05] G. Zhang. A 3-approximation algorithm for two-dimensional bin packing. *Operations Research Letters*, 2005.

[HvS10] R. Harren and R. van Stee. Absolute approximation ratios for packing rectangles into bins. *Journal of Scheduling*, 2010. To appear.

[HvS09] R. Harren and R. van Stee. Improved absolute approximation ratios for two-dimensional packing problems. In *Proc. APPROX-RANDOM*, 2009.

[HvS08] R. Harren and R. van Stee. Packing rectangles into 2OPT bins using rotations. In *Proc. SWAT*, 2008.

[LTW$^+$90] J. Y.-T. Leung, T. W. Tam, C. S. Wong, G. H. Young, and F. Y. L. Chin. Packing squares into a square. *Journal of Parallel and Distributed Computing*, 1990.

[BS04] N. Bansal and M. Sviridenko. New approximability and inapproximability results for 2-dimensional bin packing. In *Proc. SODA*, 2004.

asymptotical ratio of 1.69... in [Cap02], breaking the important barrier of 2. More recently, Bansal, Caprara and Sviridenko improved the rate to 1.52... in [BCS06] for both the rotational and non-rotational case.

A closely related problem is two-dimensional knapsack: here, every rectangle also has a profit and the objective is to pack a subset of high profit into a constant number (usually one) of target bins. The best currently known results here are a $(2+\epsilon)$-approximation by Jansen and Zhang [JZ07] for the general case, and a PTAS by Jansen and Solis-Oba [JSO08] if all items are squares. For our purposes, the special case that the profit equals the item's area is important. Bansal et al. have recently shown in [JP09, BCJ+09] that this problem admits a PTAS, and this algorithm will be an important building block of the algorithm presented here.

New result We study the non-rotational geometric two-dimensional bin packing problem, i.e. we are given a list of rectangles (items) $r_1 = (w_1, h_1), \ldots, r_n = (w_n, h_n)$ with all w_i, h_i taken from the interval $]0, 1]$, and the objective is to find a non-overlapping packing of all items into the minimum number of containers (bins) of size 1×1 without rotating the items. The main result of this chapter is the following theorem:

Theorem 4.1.2. *There is a polynomial-time 2-approximation for two-dimensional geometric bin packing.*

This result is achieved using an asymptotic approximation algorithm such as [Cap02] or [BCS06] for large optimal values; smaller (i.e. constant) values

[Cap02] A. Caprara. Packing 2-dimensional bins in harmony. In *Proc. FOCS*, 2002.

[BCS06] N. Bansal, A. Caprara, and M. Sviridenko. Improved approximation algorithms for multidimensional bin packing problems. In *Proc. FOCS*, 2006.

[JZ07] K. Jansen and G. Zhang. Maximizing the total profit of rectangles packed into a rectangle. *Algorithmica*, 2007.

[JSO08] K. Jansen and R. Solis-Oba. A polynomial time approximation scheme for the square packing problem. In *Proc. IPCO*, 2008.

[JP09] K. Jansen and L. Prädel. How to maximize the total area of rectangles packed into a rectangle? Technical Report 0908, Christian-Albrechts-Universität zu Kiel, 2009.

[BCJ+09] N. Bansal, A. Caprara, K. Jansen, L. Prädel, and M. Sviridenko. A structural lemma in 2-dimensional packing, and its implications on approximability. In *Proc. ISAAC*, 2009.

are solved by a recent breakthrough in the approximability of two-dimensional *knapsack* problems in [JP09, BCJ+09]: there, it is proven that there exists a PTAS for maximizing the area covered by rectangles within a 1×1 bin. This can be combined with other packing algorithms if the optimum is constant and at least 2 to generate a packing into $\mathrm{OPT} + 2$ bins. If the optimal packing uses only one bin, we conduct a case study, again starting from a packing that covers $(1 - \epsilon)$ of the bin and generate a packing into $\mathrm{OPT} + 1 = 2$ bins.

As it turns out, this last case is the most involved one; the following crucial theorem is proven in Section 4.5:

Theorem 4.1.3. *There is a polynomial-time algorithm that finds a packing into two bins, provided that a packing into one bin exists.*

4.2 Definitions

In the following, we consider a bin packing instance specified as a list of n items $r_1, \ldots, r_n$, where each $r_i = (w_i, h_i)$ has height h_i and width w_i taken from the interval $]0, 1]$. A packing into a number k of bins is a mapping

$$p \colon \{r_1, \ldots, r_n\} \to \{1, \ldots, k\} \times [0, 1[\times [0, 1[$$

that assigns each item's lower left corner a position in one of the bins such that no two items overlap or protude beyond their bin, without rotating the items. For these purposes, we consider an item $r_i = (w_i, h_i)$ at position (x_i, y_i) to be the cartesian product of open-ended intervals $]x_i, x_i + w_i[\times]y_i, y_i + h_i[$.

In many cases, we pack parts of the instance using the classic 2-approximation for strip packing by Steinberg, which we quote without proof:

Theorem 4.2.1 (Steinberg [Ste97])**.** *We can pack a set of items $\{r_i = (w_i, h_i), i = 1, \ldots, n\}$ into a target area of size $u \times v$ if the following conditions hold:*

1. $\max\{w_i : i = 1, \ldots, n\} \leq u$,

[Ste97] A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing*, 1997.

2. $\max\{h_i : i = 1,\dots,n\} \le v$,

3. $2\sum_{i=1}^{n} w_i h_i \le uv - (2\max\{w_i : 1 \le i \le n\} - u)^+ (2\max\{h_i : 1 \le i \le n\} - v)^+$,

where $(\cdot)^+$ *denotes* $\max\{\cdot, 0\}$.

Corollary 4.2.2. We can pack a set of items $\{r_i = (w_i, h_i), i = 1,\dots,n\}$ into a target area of size $u \times v$ if the following conditions hold:

1. $\max\{w_i : i = 1,\dots,n\} \le u$,

2. $\max\{h_i : i = 1,\dots,n\} \le v/2$,

3. $2\sum_{i=1}^{n} w_i h_i \le uv$.

(As usual, this also holds in the symmetrical case of width and height interchanged.)

4.3 Solving for large optimal values

As mentioned above, Bansal, Caprara and Sviridenko [BCS06] and before that, Caprara [Cap02] have given algorithms (randomized and deterministic, respectively, even though Bansal et al. suggest their algorithm can be derandomized) that have asymptotical approximation ratio strictly smaller than 2. Such algorithms can be used to solve our problem if the optimum is large enough, as the following simple lemma shows:

Lemma 4.3.1. *Given an approximation algorithm A that in polynomial time always generates solutions* $A(I)$ *for instances* I *that satisfy*

$$A(I) \le \rho_A \cdot \mathrm{OPT}(I) + c_A \tag{4.2}$$

for $\rho_A < 2$*, we can solve 2DBP with absolute ratio* 2 *for instances* I *with*

$$\mathrm{OPT}(I) \ge \frac{c_A}{2 - \rho_A} =: K\,. \tag{4.3}$$

Proof. Let I an instance with optimal value at least $c_A/(2-\rho_A)$. Then, we have $c_A \leq (2-\rho_A)\mathrm{OPT}(I)$ and the algorithm generates a packing into at most

$$A(I) \leq \rho_A \cdot \mathrm{OPT}(I) + c_A \leq \rho_A \cdot \mathrm{OPT}(I) + (2-\rho_A)\mathrm{OPT}(I) = 2\mathrm{OPT}(I) \tag{4.4}$$

bins. □

We will dedicate the rest of the section to find a good estimate of K, and then return to the core argumentation in Section 4.4.

Considering the large enumeration steps that will be involved in solving the instance for values of $\mathrm{OPT}(I)$ that are smaller than the threshold value K, it is important to find approximately the size of K. To this end, we observe that both the results given by Caprara and the better randomized result by Bansal et al. are, in spite of the way they are often cited, PTAS-style algorithms, i.e. for arbitrary $\epsilon > 0$, they give algorithms that use $(T_\infty+\epsilon)\mathrm{OPT}(I)+c(\epsilon)$ and $(1+\ln T_\infty+\epsilon)\mathrm{OPT}(I)+c(\epsilon)$ bins, respectively, where smaller values of ϵ not only lead to higher running times, but also to larger additive terms. (Here, $T_\infty \approx 1.69103$, the exact definition is given below.)

We will in the following briefly review Capraras deterministic algorithm and the tradeoff between the constants to find a small threshold value K. We do not repeat the proofs, the reader is referred to the original work [Cap02] for more details.

In broad strokes, the algorithm of Caprara works as shown in Algorithm 4.1. Note that two-dimensional bin packing is reduced to one-dimensional bin packing of the shelves. A very central point for the analysis of Caprara's algorithm is the 'configuration ILP' formulation of the problem that is originally due to Gilmore and Gomory [GG61] and its LP relaxation:

Definition 4.3.2. Let $(s_1 : n_1, \ldots, s_m : n_m)$ an instance of one-dimensional bin packing, meaning that there are n_i items of size s_i, and $n = \sum_{i=1}^m n_i$. A *configuration* is a vector $c = (c_1, \ldots, c_m) \in \{0, \ldots, n\}^m$ such that $c \cdot s = \sum_{i=1}^m c_i s_i \leq 1$.

[GG61] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 1961.

The Gilmore-Gomory-ILP is then given as

$$\begin{aligned} \min \quad & \sum_{c \text{ configuration}} y_c \\ & \sum_{c \text{ configuration}} c_i y_c \geq n_i \quad \forall i \in \{1, \ldots, m\} \\ & y_c \in \mathbb{N} \cup \{0\} \quad \forall \text{configurations } c \end{aligned} \tag{4.5}$$

It is easily seen that feasible solutions to (4.5) correspond exactly to feasible packings of the instance.

Algorithm 4.1: Caprara's 2D bin packing algorithm

Input: Set of items $I = \{r_1, \ldots, r_n\}$, accuracy ϵ
Output: A 2D packing
Calculate $k := k(\epsilon) = (1 + \epsilon^{-1})$;
// Partition the items by their width
$W_q := \{r_j : 1/(q+1) < w_j \leq 1/q\}$ for $q \in \{1, \ldots, k-1\}$;
$W_k := \{r_j : w_j \leq 1/k\}$;
// Pack the shelves
for $q = 1, \ldots, k$ **do**
  Apply NFDH to W_q, obtaining a set of shelves S_q.
// Generate final packing
Use the APTAS of Fernandez de la Vega and Lueker [FL81] for one-dimensional bin packing to pack all the shelves $\bigcup_{q=1}^{k} S_q$ into $(1+\epsilon)\text{OPT} + \delta(\epsilon)$ bins;

The crucial result of Caprara is the following:

Lemma 4.3.3 (Cf. Lemma 8 in [Cap02])**.** *For an instance I of 2DBP and a given k, denote with OPT_S the number of bins needed by an optimal packing of the shelves generated in Algorithm 4.1, with OPT the optimal number of bins needed to directly pack the items in bins, and with OPT_S^* the value of the LP relaxation of* (4.5)*.*

Then, we have

$$\text{OPT}_S^* \leq T_k \text{OPT} + k \tag{4.6}$$

and the algorithm of Fernandez de la Vega and Lueker yields a packing into at most

$$(1+\epsilon)\text{OPT}_S^* + \delta(\epsilon) \tag{4.7}$$

Table 4.1: T_k for $k \leq 6$.

k	2	3	4	5	6
T_k	$\frac{12}{7}$	$\frac{143}{84}$	$\frac{107}{63}$	$\frac{95}{56}$	$\frac{15271}{9030}$
$T_k \approx$	1.714	1.702	1.698	1.696	1.691

bins, where $\delta(\epsilon) = 3(1+\epsilon)^2/\epsilon^2 + 1$*, for a total value of*

$$(1+\epsilon)(T_k \mathrm{OPT} + k) + \delta(\epsilon)\,. \tag{4.8}$$

Here, the sequence $T_2, \ldots$ is defined by the following construction:

$$t_q = \begin{cases} 1 & q = 1 \\ t_{q-1}(t_{q-1}+1) & q > 1 \end{cases}$$

$$T_k = \sum_{q=1}^{m(k)} t_q^{-1} + \frac{k}{k-1} t_{m(k)+1}^{-1}$$

$$T_\infty = \lim_{k\to\infty} T_k \approx 1.69103$$

where $m(k) = \max\{q \in \mathbb{N} : t_q < k\}$. The values of T_k for $2 \leq k \leq 6$ are shown in Table 4.1. We obtain as corollary:

Corollary 4.3.4. Algorithm 4.1 generates a packing into at most

$$(1+\epsilon)\big((T_\infty + \epsilon)\mathrm{OPT} + (1+\epsilon^{-1})\big) + \delta(\epsilon) \tag{4.9}$$

bins.

Proof. It is sufficient to note that all values t_q are positive by definition, hence it holds that

$$T_k = \sum_{q=1}^{m(k)} t_q^{-1} + \frac{k}{k-1} t_{m(k)+1}^{-1} = \sum_{q=1}^{m(k)+1} t_q^{-1} + \frac{1}{(k-1)} t_{m(k)+1}^{-1} \leq T_\infty + \epsilon \cdot t_{m(k)+1}^{-1}\,,$$

which yields the claim since $t_{m(k)+1} \geq 1$. □

In the following, we will restrict ourselves to choices of ϵ that satisfy

$$\epsilon < -\frac{T_\infty + 1}{2} + \sqrt{\frac{(T_\infty + 1)^2}{4} + 2 - T_\infty} \approx 0.1103\,, \tag{4.10}$$

because otherwise, the asymptotic approximation rate of the algorithm is at least 2. In light of (4.3), a small value of K is then given by the following claim:

Lemma 4.3.5. *The function*

$$K(\epsilon) := \frac{(3+\epsilon)(1+\epsilon)^2 + \epsilon^2}{\epsilon^2(2 - (\epsilon^2 + (1+T_\infty)\epsilon + T_\infty))} \tag{4.11}$$

attains a value of $K(\epsilon_0) \leq 6257$ *for* $\epsilon_0 := 0.0715$*; for no* ϵ *in the range given by* (4.10)*,* $K(\epsilon) < 6150$*.*

Proof. For $\epsilon_0 = 0.0715$, we get, since $T_\infty < 1.692$,

$$K(\epsilon_0) = \frac{690.7993596}{1.92338775 - 1.0715 T_\infty} \leq \frac{690.7993596}{1.92338775 - 1.0715 \cdot 1.692} < 6256.69\,. \tag{4.12}$$

As to the lower bound, we are interested in the values of ϵ for which the inequality

$$6150 \leq K(\epsilon) \tag{4.13}$$

holds, in particular that all interesting values of ϵ satisfy this. Observe that the denominator of (4.11) will be positive, since $\rho_A < 2$, so by rearranging (4.11), we have to show that

$$6150\epsilon^2(2 - (\epsilon^2 + (1+T_\infty)\epsilon + T_\infty)) \leq (3+\epsilon)(1+\epsilon)^2 + \epsilon^2\,, \tag{4.14}$$

which by further simplification is equivalent to

$$-6150\epsilon^4 - (6151 + 6150T_\infty)\epsilon^3 + (12294 - 6150T_\infty)\epsilon^2 - 7\epsilon - 3 \leq 0\,. \tag{4.15}$$

Since the left-hand side is decreasing in T_∞, it is sufficient to replace T_∞ by the

lower bound 1.691, thus obtaining the condition

$$-6150\epsilon^4 - 16550.65\epsilon^3 + 1894.35\epsilon^2 - 7\epsilon - 3 \leq 0\,, \tag{4.16}$$

or equivalently

$$\epsilon^4 + \frac{1655065}{615000}\epsilon^3 - \frac{189435}{615000}\epsilon^2 + \frac{700}{615000}\epsilon + \frac{300}{615000} \geq 0. \tag{4.17}$$

We can use Ferrari's formula for quartic functions to solve this for equality. The desired outcome will be that no positive real solution in the interesting range for ϵ exists. Denoting the coefficient of ϵ^i by a_i for the moment, we proceed by standard technique and define

$$\begin{aligned}
\alpha &:= -3a_3^2/8 + a_2 = -\frac{97142670169}{40344000000} \approx -2.408\\
\beta &:= a_3^3/8 - a_3a_2/2 + a_1 = \frac{30115692477567197}{14886936000000000} \approx 2.023\\
\gamma &:= -3a_3^4/256 + a_2a_3^2/16 - a_3a_1/4 + a_0 = \frac{-9287698905263666488561}{19531660032000000000000}\,.
\end{aligned}$$

Since $\beta \neq 0$, we set

$$\begin{aligned}
P &= -\alpha/12 - \gamma = -\frac{461653103}{60516000000}\\
Q &= -\alpha^3/108 + \alpha\gamma/3 - \beta^2/8 = -\frac{4637080016869}{7443468000000000}\\
R &= -Q/2 + \sqrt{Q^2/4 + P^3/27}\\
&= \frac{4637080016869}{14886936000000000} + \frac{\sqrt{3616339753054281145133112 15}}{66991212000000000}\\
U &= \sqrt[3]{R}\\
y &= -\frac{5}{6}\alpha + U - \frac{P}{3U} \approx 2.121\\
W &= \sqrt{\alpha + 2y} \approx 1.354
\end{aligned}$$

to finally obtain the four, possibly complex, solutions

$$x = -a_3/4 + \frac{s \cdot W + s' \cdot \sqrt{-3\alpha - 2y - s \cdot 2\beta/W}}{2} \tag{4.18}$$

for $s, s' \in \{-1, +1\}$. Using the approximate values for α and y, combined with the fact that β and W are positive, we see that the solution will be real for $s = -1$, and the approximate solution is then

$$-\frac{1655065}{2460000} - \frac{1.354}{2} \pm \frac{1}{2}\sqrt{-3\alpha - 2y + \cdot 2\beta/W} \tag{4.19}$$

which even in the larger case of $s' = +1$ is only (approximately) $-0.673 - 0.677 + 1.222 = -0.128 < 0$, so both solutions are negative.

For $s = +1$, we obtain that the radicand is approximately -0.006, i.e. in that case, the solution will be a non-real complex number.

Since none of the roots are positive reals, either all or no values of ϵ satisfy (4.16). However, it is easily seen that for $\epsilon = 1/30$, we obtain $1894.35\epsilon^2 = 1894.35/900 < 3$, so substituting into (4.16) yields

$$-6150\epsilon^4 - 16550.65\epsilon^3 + 1894.35\epsilon^2 - 7\epsilon - 3 < -6150\epsilon^4 - 16550.65\epsilon^3 - 7\epsilon \leq 0\,. \tag{4.20}$$

Hence, all interesting values of ϵ satisfy (4.16), so $K(\epsilon) \geq 6150$, as desired. □

Remark 4.3.6. By closer numerical approximation of T_∞, we can obtain $K = 6199$ for $\epsilon_0 := 0.07178$, but we omit the proof, which is only of numerical nature.

In fact, we can obtain a smaller value of K by modifying the algorithm of Caprara: as mentioned there in passing, the final bin packing step in Algorithm 4.1 can also be done with an algorithm due to Karmarkar and Karp [KK82] instead of the algorithm by Fernandez de la Vega and Lueker. The ratio of the Karp/Karmarkar algorithm is given by the following theorem:

Theorem 4.3.7 (Theorem 4 in [KK82])**.** *There is a polynomial-time algorithm A for*

[KK82] N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proc. FOCS*, 1982.

(one-dimensional) Bin Packing such that

$$A(I) \leq \max\{(1+2/\text{SIZE})\text{OPT}+1, \\ \text{OPT}^* + (1+\log_2 \text{SIZE})(9+4\ln \text{SIZE}) + 2 + \frac{2}{1-(\ln \text{SIZE})/2}\} \tag{4.21}$$

where SIZE *is the total area of items and* OPT^* *is the optimal value of the bin packing LP relaxation of* (4.5).

By considering the algorithm NFDH, it is easy to see the very generous bound of $\text{SIZE} \leq \text{OPT} \leq 3\text{SIZE}$ so the first term is bounded by $\text{OPT}+7$, additionally, for $\text{SIZE} > e^2$, the fraction $2/(1-(\ln \text{SIZE})/2)$ becomes negative. Combining this, the ratio of Karp and Karmarkar's algorithm is bounded by $\text{OPT} + (1+\log_2 \text{OPT})(9+4\ln \text{OPT})+2$ for $\text{OPT} > 3e^2 \approx 22.1$.

Combining this with Lemma 4.3.3, we can replace (4.7) with (4.21) to obtain that the modification of Algorithm 4.1 will find a solution that is bounded from above by

$$\begin{aligned}&(T_k\text{OPT}+k) + (1+\log_2(T_k\text{OPT}+k))(9+4\ln(T_k\text{OPT}+k))+2 \\ &\quad = T_k\text{OPT} + (k+11) + (4+\frac{9}{\ln 2})\ln(T_k\text{OPT}+k) + \frac{4}{\ln 2}\ln^2(T_k\text{OPT}+k)\end{aligned} \tag{4.22}$$

where $k = k(\epsilon)$ is the number of width classes. Since this term contains the variables both linearly and logarithmically, it is not easily solved; however we can make the following observation about the asymptotic behaviour:

Lemma 4.3.8. *For all* $a, b > 0$, $n \in \mathbb{N}$, *the function* $x \mapsto (\ln(ax+b))^n/x$ *is monotonically decreasing for* $x > (e^n - b)/a$.

Proof. The derivative of $x \mapsto (\ln(ax+b))^n/x$ is

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}x}\frac{(\ln(ax+b))^n}{x} &= \frac{1}{x^2}\left(\frac{n(\ln(ax+b)^{n-1}ax}{ax+b} - (\ln(ax+b))^n\right) \\ &= \frac{(\ln(ax+b))^{n-1}}{x^2}\left(n\frac{ax}{ax+b} - \ln(ax+b)\right) \\ &< \frac{(\ln(ax+b))^{n-1}}{x^2}(n - \ln(ax+b))\end{aligned}$$

which is negative for $n < \ln(ax+b)$. The claim follows. □

As corollary, we obtain that for fixed k, the ratio of the modified Caprara algorithm is decreasing monotonically for $\mathrm{OPT} > (\mathrm{e}^2 - k)/T_k$.

The exact tradeoff is more difficult to study here because of the presence of both linear and logarithmic terms in k. However, numerical analysis of the values suggests a setting of $k = 6$. If we define for clarity

$$L = \ln(T_6\mathrm{OPT} + 6) \tag{4.23}$$

$$B(\mathrm{OPT}) = T_6\mathrm{OPT} + 17 + (4 + \frac{9}{\ln 2})L + \frac{4}{\ln 2}L^2, \tag{4.24}$$

we see that $B(\mathrm{OPT})/\mathrm{OPT}$ will be decreasing for $\mathrm{OPT} > 1 > (\mathrm{e}^2 - 6)/T_6$ since by Lemma 4.3.8, all terms are then either constant or decreasing. Considering the non-integral value $\mathrm{OPT} = (\mathrm{e}^8 - 6)/T_6 \in]1759, 1760[$, we obtain $L = 8$ by definition, and hence for this choice

$$\begin{aligned} B(\mathrm{OPT}) &= T_6\mathrm{OPT} + 49 + \frac{328}{\ln 2} \\ &< \frac{15271}{9030} \cdot 1760 + 49 + \frac{328}{\ln 2} < 3499 < 2 \cdot 1759 < 2\mathrm{OPT}, \end{aligned} \tag{4.25}$$

so for optimal values of 1760 or more, the algorithm has absolute ratio 2.

Again, the value was chosen to make L manageable; numerically, one can verify that substituting the values $\mathrm{OPT} = 1446$, $k = 6$ into (4.22), we get a result of

$$T_k\mathrm{OPT} + (k+11) + (4 + \frac{9}{\ln 2})\ln(T_k\mathrm{OPT} + k) + \frac{4}{\ln 2}\ln^2(T_k\mathrm{OPT} + k) = 1.9997\mathrm{OPT}. \tag{4.26}$$

In total, we obtain

Theorem 4.3.9. *For* $\mathrm{OPT} \geq 1446$*, the algorithm of Caprara yields a packing into at most* $2\mathrm{OPT}$ *bins.*

4.4 Solving with $\mathrm{OPT} + 2$ bins for constant OPT

In this section, we will prove the following theorem:

Theorem 4.4.1. *For every constant k, there is a polynomial time bin packing algorithm B_k which for every given instance either correctly decides that no packing into k bins exists at all or returns a packing into $k+2$ bins.*

The algorithm works in two steps: first, it tries to pack almost all items (up to total unpacked area at most $1/2$) into k bins. If this succeeds, the rest is packed into the remaining two bins. Hence, Theorem 4.4.1 is a direct consequence of the following two statements:

Theorem 4.4.2. *For constant k, ϵ, there is a polynomial time knapsack algorithm $K_{k,\epsilon}$ that for every given instance $r_1, \dots, r_n$ either returns a packing of a subset $S \subseteq \{r_1, \dots, r_n\}$ of items such that the total area of unpacked items is at most ϵ and every unpacked item is bounded in one direction by ϵ or correctly decides that no packing into k bins exists.*

Lemma 4.4.3. *There is a polynomial time algorithm which packs items with total area at most $1/2$ into two bins.*

Proof of Lemma 4.4.3. Assign all items of height at least $1/2$ to the first bin and note that the total width of these items is at most 1 since the total area is bounded by $1/2$, hence they can be packed trivially next to one another. All remaining items have height less than $1/2$ and a total area of at most $1/2$, so they can be packed into a 1×1 bin by using Steinberg's algorithm. □

4.4.1 Proof of Theorem 4.4.2

To prove Theorem 4.4.2, we extend the 2D knapsack algorithm of Jansen and Prädel [JP09] to a constant number of target areas. We do this by a suitable embedding into the one-area case, which is captured in the following observations:

Remark 4.4.4. An instance I has a packing into k bins if and only if it has a packing into the disjoint areas $[2j, 2j+1) \times [0, 1)$ for $j \in \{0, \dots, k-1\}$.

We now add $k-1$ items items $r_{n+1}, \dots, r_{n+k-1}$ that are all sized 1×1, denoting the resulting instance I'. Again, the following is immediate:

Remark 4.4.5. I has a packing into the disjoint areas $[2j, 2j+1) \times [0,1)$ for $j \in \{0, \dots, k-1\}$ if and only if I' has a packing that places item r_{n+j}'s lower left corner at position $(2j-1, 0)$, for $j \in \{1, \dots, k-1\}$.

The total target area of I' is sized $(2k-1) \times 1$, clearly we can rescale all items and the target area horizontally by a factor of $1/(2k-1)$ without affecting feasibility. We still call this instance I' and sum up the previous discussion in the following lemma:

Lemma 4.4.6. *I has a packing into k bins if and only if I' has a packing into one unit bin such that the lower left corner of item r_{n+j} is at position $((2j-1)/(2k-1), 0)$, for $j \in \{1, \dots, k-1\}$.*

In remains to show that we can extend the algorithm of [JP09] to the case of a constant number of 'pinned' items, which we do in the following.

Let $\epsilon \leq 1/2$ the desired accuracy of the algorithm, i.e. the area we are allowed to discard. To accomodate the horizontal scaling process, we re-set $\epsilon := \epsilon/(2k-1)$, since every item is $(2k-1)$ times bigger in the unscaled instance than in the scaled instance.

On a very high level, the algorithm, like many approximation algorithms, works with a two-pronged approach: on the one hand, we manipulate an unknown optimal solution into a canonical form which is almost as good. This is done in such a way that the set of possible canonical forms is computationally tractable, i.e. of polynomial size. On the algorithmic side, we can then try by brute force all canonical forms and return the best result we can obtain, which must be at least as good as the one for the near-optimal canonical form. In our case, the result we obtain for any single canonical form will again only almost be as good as the best possible result, but the total error still is bounded.

We first consider the manipulation of the optimal solution of I', and denote with (x_i^*, y_i^*) the position of the lower left corner of r_i in this solution. In particular, we know $(x_{n+i}^*, y_{n+i}^*) = ((2j-1)/(2k-1), 0)$ for $j \in \{1, \dots, k-1\}$ by our definition of feasible packings.

As in [JP09], we define $\epsilon' := \max\{(2z)^{-1} : z \in \mathbb{N}, (2z)^{-1} \leq \epsilon/4\}$ and a series of

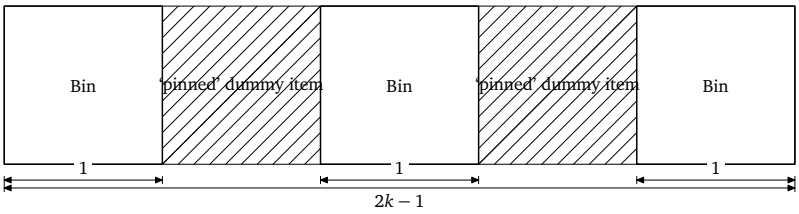


Figure 4.1: Packing k bins by using $k-1$ pinned items

threshold values by

$$\sigma_1 := \epsilon' \qquad \sigma_{j+1} = \sigma_j^{8+12/\sigma_j} \forall j \in \mathbb{N}. \tag{4.27}$$

Then, there must be some $2 \leq j \leq 1 + 4/\epsilon'$ such that the area

$$\sum \{w_i h_i : (\sigma_{j+1} < w_i \leq \sigma_j) \vee (\sigma_{j+1} < h_i \leq \sigma_j)\}$$

is bounded by $\epsilon' A'/2$. Otherwise, we would have obtained a total area of strictly more than $(4/\epsilon')\epsilon' A'/2 = 2A'$, but every item is counted at most twice, once for its width and once for its height. We set $\ell := \sigma_{j+1}$ and $u := \sigma_j$ and discard all items with w_i or h_i in $]\ell, u]$. We call all items with $w_i, h_i > u$ big. Note that items r_{n+j} for $j \geq 1$ have $h_{n+j} = 1 > \epsilon \geq \epsilon'$ and $w_{n+j} = 1/(2k-1) > \epsilon \geq \epsilon'$, so they are big and hence are never discarded in this step.

Just as in [JP09, Sect. 3], the placement of the big items induces a set of *gap rectangles* G, which have the following properties:

1. We can add the gap rectangles into the optimal packing if we remove all non-big items from it and leave the big items in their positions.
2. The possible widths of gap rectangles are drawn from a set of polynomial size.
3. The possible heights of gap rectangles are drawn from a set of polynomial size.
4. The number of gap rectangles is bounded by a constant.
5. The total area of non-big items which would intersect the boundary of a gap rectangle (if they had not been removed) is bounded by $3u \leq 3\epsilon'$.

Here, polynomial and constant refer to dependency on n, the values are super-polynomially in ϵ. The most notable property for our extension is the first: by construction, the big rectangles in the optimal packing are not shifted when creating gap rectangles. In particular, our filler items $r_{n+i}, i \in \{1, \ldots, k-1\}$ remain where they are.

We can now turn to the algorithmic side: we consider all possible candidate sets of gap rectangles, together with all big items of the instance. (We know by assumption that a packing exists, so all big items can be packed.) In [JP09], a packing, if one exists, is then found by enumeration of bottom-left (BL) packings. Recall that a BL packing is obtained by shifting each item and rectangle down and left as far as possible, i.e. until every item's lower boundary touches the top of another item or the bottom of a bin and the left boundary touches the right side of another item or the left side of the bin. Hence, every BL packing of a set of rectangles R can be given by a function

$$p : R \to (R \dot\cup \{\text{BIN}\}) \times (R \dot\cup \{\text{BIN}\}) \tag{4.28}$$

that identifies these two touched items. The number of packings can then be bounded by $(|R|+1)^{2|R|}$ which is constant provided that $|R|$ is constant. The proper packing can be reconstructed from p in polynomial time as sketched as Algorithm 4.2.

However, it is possible that no BL packing is feasible in the sense that it places the items $r_{n+i}, i \in \{1, \ldots, k-1\}$ correctly. A solution of I' thus obtained would not transform back to a packing into k unit bins of I. We extend the concept of BL packings to accomodate fixed items in the straight-forward manner: if F is a set of fixed rectangles and R a set of other rectangles, the main difference is that F does not need to have two touching neighbors. A packing is then given by

$$p : R \to (R \dot\cup F \dot\cup \{\text{BIN}\}) \times (R \dot\cup F \dot\cup \{\text{BIN}\}), \tag{4.29}$$

so the number of packings is bounded by $(|R|+|F|+1)^{2|R|}$ which again is constant if both $|F|$ and $|R|$ are constant. A packing can be reconstructed from p by simply initializing the coordinates of the rectangles in F and starting with $D := F \dot\cup \{\text{BIN}\}$

Algorithm 4.2: Create a BL packing

Input: function $p : R \to (R \dot\cup \{\text{BIN}\}) \times (R \dot\cup \{\text{BIN}\})$
Output: the corresponding packing of R
$D := \{\text{BIN}\}$;
while $R \setminus D \neq \emptyset$ **do**
  Find $r \in R \setminus D$ with $p(r) = (l, b) \in D \times D$;
  if $l = \text{BIN}$ **then**
    $x_r := 0$;
  else
    $x_r := x_l + w_l$;
  if $b = \text{BIN}$ **then**
    $y_r := 0$;
  else
    $y_r := y_b + h_b$;
  $D := D \cup \{r\}$;

in Algorithm 4.2. In total, we can still enumerate all feasible packings of the big items and gap rectangles that have items $r_{n+i}, i \in \{1, \dots, k-1\}$ in the desired place.

Given the packing that corresponds to the optimal solution, we can proceed in exactly the same way as in [JP09], so the following result holds:

Theorem 4.4.7 (Theorem 3 in [JP09])**.** *The algorithm computes a solution with area of at least* $(1-\epsilon)\text{OPT}$.

Since all big items are always selected for packing, and all non-big items are bounded in one direction by $u \leq \epsilon$, this proves Theorem 4.4.2.

4.5 Solving with 2 bins for $\text{OPT} = 1$

In this section, we consider the remaining case that there exists a packing of all items into a single bin. We will start off by some general statements that can be shown for packings into a single bin before showing that each instance falls into one of four cases, each of which we consider separately. As a manner of speaking, we define $T := \{r_i : h_i > 1/2\}$ the set of *tall items* and $W := \{r_i : w_i > 1/2\}$ the

set of *wide items*. We extend the notion of width and height to sets S of items by setting $w(S) := \sum_{i \in S} w_i$, the *total width of* S and $h(S) := \sum_{i \in S} h_i$, the *total height of* S.

Let us first make easy observations about the presence of tall and wide items in an instance that admits a packing into one bin. As everywhere in this chapter, these results still hold for wide items instead of tall items by transposing width and height.

Remark 4.5.1. We can always fit all tall items into a single bin by packing them next to each other in non-increasing order of height, since no two of them fit on top of each other in the optimum.

Remark 4.5.2. If we can pack a set of items which includes all tall items into one bin such that the total area of the packed items is at least $1/2$, we can pack the remainder of the instance into the second bin using Steinberg's algorithm.

Lemma 4.5.3. *Consider some $\gamma \in [0, 1/2[$ and let w the total width of all items of height at least $1-\gamma$. Then, the total height of items of width larger than $\max\{1/2, 1-w\}$ and height less than $1-\gamma$ is at most 2γ.*

Proof. Consider a horizontal line $y = y_0$ in any feasible packing, for any $y_0 \in]\gamma, 1-\gamma[$. Such a line clearly must intersect all items of height at least $1-\gamma$, cf. Figure 4.2, which take up total width w. In particular, it cannot intersect any other item of width more than $1-w$, so all these items must be located in the outermost γ of the bin. Since the items are also wide, no two of them could be next to one another, so the total height can be at most 2γ. □

Two parameters will appear in the following analysis, a width limit for tall items δ and the accuracy ϵ used for the knapsack PTAS. We set

$$\delta := 1/12; \quad \epsilon := \min\{\delta/144, 1/308.4\} = 1/1728\,. \tag{4.30}$$

4.5.1 Many tall or many wide items

In this section, we consider the case that the subset of tall items, i.e. those of height more than $1/2$, is comparatively large. (Symmetrically, this also solves the case

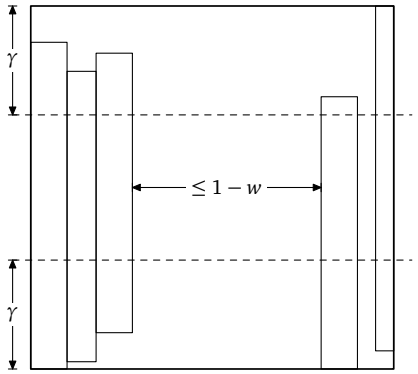


Figure 4.2: Items of height $\geq 1-\gamma$ limit items of width $> 1-w$.

that many items are wide; again, we only consider tall items explicitly.) The result we claim is the following:

Lemma 4.5.4. *If the total width of tall items $w(T)$ is at least $1-\delta$, we can pack all items into two bins.*

We will show this in the following way: we first pack all tall items next to each other by Remark 4.5.1, and then try to pack additional items so that the total area covered is at least $1/2$, at which point we can invoke Remark 4.5.2 to pack the remainder in the other bin with Steinberg's algorithm. Note that the tall items alone already cover at least $(1-\delta)/2$, so we need only $\delta/2$ in extra items. We will try this in five different ways corresponding to five classes of items, which in total cover all non-tall items. If none of these five succeeds, we know that the total area of items in each class is bounded by a small term in δ. In particular, the total unpacked area will then be bounded by $1/2$ so we can still use Steinberg's algorithm on the second bin and the unpacked items.

To see this, we first show some technical results:

Lemma 4.5.5. *Each item r_i satisfies at least one of the following conditions, which are also sketched in Figure 4.3*

1. $h_i > 1/3$,
2. $h_i \cdot w_i \geq \delta/2$,
3. $h_i \leq 2\delta$ *and* $w_i \leq 1/2$,

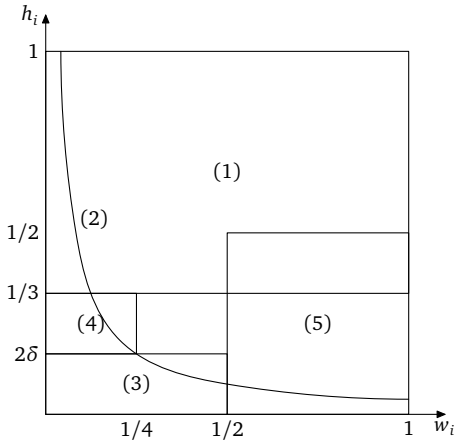


Figure 4.3: The five cases of Lemma 4.5.5

4. $2\delta < h_i \leq 1/3$ *and* $w_i \leq 1/4$,

5. $h_i \leq 1/2$ *and* $w_i > 1/2$.

Proof. Consider an item that does not satisfy Case 1. Either its height is larger than 2δ. Then either Case 2 holds or its width is at most $1/4$, in which case, Case 4 holds. Otherwise, its height is at most 2δ. Then either its width is at most $1/2$, so that Case 3 holds, or it is larger than $1/2$, then since $2\delta \leq 1/2$, Case 5 holds. □

Lemma 4.5.6. *Given a list of rectangles $q_1 = (w_1, h_1) \ldots, q_m = (w_m, h_m)$ of total width at most 1 and one extra rectangle $q' = (w', h')$ with $h' \leq 1/2$, we can either pack these items into one bin or the set $\{q'\} \cup \{q_i : h_i > 1 - h'\}$ cannot be packed into a single bin at all.*

Note in particular that we do not require the q_i to be a subset of the input instance.

Proof. By reindexing, we may assume $h_1 \geq h_2 \geq \cdots \geq h_m$. We pack the items at the bottom of the bin in this order, cf. the hatched area in Figure 4.4. This is feasible since their total width is at most 1. Assume that placing q' in the top-right corner creates an overlap with some certain q_i. Since $h' < 1/2$, we have $h_1 \geq \cdots \geq h_i > 1 - h' \geq 1/2 \geq h'$, so no two of these could be on top of one another in any feasible packing. However, we have $w_1 + \cdots + w_i > 1 - w'$, so a bin of width 1 does not admit a packing of all these items next to each other either. □

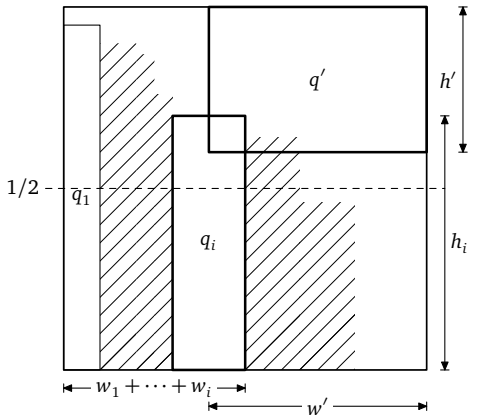


Figure 4.4: A tall item intersecting the extra item of Lemma 4.5.6

In particular, we can always place one item from the instance along with all tall items, so from Remark 4.5.2, Remark 4.5.1 and Lemma 4.5.6, we obtain:

Corollary 4.5.7. If the total area of tall items is (at least) $1/2 - \delta/2$, then all other items have individual area at most $\delta/2$, or else we can pack the instance into two bins.

Note that for purposes of proving Lemma 4.5.4, this means that we can restrict ourselves to the case that Case 2 of Lemma 4.5.5 does not hold true for any other item, i.e. all other items have (individual) area of less than $\delta/2$, and in particular they are bounded in at least one direction by $\sqrt{\delta/2}$.

Similar to Lemma 4.5.6, we can show:

Lemma 4.5.8. *If the total width of tall items $w(T)$ is at least $1 - \delta$, we can pack all tall items and leave an empty area sized $(1 - \delta/(1-2h)) \times h$ in the top right corner for any desired $0 < h < 1/2$, or we can directly pack the instance into two bins.*

Proof. As before, we order the tall items by non-increasing height and pack them from left to right. Note that there is a total area of at least $(1-\delta)/2$ covered by tall items below the line $y = 1/2$. If the area $(1 - \delta/(1-2h)) \times h$ intersects the tall items, then in particular the point $(\delta/(1-2h); 1-h)$ is within some tall item, cf. Figure 4.5. This means that there is covered area above the line $y = 1/2$ of at least $\delta/(1-2h) \cdot (1/2 - h) = \delta/2$. Thus, the total area of tall items would be at least $1/2$, and we can pack the instance by Remark 4.5.2. □

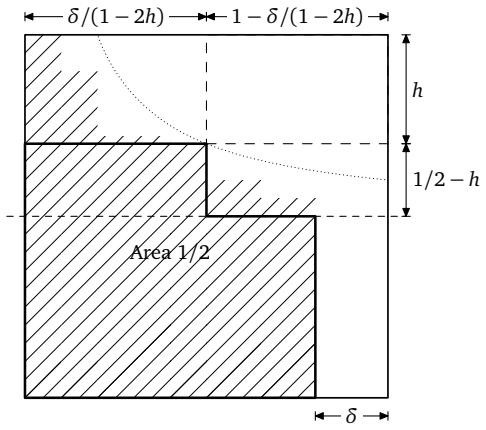


Figure 4.5: Free space available in the top right corner.

We will now use this bound of the area for two classes of items, those that satisfy either Case 3 or 4 in Lemma 4.5.5. Let us first consider Case 3, the set of items of height at most 2δ and width at most $1/2$. Assume that the total area of these items is at least $\delta/2$ and greedily select a subset S of total area in the interval $[\delta/2, \delta[$. This is possible, since every individual item can be assumed to have area at most $\delta/2$ by Corollary 4.5.7.

We define a container of size $1/2 \times 2/5$. Note that its height is $2/5 > 4/12 = 2 \cdot 2\delta$, so more than twice the height of every item, and its area is $1/5 \geq 2\delta$, so more than twice the total area of selected items S. In particular, we can pack S into this container with Steinberg's algorithm by Corollary 4.2.2. It remains to verify that the container itself can be packed by Lemma 4.5.8, and indeed its height is $2/5 < 1/2$ and the allowed width of a container of this height would even be

$$1 - \frac{\delta}{1 - 2 \cdot 2/5} = 1 - 5\delta = \frac{7}{12} > \frac{1}{2}, \tag{4.31}$$

so the container fits. This shows:

Lemma 4.5.9. *If $w(T) \geq 1 - \delta$ and the total area of items with $h_i \leq 2\delta$ and $w_i \leq 1/2$ is at least $\delta/2$, we can pack all items into two bins.*

We now turn to the items of Case 4, having height in the interval $]2\delta, 1/3]$ and width at most $1/4$. Again, if these items have total area at least $\delta/2$, we can select a subset S with area in the interval $[\delta/2, \delta[$ and pack this subset into a container

sized $1/2 \times 2/5$: as just seen, this container's area is large enough; obviously its width is at least twice that of any item in S, and we have also already seen that this container fits in the top right corner. This shows:

Lemma 4.5.10. *If $w(T) \geq 1 - \delta$ and the total area of items with $2\delta < h_i \leq 1/3$ and $w_i \leq 1/4$ is at least $\delta/2$, we can pack all items into two bins.*

Next, we consider Case 5 of Lemma 4.5.5, the items of width more than $1/2$ which are not already packed. (There could be one wide item that is also tall.) Their individual height is automatically less than δ by Corollary 4.5.7. We can pack a specific subset of these by using the following lemma:

Lemma 4.5.11. *Let $r_1, \ldots, r_m$ be the items of $W \setminus T$, i.e. all wide items apart from up to one item which is tall as well, ordered by non-decreasing width, and let $k \leq m$ such that $\sum_{i=1}^{k} h_i \leq h(W \setminus T)/2$. We can then pack T and $\{r_1, \ldots, r_k\}$ into a single bin.*

Proof. Pack the tall items from left to right ordered by non-increasing height at the bottom of the bin and stack the wide items from the top right corner downwards ordered by non-increasing width as shown in Figure 4.6, and assume that there is an overlap. Choose $j \leq k$ maximal such that r_j intersects a tall item r_ℓ. Clearly, the total width of items at least as tall as r_ℓ is larger than $1 - w_j$, otherwise, the overlap would not have occurred. By Lemma 4.5.3, setting $\gamma = 1 - h_\ell < 1/2$, the total height of wide non-tall items of width at least w_j is then at most $2(1 - h_\ell)$, however, it is also at least

$$\sum_{i=j}^{m} h_i = \sum_{i=j}^{k} h_i + \sum_{i=k+1}^{m} h_i \geq \sum_{i=j}^{k} h_i + h(W \setminus T)/2 \geq 2\sum_{i=j}^{k} h_i > 2(1 - h_\ell), \tag{4.32}$$

which contradicts the assumption of overlap. □

If we assume that the total area of items in $W \setminus T$ is at least 4δ, then their total height is also at least 4δ. In particular, we can greedily select the narrowest wide items of total height at least δ and at most 2δ. (Recall the individual height of these items is at most δ.) Their total area will then be at least $\delta/2$ and they must be packable by the previous discussion. This shows:

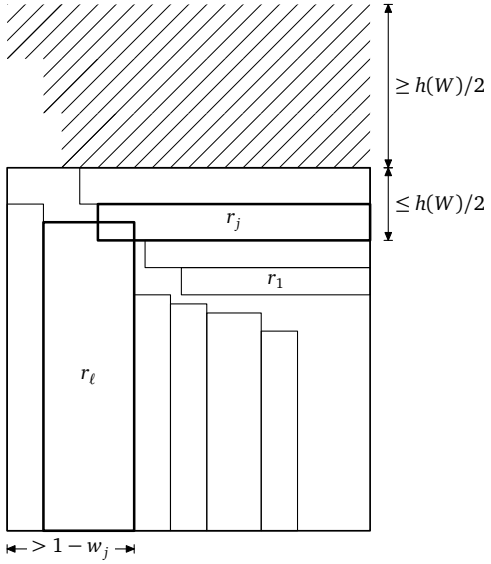


Figure 4.6: Tall and wide items in a single bin.

Corollary 4.5.12. If $w(T) \geq 1 - \delta$ and the total area of items with $w_i > 1/2$ and $h_1 \leq 1/2$ is at least 4δ, we can pack all items into two bins.

Finally, we consider Case 1, items of height larger than $1/3$, but at most $1/2$. Each item's width is then bounded by $3\delta/2$ by Corollary 4.5.7. We then succeed by the following lemma:

Lemma 4.5.13. *We can pack all but one items of height larger than* $1/3$ *into one bin, and the unpacked item has height at most* $1/2$.

Proof. The idea of this proof is a generalization of a result implicit in Graham's proof of the performance of the Longest Processing Time scheduling heuristic [Gra69], i.e. that this heuristic is optimal as long as there are at most two jobs per machine. We sort all items by non-increasing height (assume by reindexing $h_1 \geq h_2 \ldots$) and start packing them at the bottom of the bin until the total width is at least 1. If the width is strictly larger, the last item, r_k, protrudes beyond the bin and we split it. (This is the one item that we are allowed to not pack at the end.) By Remark 4.5.1, the split item cannot have height larger than $1/2$. The rest of the split item and

[Gra69] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 1969.

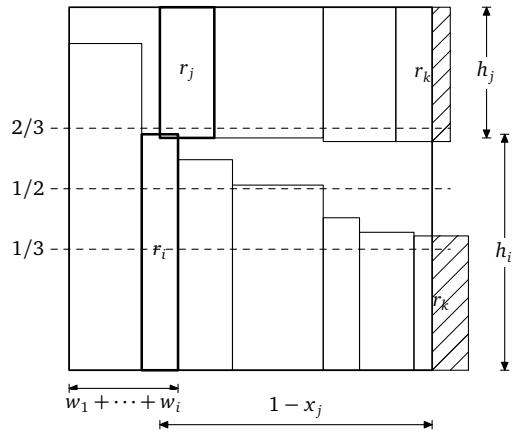


Figure 4.7: Items of height larger than 1/3 in a single bin

all further items are packed from right to left at the top of the bin into a 'reverse shelf', cf. Figure 4.7.

Assume now that there is a collision of items, say r_i at position $(x_i, 0)$ in the lower shelf, collides with r_j at $(x_j, 1 - h_j)$ in the upper shelf. Since all items in the upper shelf have height at most 1/2, we obtain $h_i > 1/2$. Also, we can conclude that the total width of items of height at least h_j, $w_1 + \cdots + w_j$, is at least $2 - x_j$, and the total width of items of height at least $1 - h_j$ is at least $w_1 + \cdots + w_i > x_j$.

However, in *any* feasible packing of the items $r_1, \ldots, r_j$, the items $r_1, \ldots, r_i$ of total width $w_1 + \cdots + w_i > x_j$ cannot be above each other, because $h_i > 1/2$, nor can any of the items $r_{i+1}, \ldots, r_j$ be above or below one of them because already $h_i + h_j > 1$. Since $h_j > 1/3$, at most two items in $r_{i+1}, \ldots, r_j$ can be on top of one another in any packing. Hence, the total width taken up by all these items in any packing is at least

$$w_1 + \cdots + w_i + \frac{w_{i+1} + \cdots + w_j}{2} = \frac{1}{2}\sum_{k=1}^{j} w_k + \frac{1}{2}\sum_{k=1}^{i} w_k > \frac{1}{2}(2 - x_j) + \frac{1}{2}x_j = 1\,, \tag{4.33}$$

which contradicts that there is a feasible packing into one bin.

Hence, no overlap can have occured, so our packing is feasible apart from the fact that at most one item is split. We discard this item. □

Assume the total area of items of height larger than 1/3 and at most 1/2 is at least δ. The previous lemma then packs all but one item, but the area of the

discarded item is at most $\delta/2$ by Corollary 4.5.7, so we have still packed at least $\delta/2$ additional area.

Note that the precondition of all cases was that certain items have a certain minimum total area, and all non-tall items are counted at least once by Lemma 4.5.5. If none of these attempts solves the problem, we can hence bound the total area of non-tall items as follows: δ by Case 1, 0 by Case 2, $\delta/2$ for each of Case 3 and 4 and 4δ for Case 5, for a total of $6\delta \leq 1/2$, so we can nonetheless pack all non-tall items in the second bin using Steinberg's algorithm.

This, all put together, shows Lemma 4.5.4. In the following, we therefore always assume that $w(T) \leq 1 - \delta$ and, by symmetry, $h(W) \leq 1 - \delta$.

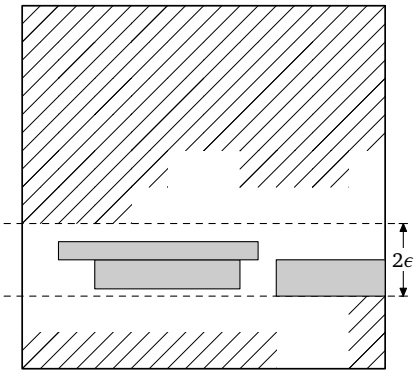
In the remaining cases, we will always pursue the same angle of attack: starting off with a packing of area $(\sum_{i=1}^{n} w_i h_i) - \epsilon$ into one bin generated with the algorithm in [JP09], we will identify a suitable strip of size 2ϵ and move all items that properly intersect the strip into the second bin, cf. Figure 4.8a. For convenience, we always consider horizontal strips, but all results still hold with 'horizontal' and 'vertical' interchanged.

All unpacked items that are bounded in height by ϵ can then be packed into the empty strip sized $1 \times 2\epsilon$ in the first bin using Steinberg's algorithm by Corollary 4.2.2.

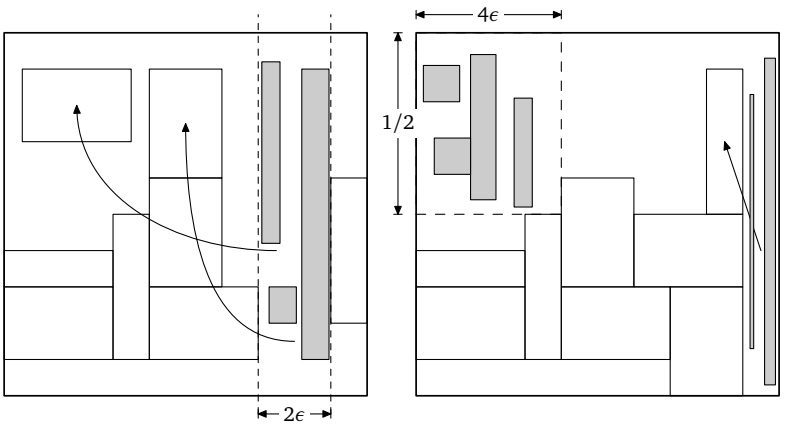
We will then re-arrange the moved items in the second bin in such a way that the second bin also accomodates the other unpacked items of area at most ϵ (each of which is bounded in width by ϵ) in one of two ways: we either clear a full-height area of size $2\epsilon \times 1$, Figure 4.8b, into which they can be packed by Steinberg's algorithm again, or we will argue that in specific cases, a certain subset of tall unpacked items can be packed 'manually' so that the rest can fit into a free area of height less than 1 but width larger than 2ϵ as in Figure 4.8c.

The following lemma will prove useful for rearranging items of height at most 1/2:

Lemma 4.5.14. *Given a set $\{a_1 \geq \ldots \geq a_m\}$ of numbers, a total width $S \geq \sum_{i=1}^{m} a_i$ and a desired target value T such that $S \geq 2T + a_1$, we can find in linear time a subset $P \subseteq \{1, \ldots, m\}$ such that $\sum_{i \in P} a_i \leq S - T$ and $\sum_{i \notin P} a_i \leq S - T$.*



(a) Clearing a horizontal strip in the first bin



(b) Clearing a vertical strip in the second bin

(c) Finding two areas in the second bin

Figure 4.8: General approach

Proof. If $\sum_{i=1}^{m} a_i \leq S - T$, $P := \emptyset$ is a trivial solution. Otherwise, we find $k < m$ such that $\sum_{i=1}^{k} a_i \leq S - T < \sum_{i=1}^{k+1} a_i$. Then, we also have

$$\sum_{i=k+1}^{m} a_i \leq S - \sum_{i=1}^{k+1} a_i + a_{k+1} < T + a_{k+1} \leq T + a_1 \leq S - T\,, \tag{4.34}$$

so $P = \{1, \ldots, k\}$ is the desired set. □

4.5.2 One big item

In this section, we will consider the case that there exists one item in the packing, say r_1, such that $w_1, h_1 > 1/2$. By Lemma 4.5.4, we also may assume that $w_1, h_1 \leq 1-\delta$. Let (x_1, y_1) the coordinates of r_1's lower left corner. Without loss of generality, we assume the bottom edge of r_1 is at least as close to the bottom of the bin as the top edge to the top, i.e. $y_1 \leq 1 - h_1 - y_1$, which means $y_1 \leq (1-h_1)/2$. (Otherwise, we imagine the packing flipped upside down.) We consider the strip defined by $y \in]y_1, y_1+2\epsilon[$ and denote with S the set of items that intersect the strip. We move all items in S that intersect the line $y = y_1 + 2\epsilon$ (in particular, r_1) to the second bin. Note that all items in S that do not intersect the line $y = y_1 + 2\epsilon$ are already packed in two areas sized $x_1 \times (y_1 + 2\epsilon)$ and $(1 - w_1 - x_1) \times (y_1 + 2\epsilon)$, because they were either to the left or to the right of r_1. Since $x_1 + (1 - w_1 - x_1) = 1 - w_1 \leq 1/2 \leq w_1$ and $y_1 + 2\epsilon \leq (1 - h_1)/2 + 2\epsilon \leq 1/4 + 2\epsilon \leq h_1 - 2\epsilon$, we can pack these areas into the empty space freed by r_1 without obstructing the horizontal strip at the bottom, cf. Figure 4.9.

Let us now order the items in the second bin by non-increasing height, and note in particular that by Lemma 4.5.4, we may assume that the total width of tall items, $w_T := w(S \cap T)$, is at most $1 - \delta$.

We consider two cases now: either there is a non-tall item, say r_2, of width at least $1 - w_T - 4\epsilon$ or not. If there is no such item, we can apply Lemma 4.5.14 to free a vertical strip of width at least 2ϵ as shown in Figure 4.10a: we set the target width $T := 2\epsilon$. The total width available S is at least $1 - w_T$. The numbers are the widths of the non-tall items, so the lemma yields a partition of the items into two subsets, each of which has total width at most $1 - w_T - 2\delta$. Since all these items are not tall, this yields a packing into two shelves atop one another.

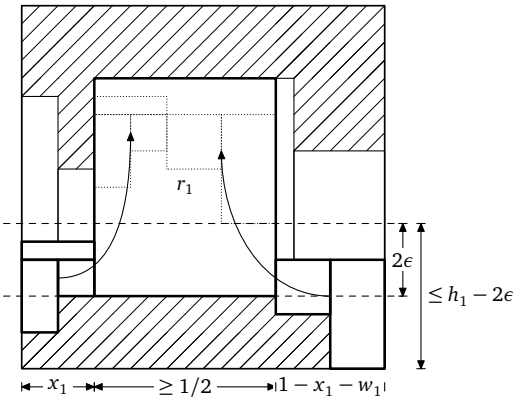


Figure 4.9: Re-packing items in the first bin if a big item exists

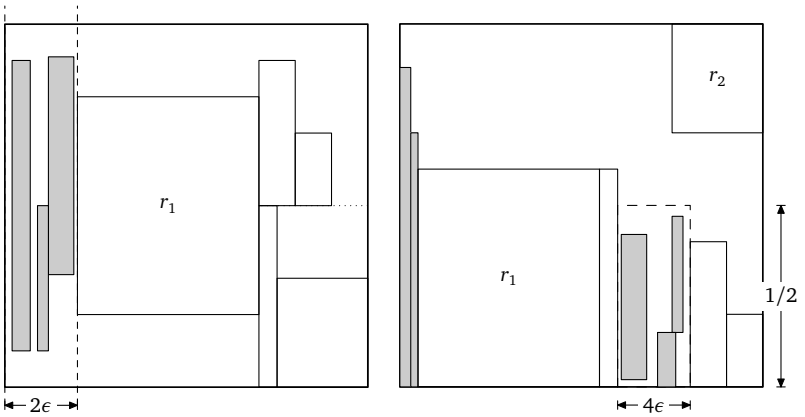


Figure 4.10: Re-packing if a big item r_1 exists

(a) Clearing a strip if many items are narrow enough

(b) Clearing areas if a sufficiently wide item exists

If, however, such r_2 exists, it has width at least $1 - w_T - 4\epsilon \geq \delta - 4\epsilon \geq 6\epsilon$. In particular, we can apply Lemma 4.5.6 on the following items: 1. $S \setminus \{r_2\}$, 2. all unpacked tall items (of total width at most 2ϵ) 3. a container sized $4\epsilon \times 1/2$, into which we can pack all remaining unpacked items by Steinberg's algorithm, and use r_2 as the extra item to pack into the top-right corner, as depicted in Figure 4.10b. Note that r_2 and the container will not intersect since both their heights are bounded by $1/2$.

4.5.3 One medium item

In this section, we will consider the case that there exists some item in the packing, say r_1, such that $w_1, h_1 \geq 12\epsilon$, and this item's lower left corner is at (x_1, y_1). In light of the previous section, we can assume $\min\{w_1, h_1\} \leq 1/2$, say $h_1 \leq 1/2$. We also assume again that "r_1 is in the lower half of the bin": $y_1 \leq 1 - y_1 - h_1$, i.e. $y_1 \leq (1 - h_1)/2$, otherwise we flip the packing upside-down.

We now set $y_0 := \max\{2\epsilon, y_1\}$ and consider three consecutive horizontal strips: Strip I is defined by $y \in]y_0, y_0 + 2\epsilon[$, Strip II by $y \in]y_0 + 2\epsilon, y_0 + 4\epsilon[$ and Strip III by $y \in]y_0 + 4\epsilon, y_0 + 6\epsilon[$, see Figure 4.12a. Since $y_0 + 6\epsilon \leq y_1 + 2\epsilon + 6\epsilon < y_1 + h_1$, all three strips are entirely bisected by r_1.

We claim the following properties hold:

$$y_0 + 6\epsilon \leq 1/2\,, \tag{4.35}$$

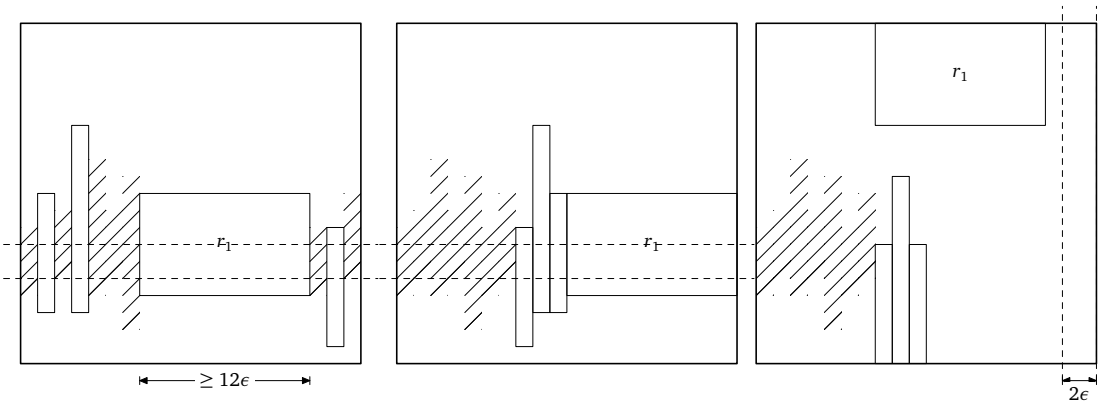
$$y_0 + 4\epsilon \leq 1 - h_1\,. \tag{4.36}$$

Eq. (4.35) is trivial for $y_0 = 2\epsilon$ since $\epsilon \leq 1/16$, and for $y_0 = y_1$ we have $y_0 + 6\epsilon \leq y_1 + h_1/2 \leq (1 - h_1)/2 + h_1/2 = 1/2$. As to (4.36), it is now sufficient to note that $y_0 + 4\epsilon < y_0 + 6\epsilon \leq 1/2 \leq 1 - h_1$.

We are now interested in the sets of items that intersect the strips, which we will denote by S_I, S_{II} and S_{III}, respectively.

Remark 4.5.15. If one of Strips I, II, III contains items other than r_1 of height at most $1 - h_1$ and total width at least 2ϵ that totally bisect the strip, we can pack the instance into two bins.

Proof. Move the strip in question and the corresponding items to the second bin,



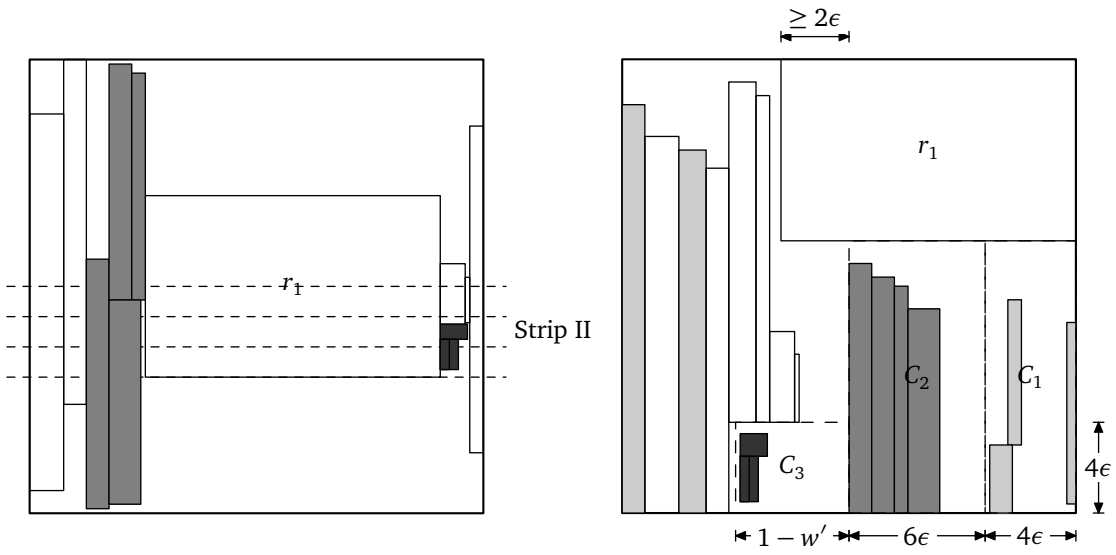
(a) Strip moved to the second bin (b) Reordering items (c) Shifting items

Figure 4.11: Packing items if enough items exist in Remark 4.5.15

maintaining their packing, as shown in Figure 4.11a. We modify the packing as follows: By reordering, we can assume that all bisecting items are adjacent to r_1 and r_1 is at the right side of the bin, as shown in Figure 4.11b. r_1 is shifted up to the top of the bin, all other items that bisect the strip are shifted down to the bottom of the bin. It is then possible to shift r_1 to the left by at least 2ϵ, which frees a vertical strip of width 2ϵ, cf. Figure 4.11c. □

If this does not apply, we will move S_{II} to the second bin and rearrange it to accomodate the remaining items. In more detail, we create the following packing (cf. Figure 4.12b): the item r_1 is packed in the top right corner of the bin. Below it, there are two containers, C_1 sized $4\epsilon \times (1-h_1)$ and C_2 sized $6\epsilon \times (1-h_1)$. The first holds all unpacked items of height at most $1-h_1$, packed with Steinberg's algorithm. This is feasible by Corollary 4.2.2 since each unpacked item's width is bounded by $\epsilon \leq 4\epsilon/2$ and their total area is at most $\epsilon \leq (4\epsilon)/4 \leq (4\epsilon)\cdot(1-h_1)/2$. The container C_2 contains all items of S_{II} with height at most $1-h_1$ that bisected at least one of Strip I, II or III entirely, which means they can be packed next to each other since by Remark 4.5.15, their total width is at most 6ϵ. (These are marked in a darker shade in both Figure 4.12a and Figure 4.12b.) Note in particular that all items with height in the interval $[4\epsilon, 1-h_1]$ end up in C_2.

The remaining items in $S_{II} \setminus S_{III}$, shaded darkest, can be shifted into a container C_3 sized $(1-w') \times 4\epsilon$, where w' is the total width of all items in $S_{II} \cap S_{III}$ that



(a) A suitable strip, before reordering

(b) after reordering

Figure 4.12: Reordering items that intersect Strip II.

bisect Strip II. It is immediate that this width is sufficient, because all items in $S_{II} \setminus S_{III}$ do not intersect Strip III and no item in S_{II} is below an item that bisects Strip II. As to the height, note that all these items are bounded in height by $1 - h_1$ by (4.36), and all those that bisected an entire strip were already removed to C_2 above. This means that all remaining items in $S_{II} \setminus S_{III}$ do not cross the line $y = y_0$ nor $y = y_0 + 4\epsilon$. We position C_3 at the bottom of the bin, next to C_1 and C_2, and note that it is shifted at least 2ϵ under r_1. Since its width is at most $1 - w_1$, the combined width of C_1, C_2, C_3 is less than $1 - 2\epsilon$.

Now, the following items are still remaining: unpacked items of height more than $1 - h_1$, and packed items of height either larger than $1 - h_1$ or smaller than 4ϵ from the set $S_{II} \cap S_{III} \setminus \{r_1\}$, i.e. they all intersected the line $y = y_0 + 4\epsilon$. Note that the total width of all these is at most $\epsilon/(1-h_1) + (1-w_1) \leq 2\epsilon + 1 - w_1 \leq 1 - 10\epsilon$.

We sort these items by decreasing height and pack them left-to-right, starting at position $(0, 0)$, and continuing on top of C_3. The total width available is hence $1 - 10\epsilon$, so the items will not intersect C_2, but conceivably intersect r_1 or extend beyond the top of the bin.

Assume that some item r_i collides with r_1. This cannot be an item of height at

most 4ϵ since $y_i + h_i \leq 4\epsilon + 4\epsilon \leq 1/2 \leq 1 - h_1$, so its height must be larger than $1 - h_1$. However, the collision would then contradict Lemma 4.5.6, since all such items must have fit next to r_1 in an optimal packing.

Finally consider that some item r_i might protude beyond the top of the bin (whether or not it collides with r_1). Such an item must have $h_i > 1 - 4\epsilon$ and be positioned atop C_3. However, since $1/2 > y_0 \geq 2\epsilon$, this means that r_i either completely bisected both S_{II} and S_{III} or was unpacked. The total width of such items (other than r_1) is at most $(w' - w_1) + \epsilon/(1 - 4\epsilon) < w' - w_1 + 2\epsilon$. The width of the area next to C_1, C_2, C_3 is $1 - (1 - w') - 10\epsilon = w' - 10\epsilon \geq w' - w_1 + 2\epsilon$ since $w_1 \geq 12\epsilon$, so all items of height more than $1 - 4\epsilon$ were successfully packed there by the algorithm.

4.5.4 All small and elongated items

In this section, we consider the remaining case that every packed item is bounded in at least one direction by 12ϵ. Note that all unpacked items are even bounded by ϵ in one direction. First of all, we want to show that the difficult subcase here is if there are few items which have one 'medium' sidelength. (We show the claim for items of medium height, but the same argument works for medium width.)

Lemma 4.5.16. *If the total area of packed items of height at least* 12ϵ *and at most* $1/2$ *and width at most* 12ϵ *('tallish items') is at least* 19.2ϵ*, we can pack all items into two bins.*

Proof. Suppose for illustration that there is a strip $y \in]y_0, y_0 + 2\epsilon[$ that is entirely bisected by some tallish items. If the total width of these items is at least 16ϵ, we can move this strip to the second bin and apply Lemma 4.5.14 with $T := 2\epsilon$ to find a partition of the tallish items into two shelves such that we clear a vertical strip of width 2ϵ in the second bin.

To formalize this notion and show that such a strip must exist, we define for every tallish item r_i packed at location (x_i, y_i) the function

$$\chi_i(y) := \begin{cases} w_i, & y \in [y_i, y_i + h_i - 2\epsilon] \\ 0, & \text{otherwise.} \end{cases} \tag{4.37}$$

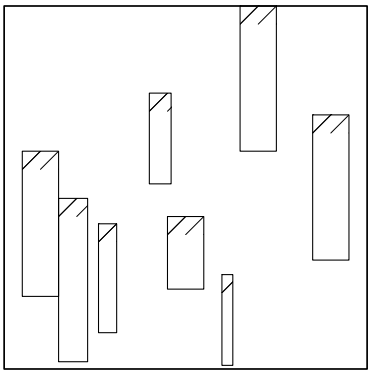


Figure 4.13: Areas of tallish items that "do not count towards the 16ϵ".

Note that r_i will completely bisect the strip $]y, y+2\epsilon[$ iff $\chi_i(y) = w_i$. (See also Figure 4.13, where the missing 2ϵ are hatched.) We have that

$$\int_0^1 \chi_i(y)dy = w_i \cdot (h_i - 2\epsilon) \geq w_i \cdot h_i \cdot 10/12\,, \tag{4.38}$$

since $h_i \geq 12\epsilon$. Summing over all tallish items, we obtain that

$$\int_0^1 \sum_{r_i \text{ tallish}} \chi_i(y)dy \geq \sum_{r_i \text{ tallish}} \frac{10}{12} w_i h_i = \frac{10}{12} \cdot \sum_{r_i \text{ tallish}} w_i h_i \geq \frac{10}{12} \cdot 19.2\epsilon = 16\epsilon\,. \tag{4.39}$$

In particular, there exists some y such that $\sum\{\chi_i(y) : r_i \text{ tallish}\} \geq 16\epsilon$, which identifies a suitable strip for re-packing in the second bin.

We can also find such a strip in polynomial time. To do this, note that when sweeping the horizontal line from the bottom of the bin upwards, the amount of 'counting' tallish items $\sum\{\chi_i(y) : r_i \text{ tallish}\}$ only increases if $y = y_i$ for some tallish item r_i. In particular, the maximum value, which is at least 16ϵ, is attained in one of the at most n elements of $\{y_i : r_i \text{ packed and tallish}\}$. □

If the previous lemma does not give us a solution, we know that most of the area of the instance is in items that are either tall or wide or very small in both directions. (We have $2 \cdot 19.2\epsilon$ in tallish and widish items and ϵ in unpacked items that we have not reasoned about yet.) In this case, we will construct a packing from scratch as shown in Figure 4.14. Beforehand, we would like to recall a classical lemma concerning Next Fit Decreasing Height (NFDH) when applied to small items:

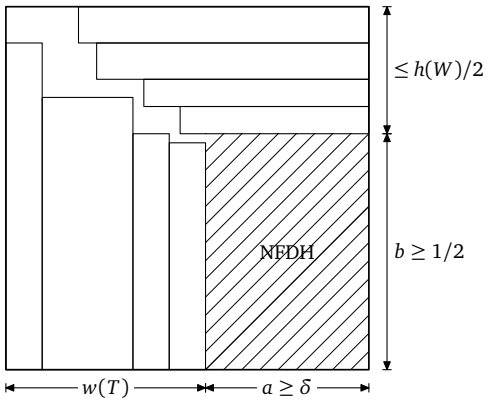


Figure 4.14: Packing items if few tallish and widish items exist

Lemma 4.5.17. *Given a set of items which are bounded in both width and height by* 12ϵ *and a target area sized* $a \times b$ *(for* $a, b \geq 12\epsilon$*), NFDH packs all the items or covers an area of at least* $(a - 12\epsilon)(b - 24\epsilon)$*.*

Proof. Consider the case that NFDH does not pack all the items, and denote with H_i and W_i the height and used width of the ith shelf, for $i = 1, \ldots, k$. Then, $\sum_{i=1}^{k} H_i > b - 12\epsilon$, otherwise, another shelf could have been added. Also, all W_i are at least $a - 12\epsilon$, because otherwise, a further item would have been added. The total area covered by in the ith shelf is then at least $(a - 12\epsilon) \cdot H_{i+1}$, so summing over all shelves, we cover

$$(a - 12\epsilon) \sum_{i=1}^{k-1} H_{i+1} = (a - 12\epsilon)(\sum_{i=1}^{k} H_i - H_1) > (a - 12\epsilon)(b - 12\epsilon - 12\epsilon), \tag{4.40}$$

as desired. □

In the following, we denote with A_{tall} the total area of all tall items, with A_{wide} the total area of wide items and with A_{small} the total area of items which are bounded by 12ϵ in both directions. Without loss of generality, we assume $A_{tall} \geq A_{wide}$. We have already shown in Lemma 4.5.11 that we can arrange all of the tall items and approximately half (in terms of total height) of the wide items as shown in Figure 4.14. The area covered by these items is at least $A_{tall} + A_{wide}/3 - 12\epsilon$, since all packed wide items might have width close to $1/2$ while all unpacked wide items might have width 1, and one item of individual area at most 12ϵ might be

split. Denote with a and b the width and height of the area to be filled with NFDH. Following Lemma 4.5.4, we may assume that $a \geq \delta$ and $b \geq 1 - (1-\delta)/2 > 1/2$. (Bear in mind we have not packed at least half of the stack of wide items in Lemma 4.5.11.) In particular, by Lemma 4.5.17 we either pack all small items there or cover an area of

$$(a - 12\epsilon)(b - 24\epsilon) > ab - 12\epsilon(2a + b) \geq ab - 36\epsilon \geq ab - \delta/4 \geq ab/2, \tag{4.41}$$

where we use that $\epsilon \leq \delta/144$ and $ab \geq \delta/2$. In this case, we have filled the entire bin at least halfway: the area sized $(1-a) \times 1$ at the left side of the bin is covered at least halfway by the tall items, the (not disjoint) area sized $1 \times (1-b)$ at the top of the bin is covered at least halfway by wide items, and as we have just seen, the NFDH region is also covered with at least $ab/2$.

Even if we run out of small items, the area remaining for the second bin is small: it is bounded by $(2A_{wide}/3 + 12\epsilon) + 2 \cdot 19.2\epsilon + \epsilon$ for wide, widish and tallish and unpacked (non-tall non-wide) items, respectively. Since $A_{wide} \leq A_{tall}$, we have $A_{wide} \leq 1/2$, so the above sum is bounded by $1/3 + 51.4\epsilon$, which is at most $1/2$ since $\epsilon \leq 1/308.4$. Hence, in either case, the second bin can be packed using Steinberg's algorithm.

Summing up, the overall algorithm works as outlined in Algorithm 4.3.

4.6 Conclusion

We have presented an algorithm that generates 2-approximate solutions for two-dimensional geometric bin packing, which matches the rate known for the rotational problem. Since both the rotational and non-rotational problem are not approximable to any $2 - \epsilon$ unless P = NP, this settles the question of absolute approximability of these problems. For practical applications, it would be interesting to find faster algorithms: our algorithm relies heavily on the knapsack PTAS in [JP09, BCJ+09] and techniques in [JS07] with a doubly-exponential dependency on ϵ, in particular when compared to the running time $O(n \log n)$ of Harren

[JS07] K. Jansen and R. Solis-Oba. New approximability results for 2-dimensional packing problems. In *Proc. MFCS*, 2007.

Algorithm 4.3: 2-approximation for 2D Bin Packing

```
Run the algorithm of Bansal et al. [BCS06];
for k = 2, ..., K do
    Run the algorithm of Section 4.4;
if the area of all items is at most 1 then
    if Lemma 4.5.4 can be applied then
        Apply Lemma 4.5.4;
    else
        Generate a packing of (1 − ϵ) area in the first bin using [JP09];
        if this packing contains a big item then
            apply the algorithm in Subsection 4.5.2;
        else if this packing contains an item of at least 12ϵ in both directions
        then
            apply the algorithm in Subsection 4.5.3;
        else
            apply the algorithm in Subsection 4.5.4;
Return the best solution found;
```

and van Stee's 3-approximation in [HvS10]. Still, our result is an important step in the study of two-dimensional packing problems.

Another important open problem is the gap in asymptotic behaviour between the non-existence of an APTAS and the best known algorithm with asymptotic quality of 1.525.

5 Concluding Remarks

In this thesis, we have presented results on three subjects in packing and scheduling problems: online scheduling, with special focus on the impact of machine unavailability; offline scheduling on unrelated machines; and two-dimensional geometric bin packing.

Of course, not all questions concerning these problems have be answered here. In particular, two important issues remain open for future research concerning bin packing and restricted assignment:

We have shown in Chapter 4 that there exists a polynomial-time 2-approximation for 2DBP, which matches the best lower bound possible unless P = NP. A crucial subroutine for this was an algorithm with asymptotic ratio better than 2, and any improvement there that yields smaller values of the threshold constant K would yield better running times, in addition to being an interesting research subject in its own right. Hence the following problem deserves future attention:

Open Question 1. *What asymptotic ratios are possible for 2DBP in polynomial time? In particular, does 2DBP admit an asymptotic* 1.5*-approximation?*

Our second question concerns Scheduling with Assignment Restrictions: we have shown in Subsection 3.2.1 that this problem seems to be easier on interval graphs. Our advances are comparatively small, but it opens an important direction: to the author's best knowledge, no research has been previously done on the subject that exploits special properties of the LP formulation such as total unimodularity or total dual integrality. The full impact this will have is not quite clear, so we ask:

Open Question 2. *Does Scheduling with Interval Assignment Restrictions admit polynomial-time algorithms with approximation ratio* $1 + \epsilon$ *for some* $\epsilon < 1$*, possibly even a PTAS?*

Bibliography

[ABC+99] Foto N. Afrati, Evripidis Bampis, Chandra Chekuri, David R. Karger, Claire Kenyon, Sanjeev Khanna, Ioannis Milis, Maurice Queyranne, Martin Skutella, Clifford Stein, and Maxim Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proceedings of FOCS*, pages 32–44, 1999.

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[AS01] Susanne Albers and Günter Schmidt. Scheduling with unexpected machine breakdowns. *Discrete Applied Mathematics*, 110(2-3):85–99, 2001.

[BCJ+09] Nikhil Bansal, Alberto Caprara, Klaus Jansen, Lars Prädel, and Maxim Sviridenko. A structural lemma in 2-dimensional packing, and its implications on approximability. In *Proceedings of ISAAC*, volume 5878 of *LNCS*, pages 77–86. Springer, 2009.

[BCS06] Nikhil Bansal, Alberto Caprara, and Maxim Sviridenko. Improved approximation algorithms for multidimensional bin packing problems. In *Proceedings of FOCS*, pages 697–708. IEEE Computer Society, 2006.

[BL76] Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976.

[BM10] Péter Biró and Eric McDermid. Matching with sizes (or scheduling with processing set restrictions). Technical Report TR-2010-307, University of Glasgow, 2010.

[BS04] Nikhil Bansal and Maxim Sviridenko. New approximability and inapproximability results for 2-dimensional bin packing. In J. Ian Munro, editor, *Proceedings of SODA*, pages 196–203. SIAM, 2004.

[Cap02] Alberto Caprara. Packing 2-dimensional bins in harmony. In *Proceedings of FOCS*, pages 490–499. IEEE Computer Society, 2002.

[CLRS90] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 1990.

[DJ09] Florian Diedrich and Klaus Jansen. Improved approximation algorithms for scheduling with fixed jobs. In Claire Mathieu, editor, *Proceedings of SODA*, pages 675–684. SIAM, 2009.

[DJPT07] Florian Diedrich, Klaus Jansen, Fanny Pascual, and Denis Trystram. Approximation algorithms for scheduling with reservations. In Srinivas Aluru, Manish Parashar, Ramamurthy Badrinath, and Viktor K. Prasanna, editors, *Proceedings of HiPC*, volume 4873 of *Lecture Notes in Computer Science*, pages 297–307. Springer, 2007.

[DS07] Florian Diedrich and Ulrich Michael Schwarz. A framework for scheduling with online availability. In Anne-Marie Kermarrec, Luc Bougé, and Thierry Priol, editors, *Proceedings of Euro-Par*, volume 4641 of *Lecture Notes in Computer Science*, pages 205–213. Springer, 2007.

[EDMT07] Lionel Eyraud-Dubois, Gregory Mounie, and Denis Trystram. Analysis of scheduling algorithms with reservations. In *Proceedings of IPDPS*, pages 1–8. IEEE, 2007.

[EKS08] Tomás Ebenlendr, Marek Krcal, and Jiri Sgall. Graph balancing: a special case of scheduling unrelated parallel machines. In Shang-Hua Teng, editor, *Proceedings of SODA*, pages 483–490. SIAM, 2008.

[FHZ09] Bin Fu, Yumei Huo, and Hairong Zhao. Exponential inapproximability and FPTAS for scheduling with availability constraints. *Theoretical Computer Science*, 410(27-29):2663 – 2674, 2009.

[FL81] Wenceslas Fernandez de la Vega and George S. Lueker. Bin packing can be solved within $1 + \epsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.

[FN04] Ari Freund and Joseph Naor. Approximating the advertisement placement problem. *Journal of Scheduling*, 7(5):365–374, 2004.

[GG61] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9:849–859, 1961.

[GH62] A. Ghouila-Houri. Characterisation des matrices totalement unimodulaires. *Comptes Rendus de l'Académie des sciences*, 254:1192–1194, 1962.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[GK07] Celia A. Glass and Hans Kellerer. Parallel machine scheduling with job assignment restrictions. *Naval Research Logistics*, 54:250–257, 2007.

[GLMM04] Martin Gairing, Thomas Lücking, Marios Mavronicolas, and Burkhard Monien. Computing nash equilibria for scheduling on restricted parallel links. In László Babai, editor, *Proceedings of STOC*, pages 613–622. ACM, 2004.

[Gra69] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

[HL10] Yumei Huo and Joseph Y.-T. Leung. Parallel machine scheduling with nested processing set restrictions. *European Journal of Operational Research*, 204:229–236, 2010.

[HM84] Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in robotics and VLSI. In Max Fontet and Kurt Mehlhorn, editors, *Proceedings of STACS*, volume 166 of *Lecture Notes in Computer Science*, pages 55–62. Springer, 1984.

[HS76] Ellis Horowitz and Sartaj Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM*, 23(2):317–327, 1976.

[HS87] Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987.

[HSW96] Leslie A. Hall, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line algorithms. In *Proceedings of SODA*, pages 142–151, 1996.

[HvS08] Rolf Harren and Rob van Stee. Packing rectangles into 2OPT bins using rotations. In Joachim Gudmundsson, editor, *Proceedings of SWAT*, volume 5124 of *Lecture Notes in Computer Science*, pages 306–318. Springer, 2008.

[HvS09] Rolf Harren and Rob van Stee. Improved absolute approximation ratios for two-dimensional packing problems. In Irit Dinur, Klaus

Jansen, Joseph Naor, and José D. P. Rolim, editors, *Proceedings of APPROX-RANDOM*, volume 5687 of *Lecture Notes in Computer Science*, pages 177–189. Springer, 2009.

[HvS10] Rolf Harren and Rob van Stee. Absolute approximation ratios for packing rectangles into bins. *Journal of Scheduling*, 2010. To appear.

[JM08] Klaus Jansen and Marian Margraf. *Approximative Algorithmen und Nichtapproximierbarkeit*. de Gruyter, 2008.

[JM09] Klaus Jansen and Monaldo Mastrolilli. Scheduling unrelated parallel machines: linear programming strikes back. Submitted, 2009.

[JP09] Klaus Jansen and Lars Prädel. How to maximize the total area of rectangles packed into a rectangle? Technical Report 0908, Christian-Albrechts-Universität zu Kiel, 2009.

[JPS09] Klaus Jansen, Lars Prädel, and Ulrich M. Schwarz. Two for one: Tight approximation of 2d bin packing. In Frank K. H. A. Dehne, Marina L. Gavrilova, Jörg-Rüdiger Sack, and Csaba D. Tóth, editors, *Proceedings of WADS*, volume 5664 of *Lecture Notes in Computer Science*, pages 399–410. Springer, 2009.

[JS07] Klaus Jansen and Roberto Solis-Oba. New approximability results for 2-dimensional packing problems. In Ludek Kucera and Antonín Kucera, editors, *Proceedings of MFCS*, volume 4708 of *Lecture Note in Computer Science*, pages 103–114. Springer, 2007.

[JSO08] Klaus Jansen and Roberto Solis-Oba. A polynomial time approximation scheme for the square packing problem. In Andrea Lodi, Alessandro Panconesi, and Giovanni Rinaldi, editors, *Proceedings of IPCO*, volume 5035 of *Lecture Notes in Computer Science*, pages 184–198. Springer, 2008.

[JZ07] Klaus Jansen and Guochuan Zhang. Maximizing the total profit of rectangles packed into a rectangle. *Algorithmica*, 47(3):323–342, 2007.

[KK82] Narendra Karmarkar and Richard M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of FOCS*, pages 312–320. IEEE, 1982.

[KK86] Tsuyoshi Kawaguchi and Seiki Kyan. Worst case bound of an LRF schedule for the mean weighted flow-time problem. *SIAM Journal on Computing*, 15(4):1119–1129, 1986.

[Knu05] Donald Ervin Knuth. *MMIX - A RISC Computer for the New Millennium*, volume 1 of *The Art Of Computer Programming*. Addison-Wesley Longman, 2005.

[LL04] Yixun Lin and Wenhua Li. Parallel machine scheduling of machine-dependent jobs with unit-length. *European Journal of Operational Research*, 156(1):261–266, 2004.

[LLP09] Kangbok Lee, Joseph Y.-T. Leung, and Michael Pinedo. A note on graph balancing problems with restrictions. *Information Processing Letters*, 110(1):24–29, 2009.

[LS95] Zhen Liu and Eric Sanlaville. Preemptive scheduling with variable profile, precedence constraints and due dates. *Discrete Applied Mathematics*, 58(3):253–280, 1995.

[LST90] Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.

[LTW+90] Joseph Y.-T. Leung, Tommy W. Tam, C. S. Wong, Gilbert H. Young, and Francis Y. L. Chin. Packing squares into a square. *Journal of Parallel and Distributed Computing*, 10(3):271–275, 1990.

[LW10] Chung-Lun Li and Xiuli Wang. Scheduling parallel machines with inclusive processing set restrictions and job release times. *European Journal of Operational Research*, 200(3):702 – 710, 2010.

[LY74] Jane W. S. Liu and Ai-Tsung Yang. Optimal scheduling of independent tasks on heterogeneous computing systems. In *ACM 74: Proceedings of the 1974 annual conference*, pages 38–45, New York, NY, USA, 1974. ACM.

[McN59] Robert McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6:1–12, 1959.

[MSW10] Gabriella Muratore, Ulrich M. Schwarz, and Gerhard J. Woeginger. Parallel machine scheduling with nested job assignment restrictions. *Operations Research Letters*, 38(1):47–50, 2010.

[OLL08] Jinwen Ou, Joseph Y.-T. Leung, and Chung-Lun Li. Scheduling parallel machines with inclusive processing set restrictions. *Naval Research Logistics*, 55:328–338, 2008.

[PM96] G. N. Srinivasa Prasanna and B. R. Musicus. The optimal control approach to generalized multiprocesor scheduling. *Algorithmica*, 15:17–49, 1996.

[PST95] Serge A. Plotkin, David B. Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20(2):257–301, 1995.

[San95] Eric Sanlaville. Nearly on line scheduling of preemptive independent tasks. *Discrete Applied Mathematics*, 57(2-3):229–241, 1995.

[Sch08] Ulrich M. Schwarz. Online scheduling on semi-related machines. *Information Processing Letters*, 108(1):38–40, September 2008.

[SS98] Eric Sanlaville and Günter Schmidt. Machine scheduling with availability constraints. *Acta Informatica*, 35(9):795–811, 1998.

[Ste97] A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing*, 26(2):401–409, 1997.

[SV05] Evgeny V. Shchepin and Nodari Vakhania. An optimal rounding gives a better approximation for scheduling unrelated machines. *Operations Research Letters*, 33(2):127–133, 2005.

[SWW95] David B. Shmoys, Joel Wein, and David P. Williamson. Scheduling parallel machines on-line. *SIAM Journal on Computing*, 24(6):1313–1331, 1995.

[Tur37] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937.

[Ull75] Jeffrey D. Ullman. NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10(3):384–393, 1975.

[Vaz01] Vijay V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., 2001.

[Woe02] Gerhard J. Woeginger. Open problems in the theory of scheduling. *Bulletin of the EATCS*, 76:67–83, 2002.

[Zha05] Guochuan Zhang. A 3-approximation algorithm for two-dimensional bin packing. *Operations Research Letters*, 33(2):121–126, 2005.

Curriculum vitæ

The author was born July 2nd, 1980, in Aurich (Ostfr.). He went to school in Oldenburg and obtained his *Abitur* there in 1999, with a grade of 1.3. From 2000–2006, he was a student of computer science at the Christian-Albrechts-Universität at Kiel; he was awarded the degree of Diplom-Informatiker; graded "with distinction", on April 11th, 2006.
From 2006 on, he was employed at Christian-Albrechts-Universität as a Ph.D. student in the group of Prof. Dr. Klaus Jansen; first in EU project AEOLUS, later as regular staff. On July 1st, 2010, he successfully defended the present thesis with an overall evaluation of "Magna cum laude".