UNIVERSITE
PARIS-SUD XI

Lasse Rempe

A Limit Problem for Online Scheduling

Rapport de Stage, DEA Algorithmique

Supervisor: Prof. Claire Kenyon

Stony Brook
15th November 2002

Laboratoire de Recherche en Informatique

Université de Paris-Sud, Orsay

A Limit Problem
for Online Scheduling

Résumé

Nous étudions le problème d'ordonnancement en-ligne sur m processeurs. Après avoir donné un résumé des résultats connus sur le problème, nous proposons deux approches qui nous trouvons utiles pour l'étude du problène,

D'une part, nous proposons de modifier la définition de compétitivité en utilisant des bornes inférieures simples sur la valeur de l'optimum. Cette approche se justifie par le fait que tous les résultats obtenus de par le passé utilisent uniquement des bornes inférieures très simples.

Nous développons également une "version limite" du problème, qui simplifie la terminologie pour m grand, et discutons de ses propriétés de base. Enfin, nous proposons un algorithme qui, pour m grand, pourrait avoir un rapport de compétitivité meilleur que les bornes précédemment connues.

Abstract

We give an overview of recent results on the m-machine online scheduling problem. Furthermore, we propose two programs that we believe will be useful in the study of this problem.

On the one hand, we propose the use of simple lower bounds on the optimum makespan in the definition of competitiveness, and study the resulting problem. This approach is supported by the fact that all results obtained so far use only the simplest of such bounds.

We also develop a limit version of the problem that provides a simpler language to discuss the case of large m, and discuss its basic properties. Furthermore, we propose an algorithm that might give better bounds for large m.

Contents

Introduction

We study the online version of the multiprocessor scheduling problem. This problem was first introduced by Graham in [G1]. Given a number m of processors and a sequence of jobs, each with a job length $\ell_i > 0$, the scheduler has the task of distributing these jobs on the processors. The completion time of a processor is then the sum of all job lengths this processor has been assigned. The goal is to minimize the total completion time, or *makespan*; that is, the largest processor completion time.

This problem is NP-hard, although for fixed m, there exists a polynomial time approximation scheme. We will give proofs of these classical results in Sections 1.4 and 1.5.

This thesis is, however, dedicated to the study of the *online* version of the problem. Here the jobs arrive one after another, and the algorithm must assign each job to a processor without knowledge of any further jobs to come. The goal is to do this in such a way that the *competitive ratio*, i.e. the ratio between the makespan of the schedule and the optimal offline makespan, is as small as possible. Note that many variants of this simplest scheduling problem have been defined, on which we will not comment in this report. For a comprehensive overview, see [S2].

Graham gave a simple heuristic for scheduling, the now famous "list scheduling" algorithm, which has as competitive ratio of $2 - \frac{1}{m}$. An family of algorithms whose competitive ratio stays away from 2 as m gets large was first presented in [BFK] and subsequently improved, up to a ratio of 1.9201 [KPT, A1, FW].

In this report, we make two contributions to the study of online algorithms, particularly for large m. On the one hand, we suggest that, instead of considering the ratio of the constructed makespan to the *optimal* makespan, we consider the ratio to some elementary lower bound to the makespan. This has been inspired by the fact that results about online scheduling have always relied on such bounds, as well as the fact that it seems the difficulties for an online algorithm rely much more on certain characteristics of input sequences that are caught by these bounds rather than the fine structures of these job sequences that result in NP-hardness of computing the optimum.

As this thesis neared completion, we have learned that this approach has, in fact, been recently used to great profit by Susanne Albers [A2]. In this article, Albers presents a lower bound of 1.919 on the competitive ratio of an online algorithm with respect to such bounds, which is a dramatic improvement over the previously known lower bound of 1.88 from [R].

On the other hand, we introduce a limit problem for the scheduling problem as the number of processors increases. To do this, we replace the discrete set of processors by the continuum $[0, 1]$. This facilitates many discussion about scheduling for large m. Algorithms are allowed to split jobs in a way most beneficial to their goal, which removes many problems of rounding which turn up in the analysis for finite m. On the other hand, job sequences that produce a lower bound on the makespan of an online algorithm can be more naturally formulated, because in a job sequence the length of the job may be allowed to vary continuously.

While we are not able to prove an automatic correspondence between algorithms for the finite problems and for the limit problem, such a correspondence exists for many important cases. In particular, all known upper and lower bounds descend to the limit problem, including the bound from [A2].

The first algorithm [BFK] which achieved a competitive ratio better than two (in the limit) had — as was pointed out in [KPT] — the flaw of being rather unnatural. At any time, it would only schedule jobs on one of two possible processors: the one with the least load, or the one with the .44th smallest load. While this was removed in [KPT] the better algorithms of [A1] and [FW] again show this type of behavior.

In the conclusion, we will present an algorithm which, while following the same philosophy as [A1], has no such restriction. However, this algorithm has not yet been analyzed.

Acknowledgements My attendance at the DEA Algorithmique program at Orsay during the fall semester 2002 was financed through the Informatik-Preis of the Datenzentrale Schleswig-Holstein, Germany. I wish to thank the Department of Computer Science in Kiel and the Datenzentrale for granting me this opportunity. I also thank the Institute for Mathematical Sciences at the State University of Stony Brook for its ongoing support and hospitality. I thank my advisor, Prof. Claire Kenyon, for inviting me to Orsay, introducing me to the online scheduling problem, countless productive discussions of the subject as well as immense help with questions regarding the DEA program. Many thanks go to Jéremy Barbay for all the things he helped me with during my stay at Orsay. I also wish to thank Klaus Jansen in Kiel and Michael Bender in Stony Brook for interesting discussions. Finally, I thank my girlfriend, Emma Gillen, and my parents for their love and support.

Chapter 1

Scheduling: An Overview

1.1 Definitions

In the following, $m \geq 2$ will always be a positive integer, the number of machines. A *processor* is a number between 1 and m.

1.1.1 Definition (Basic Definitions)
A job *is a positive real number; a* job sequence *is a finite sequence* $\ell_1, \ldots, \ell_s$ *of jobs. The total load* L *and the average load* A *of a job sequence are defined to be* $L := \sum \ell_i$ *and* $A := L/m$.

A schedule *of a given job sequence assigns each job* t_i *to exactly one processor. The load of machine* k *in this schedule is* $\ell(k) := \sum_{i:i \text{ assigned to } k} \ell_i$. *The* makespan *of the schedule is* $M := \max_k \ell(k)$. *The* optimum makespan *is* $OPT := \min M$, *where the minimum is taken over all schedules.*

Given a schedule, we shall usually assume the machines ordered by non-increasing load, i.e. $M = \ell(1) \geq \ell(2) \geq \cdots \geq \ell(m)$.

1.1.2 Definition (Scheduling Problem)
The Scheduling problem consists of determining, given m *and a job sequence, a schedule that minimizes the makespan. The* m*-machine scheduling problem is the scheduling problem with* m *fixed.*

We review NP-Hardness and approximation results for the scheduling problem in Sections 1.4 and 1.5. Let us now turn to the online version of the problem, which is the main concern of this report.

1.1.3 Definition (Online Scheduling Algorithms)
An online algorithm for the scheduling problem assigns each job j_i *to a machine without knowledge of the following jobs* $j_{i+1}, \ldots, j_s$. *For any online algorithm* A *and a schedule* J, *we denote by* $A(J)$ *the makespan of the schedule of* J *produced by* A.

1.1.4 Definition (Competitive Ratio)
Let A be an online algorithm for the scheduling problem. Then

$$\sigma(A) := \sup_J \frac{A(J)}{OPT(J)}$$

is called the competitive ratio *of A. A is called C-competitive if $\sigma(A) \leq C$. The number*

$$\sigma(m) := \inf_A \sigma(A)$$

is called the optimal competitive ratio *for the m-machine online scheduling problem.*

1.2 Generalized Competitive Ratios

In this work, we examine a concept which, to our knowledge, has not been introduced into the study of online scheduling so far. The idea relies on the fact that, until now, the proof of competitiveness for any online scheduling algorithm has always involved proof of a stronger property, namely competitiveness with respect to elementary lower bounds on the makespan. We have decided to see this as a virtue rather than a vice, and believe that it will be useful to restrict to this notion in the analysis of the problem.

This idea does not seem to be new in the world of online algorithms. For example, to analyze bin-packing, Johnson et al. [CGJ] study not the ratio to the optimal packing, but the ratio to the sum of the sizes. Similarly, for analyzing algorithms for the Travelling Salesman Problem, Johnson et al. study not the ratio to the optimal tour, but the ratio to the Held-Karp lower bound.

1.2.1 Definition (Makespan Lower Bounds)
A function B that assigns to each job sequence J a number $B(J) > 0$ such that $B(J) \leq OPT(J)$ is called a makespan lower bound.

We will describe a sequence of elementary lower bounds on the makespan that have been useful in the study of online scheduling. Consider any job sequence $J = (t_1, \ldots, t_s)$, and let $r > 0$. Surely, if we have at least $m+1$ jobs of length $\geq r$, then the makespan is at least $2r$. Similarly, if there are at least $2m+1$ jobs of length $\geq r$, then the makespan is at least $3r$ etc. This will already give rise to a sequence of lower bounds; however, let us refine them a little bit. If there are $2m$ jobs of length r and one of length $2r$, then the makespan is still at least $3r$; i.e., we should count jobs which are of size $\geq nr$ as n jobs of length r. This leads us to define:

1.2.2 Definition (Elementary Bounds)
Let $j \geq 1$; we define

$$b_j(J) := j \cdot \max\{r : \sum_i \left\lfloor \frac{t_i}{r} \right\rfloor \geq m(j-1)+1\}.$$

We also define $B_0(J) := A$ *and inductively*

$$B_{j+1} := \max\{B_j, b_{j+1}\}.$$

We set $B_\infty(J) := \sup B_j(J)$.

In fact, we can still sharpen these bounds a little bit:

1.2.3 Definition (Sharpened Elementary Bounds)
We define for $j \geq 1$*:*

$$b'_j := \min\left\{t_{i_1} + \cdots + t_{i_k} : \exists r > 0 : \sum \left\lfloor \frac{t_i}{r} \right\rfloor \geq m(j-1)+1 \text{ and } \sum \left\lfloor \frac{t_{i_\nu}}{r} \right\rfloor \geq j \right\}.$$

Note that b'_j *will not be defined for arbitrary* j*, in which case we may set* $b'_j := 0$.

As before, define $B'_0 := B_0 = A$ *and inductively*

$$B'_{j+1} := \max\{B'_j, b'_{j+1}\}.$$

1.2.4 Fact
For any job sequence J,

$$B_0(J) \leq B_1(J) \leq B_2(J) \cdots \leq B_\infty(J) \leq OPT(J);$$

so in particular each B_i *is a makespan lower bound. The same is true with* B *replaced by* B'*. Furthermore, there exists a sequence* J *with* $B'_\infty(J) < OPT(J)$.

Proof. The makespan is certainly larger than B_0, the average load. Now let $i > 0$, and let $r > 0$. Create a new job sequence by replacing any job of length l by $\lfloor \frac{l}{r} \rfloor$ jobs of length r. The optimum for this sequence is certainly smaller than that for the original sequence. If there are, in all, at least $j(m-1)+1$ jobs in the new sequence, then by the pigeon hole principle the makespan of the new sequence is at least jr. This proves the claim for B_i.

The proof for B'_i is analogous. By the same argument as before, in any schedule there has to be a processor whose loads $l_1, \ldots, l_k$ satisfy $\sum \lfloor \frac{l_i}{r} \rfloor \geq j$. So the makespan is at least the sum of the loads of any such configuration of jobs.

Now let m be even. Consider the job sequence J_m consisting of $\frac{m}{2}$ jobs of length 10, $\frac{m}{2}$ jobs of length 9, m jobs of length 6 and $\frac{m}{2}$ jobs of length 4. This sequence has optimal makespan 18 if m is divisible by 4, and 19 otherwise. Then $B'(J_m) = 18$, and so for m not divisible by 4, $B'(J_m) < \mathrm{OPT}(J_m)$. ∎

1.2.5 Definition (Generalized Competitive Ratio)
Let B *be a makespan lower bound. If* A *is any algorithm, then*

$$\sigma_B(A) := \sup_J \frac{A(J)}{B(J)}$$

is called the B*-competitive ratio of* J*. We define*

$$\sigma_B(m) := \inf_A \sigma_B(A) \ (\geq \sigma(m)).$$

REMARK. The bound $B_1 = \max\{A, t_1, \ldots, t_s\}$ is certainly the simplest bound one can use in the analysis of online-scheduling algorithms. However, this bound is too weak to be a useful tool. Indeed, consider a sequence J containing $m + 1$ jobs of unit length. Then $\mathrm{OPT}(J) = 2$ but $B_1(J) = \frac{m+1}{m}$; in other words, even an algorithm with B_1-competitive ratio 1 does not need to be more than $2 - \frac{1}{m}$ competitive, which is achieved by the first scheduling algorithm ever, Graham's greedy "List Scheduling" (Section 1.6). Consequently, the bound B_2 (or rather, in fact, a somewhat weaker version of this bound) was introduced, and to the present day it is the strongest bound that has been used to prove OPT-competitiveness.

Of course a restriction to B-competitiveness for some weaker bound than OPT might increase the competitive ratio. However, one might argue that the features that make the optimum hard to compute are much too subtle to influence the performance of an online algorithm; compare also Section 1.4. We therefore believe that it is a good idea, for the time being, to restrict study of the scheduling problem to the class of B_∞-competitive algorithms, or even to some of the weaker lower bounds. It is much easier to develop lower bounds for this class of algorithms. As we noted in the introduction, recently Susanne Albers has followed just this path in proving a lower bound on the $\tilde{B}_2$-competitive ratio of deterministic online algorithms [A2]. As discussed above, it might very well turn out that this is not a real restriction.

1.2.6 Open Question
Is $\sigma(m) = \sigma_{B_\infty}(m)$ for all m?

1.3 Lower Bounds on Competitive Ratios

An *adversary* or *taskmaster* is an algorithm that creates an input sequence for the online scheduling, where it may react to the choices of the algorithm on past jobs when creating future jobs. Thus online scheduling can be seen as a game between the taskmaster and the online algorithm: the taskmaster tries to achieve, at some time, a large ratio between the current schedule and the optimum, whereas the online algorithm must try to keep this ratio low at all times.

1.3.1 Definition (Forcing a Competitive Ratio)
An adversary forces *a competitive ratio of c if, for every online algorithm playing against this adversary, there exists a time when the makespan of the schedule computed by the algorithm has a ratio of at least c to the optimum.*

An adversary can be thought of as a finite or infinite tree each of whose internal nodes is labeled by a job length $l > 0$ and has m children: one for each possible choice of the algorithm of where to put the job of length l.

It is sufficient to consider *finite adversaries*; i.e. such for which the corresponding tree is finite. To see this, note that we can end the tree at any leaf at which a competitive ratio of at least c has been reached. By the definition of an adversary, every possible (infinite) path descending from the root will, at some point, have a competitive ratio of at least c. Therefore, in the "pruned" tree, all paths are finite.

Note that any job sequence $J = (t_1, \dots, t_s)$ can be considered as an adversary that is independent of the algorithm's choices.

1.3.2 Definition (Forcing Sequence)
A sequence is called a c-forcing sequence, if, as an adversary, it forces a competitive ratio of c.

As we have suggested studying generalized competitive ratios, it makes sense to also extend these definitions to makespan lower bounds other than OPT. If we replace OPT in the previous definition of forcing by some makespan lower bound B, we will say that the corresponding adversary forces a B-competitive ratio of c.

EXAMPLE. In the last section we remarked that restricting to the bound B_1 can never prove a competitive ratio less than $2 - \frac{1}{m}$. In our new terminology, we can phrase the argument more concisely: The job sequence consisting of $m+1$ jobs of equal length forces a B_1-competitive ratio of $2 - \frac{1}{m}$.

Let us turn to an even more restricted class of adversaries, so called "layered forcing sequences". In fact, all known forcing sequences used to prove lower bounds on the competitive ratio are layered forcing sequences.

These sequences are divided into "layers" of m jobs. Usually the job lengths are nondecreasing in each layer, though this is not necessary. The point of a layered forcing sequence is that an algorithm will have to schedule all m jobs of the same layer on different machines in order not to force a competitive ratio of c. After several layers, the sequence ends with a long job that will force the competitive ratio even if all jobs from every layer *were* scheduled on different machines.

1.3.3 Definition
If B is any makespan lower bound, we define

$$\mu_B(m) := \sup\{B\text{-}\textit{competitive ratios induced by layered forcing sequences}\}.$$

We also set $\mu(m) := \mu_{OPT}(m)$.

Considering only layered forcing sequences seems to be a rather severe restriction. However, it is conceivable that this is not true. All lower bounds that have been obtained until this date were proved by more and more sophisticated forcing sequences, reaching their climax in John Rudin's PhD thesis [R]. In general, it seems difficult to construct an adaptive adversary (that chooses jobs depending on the algorithm's choices) that forces a ratio that could not have been obtained using a forcing sequence. The reason is that these bounds must really on forcing the algorithm into building a "flat" schedule, and if the algorithm is allowed to make choices that destroy the flatness of this schedule, the adversary will usually have to build up this flatness again. Therefore it seems reasonable to ask the following counterpart of Problem 1.2.6.

1.3.4 Open Question
Is $\mu(m) = \sigma(m)$? Is $\mu_{B_\infty}(m) = \sigma_{B_\infty}(m)$?

We should note that if $\mu(m) = \sigma(m)$, this would have a rather surprising consequence. Since the adversary does not take actions of the algorithm into account, one could allow the algorithm the knowledge of all jobs beforehand. This would mean that the difficulty of online scheduling lies *only* in the fact that the order of scheduling is fixed, and the schedule has to be good at any point in time, rather than that the algorithm does not know of any following jobs. This would be a rather strong and somewhat surprising result.

1.4 NP-Hardness

In this section, we prove NP-hardness of the scheduling problem, and, in the process, examine the equivalence between the offline scheduling problem and the well-known offline bin-packing problem. As the online versions of these problems are very different, this is an indication that the difficulties for online algorithms are fundamentally different from those in the offline case.

For this section only, we will consider the scheduling problem as a decision problem: The Scheduling Decision problem is to decide, given a sequence of jobs J, a number m of processors and a goal G, whether $\mathrm{OPT}(J) \leq G$. The m-processor version is the same problem with fixed m. Of course any algorithm for the minimization problem also solves the decision problem. We shall prove the following two classical theorems.

1.4.1 Theorem (Scheduling is NP-Complete)
The scheduling problem is strongly NP-complete.

1.4.2 Theorem (m-Scheduling is NP-Complete)
For any $m \geq 2$, the m-processor scheduling problem is NP-complete.

First Proof. We reduce scheduling to the well-known bin packing problem. The *bin packing problem* is to decide, given a sequence I of items, a "bin size" B and a goal G, whether the items can be distributed into at most G bins of maximum capacity B. The m-bin packing problem is the same problem, with a fixed number m of bins. Of course this is just the same as the scheduling problem, where the number of processors is the number of bins and the capacity is the goal makespan. Thus we have the following result.

1.4.3 Theorem
The scheduling problem is equivalent to the bin packing problem. The m-processor scheduling problem is equivalent to the m-bin-packing problem.

Second Proof. We prove NP-Hardness of scheduling (and bin packing) by reducing two well-known partitioning problems to them.

The *partitioning problem* is as follows. The algorithm is given a list of numbers and must decide whether these can be partitioned into two subsets such that the sums of the numbers in each set are equal. This problem is NP-complete [K]. It clearly is a special case of the 2-processor scheduling problem, and thus of the m-processor scheduling problem.

(Any instance (J,G) of the m-processor scheduling problem can be extended to one of the m'-processor scheduling problem, $m' > m$, by adding $m - m'$ jobs of length G.)

To prove strong NP-completeness, note that the *3-partitioning problem* is a special subproblem. This problem is something of a prototype of strongly NP-complete problems: Given a goal B and a set S of $3m$ numbers between $\frac{B}{3}$ and $\frac{B}{4}$, the algorithm has to decide whether S can be partitioned into m sets of cardinality 3 such that the sum of the three elements in each set is B. ■

1.5 A Polynomial Approximation Scheme

Hochbaum and Shmoys [HS] proved the following theorem:

1.5.1 Theorem
There is a PTAS for the scheduling problem.

This PTAS first introduced the dual approximation approach for strongly NP-hard problems. We will be content to prove the following, much more basic result.

1.5.2 Theorem
There is a PTAS for the m-processor scheduling problem.

PROOF. Let $k = \lceil \frac{1}{\varepsilon} \rceil$. The algorithm proceeds in two steps.

1. Schedule the largest $km - 1$ items optimally, using exhaustive search.
2. Schedule the rest of the jobs one-by-one in a greedy fashion, assigning every job to the machine with the currently lowest load.

Analysis.

In step one, we perform an exhaustive search over all possible schedules for the largest $km - 1$ items. This is exponential in k; however, the step is constant for fixed ε. m^{km-1}, is constant for fixed m and ε. The running time of the greedy heuristic in the second step is clearly polynomial.

It remains to prove that the algorithm obtains a schedule of makespan at most $(1 + \varepsilon)\text{OPT}$. Denote the makespan of this schedule obtained in step 1 by t', and let $A = \frac{L}{m}$ be the average load of the jobs sequence as usual. Define $t := \max\{t', A\} \leq \text{OPT}$.

Let ℓ be the kmth largest load in the sequence. By definition, every job scheduled in step 2 has length at most l. Since $\ell km \leq L$,

$$\ell \leq \varepsilon \frac{L}{m}.$$

Scheduling jobs of length at most ℓ in a greedy fashion then clearly results in a makespan of at most $t' + \ell \leq \text{OPT} + \varepsilon\text{OPT}$. ■

1.6 List Scheduling

Graham [G1] introduced the now classic greedy algorithm for online scheduling known as "list scheduling" (LS). The algorithm follows the following simple rule:

Schedule any job on a processor that has minimal load in the current schedule.

1.6.1 Proposition

List Scheduling is $2 - \frac{1}{m}$ *competitive. More precisely,*

$$\sigma(\mathrm{LS}) = \sigma_{B_1}(\mathrm{LS}) = 2 - \frac{1}{m}.$$

Proof. Consider the addition of a job J of length l, which is being scheduled on processor p_i. Denote the average load *before* the scheduling of l by A. Since p_i has minimal load, this load is at most A. If B_1 is the the elementary bound for the schedule *including* l, then

$$l \leq B_1$$

and

$$A + \frac{l}{m} \leq B_1$$

Thus the load of p_i after the addition of l is at most

$$A + l = A + \frac{l}{m} + l\frac{m-1}{m} \leq (1 + \frac{m-1}{m})B_1 = (2 - \frac{1}{m})B_1.$$

Thus $\sigma_{B_1}(\mathrm{LS}) \leq 2 - \frac{1}{m}$.

As noted before, no algorithm can be better than $2 - \frac{1}{m}$ competitive with respect to the bound B_1. To prove that List Scheduling is also no better than $2 - \frac{1}{m}$ OPT-competitive, consider a job sequence consisting of $m(m-1)$ jobs of length 1 and one of length m. The optimal schedule has makespan m, whereas the algorithm obtains a schedule of makespan $2m - 1$. ■

1.6.2 Theorem

$\sigma(2) = \frac{3}{2}$ *and* $\sigma(3) = \frac{5}{3}$. *In particular, List Scheduling is optimal for* $m \in \{1, 2\}$.

Proof. For $m = 2$, the job sequence $(1, 1, 2)$ is a $\frac{3}{2}$ forcing sequence. Indeed, if the algorithm wishes to maintain a ratio less than 2, it must schedule the first two jobs on different machines. Therefore the last job will be scheduled on a machine which has already received a job of length 1, giving a makespan of 3 when the optimal makespan is 2. Thus $\sigma_2 \geq \frac{3}{2}$.

For $m = 3$, consider the job sequence $J = (1, 1, 1, 3, 3, 3, 6)$. This sequence is a $\frac{5}{3}$ forcing sequence. Again, the three first jobs have to be scheduled on different machines in order to obtain a competitive ratio $\leq \frac{5}{3}$. The same goes for the three jobs of length 3: If two of them were scheduled on the same machine (which already contains one job of length 1), the makespan would be at least 7. However, the optimum makespan after the first six jobs

m	lower bound	upper bound
2	3/2	3/2
3	4/3	4/3
4	$\sqrt{3} \approx 1.732$	1.7333
5	1.7483	1.7708
$\vdots$	$\vdots$	$\vdots$
∞	1.88	1.9201

Figure 1.1: Known Bounds for Deterministic Online Scheduling

is clearly 4, and $\frac{7}{4} > \frac{5}{3}$. So again, in the end one processor will contain one job of length 1, 3 and 6 each, where the optimum is 6. So $\sigma_3 \geq \frac{10}{6} = \frac{5}{3}$. ∎

$m = 2$ and $m = 3$ are the only cases in which List Scheduling is optimal, and also the only cases in which an optimal algorithm is known. John Rudin has given a lower bound of $\sqrt{3}$ for $m = 4$; this bound is generally believed to be sharp. Figure 1.1 gives an overview of some of the known bounds for deterministic online scheduling.

As noted before, Susanne Albers [A2] recently proved a lower bound of 1.919 on the B_2-competitive ratio for large m.

1.7 Remarks on Randomized Online Scheduling

In this report, randomized online scheduling is not the focus of attention; however, we should make some remarks on the situation here. For $m = 2$, an optimal algorithm is known [BFK]; its competitive ratio is $\frac{2}{3}$. (To define the competitive ratio, the expected makespan is considered.)

There is a general lower bound of

$$1 + \frac{1}{\left(\frac{m}{m-1}\right)^m - 1}$$

for the randomized m-processor scheduling problem [S1]. This bound was long believed to be optimal.

However, Tomáš Tichý recently proved the surprising result that, already for $m = 3$, no randomized online algorithm can achieve this ratio [T1, T2]. However, the proof does not rule out the possibility that this ratio can be reached up to ε, for any ε. The reason for the impossibility of the ratio is, in a very strong way, a strong restriction on the scheduling behavior of an algorithm which wants to achieve the ratio. If such an algorithm is faced with a sequence in which, at each point in time, the length of the last job equals both the average area A of all jobs to that time and the optimum makespan, it basically has no meaningful choices left and cannot exploit randomization at all. In fact, these sequences are also used to prove the lower bound in the first place.

For a long time, no randomized algorithm was known for large m that performs better than the best known deterministic algorithms. In the recent breakthrough in [A2], S. Albers presented an algorithm that achieves just that; it is 1.915-competitive. As noted above, the same article proves that $\limsup \mu_{\tilde{B}_2} \geq 1.919$. However, the ratio of 1.915 is still far from the known lower bounds.

Even though we do not discuss randomized online scheduling any more in this report, the principles we propose can also be applied in this setting. The restriction to makespan lower bounds is certainly as useful in the randomized as in the deterministic case. Also, the limit problem we will introduce in the next chapter can be used to discuss randomized online scheduling for large m. In fact, the lower bound of $\frac{e}{e-1}$ can be nicely deduced in the language of the limit problem.

Chapter 2

Continuous Scheduling Problems

2.1 Overview

In this chapter, we propose a limit problem for the scheduling problem as $m \to \infty$. The purpose of introducing this limit problem is to simplify the discussion of online scheduling for large m through the ability to work directly on the limit problem, and also to separate some of the difficulties that appear as m grows large.

The idea is to replace the discrete set of m processors, in the limit, by the interval $[0, 1)$ of "processors". However, there are many possible choices with respect to how far one goes in removing the discrete nature of the problem. We will therefore introduce the limit problem in several versions. All of these versions are equivalent in the offline version. This is conjecturally true also in the online case. These versions differ by the form in which the input is given, as well as in the way the algorithm is allowed to schedule. Depending on the context, some of these versions will be more convenient to use than others.

2.2 Definition

Let us start the process of removing the discrete nature of the scheduling problem by considering the following problem:

2.2.1 Definition (Rigid Continuous Scheduling Problem)
In the rigid continuous scheduling problem, the algorithm is handed a sequence of rectangles which have to be "dropped" into the interval $[0, 1]$, *where we assume that the endpoints* 0 *and* 1 *are identified. More precisely, a* configuration *is a piecewise constant function* $f : [0, 1] \to \mathbb{R}^+$. *Each rectangle of width* $\delta < 1$ *and height* t *is to be assigned to a starting processor* x_0, *and the resulting configuration is*

$$\tilde{f}(x) = \begin{cases} f(x) + t & x \in [x_0, x_0 + \delta) (\text{mod } 1) \\ f(x) & \textit{otherwise} \end{cases}$$

The goal is to minimize the makespan of the configuration; i.e. $\max_{x \in [0,1]} f(x)$.

The online version is defined in the obvious way.

This problem, in full generality, is quite different from the scheduling problems because geometric information becomes difficult, while in the scheduling problems processors can always be reordered. We mention it here only because it can sometimes serve as a simplification for talking about the *original* scheduling problem, when all jobs have width $\frac{1}{m}$. This can be useful if one does not wish to worry about placing the jobs exactly at a multiple of $\frac{1}{m}$:

2.2.2 Lemma
When restricted to sequences of rectangles of width $\frac{1}{m}$, the rigid continuous scheduling problem reduces to the m-processor scheduling problem.

PROOF. Clearly any algorithm for the m-processor problem is also one for the restricted continuous problem. Conversely, if any algorithm A for the continuous version is given that achieves a competitive ratio σ, obtain an algorithm B for the m-machine problem that achieves the same ratio as follows. A executes B on the given schedule, but each time a job is assigned to a processor x_0, assign it to the processor $\frac{\lceil mx_0 \rceil}{m}$ instead. When we compare the A schedule f and the B schedule g, we see that $f(\frac{i}{m}) = g(\frac{i}{m})$. On the other hand, g is constant on each interval $[\frac{i}{m}, \frac{i+1}{m})$, so $\max g \leq \max f$. ■

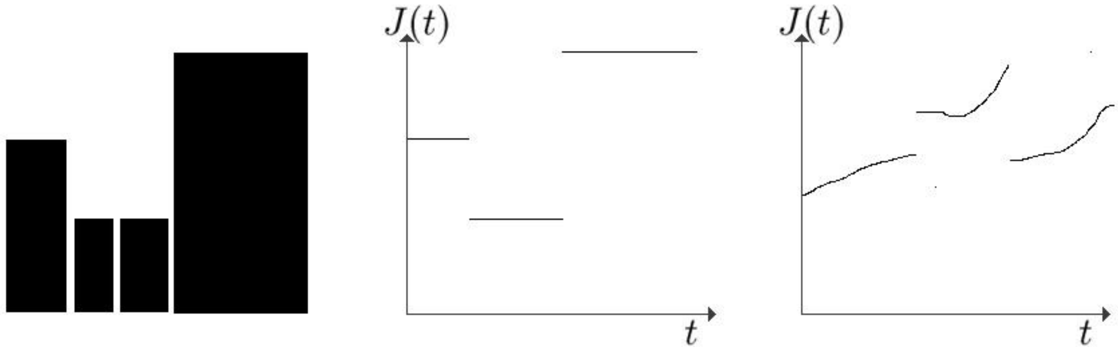
In order to pass to a limit for the scheduling problem as $m \to \infty$, we need to consider what the input for the problem is meant to be. A possible definition was already used implicitly in the previous section:

2.2.3 Definition (Block Sequence)
A job block *(or just* block *is a rectangle of some length (or height) $h > 0$ and width $w > 0$, $w \in \mathbb{Q}$. A* block sequence *is a finite sequence of jobs. A block sequence is called an m-*block sequence *if the width of every block is a multiple of $\frac{1}{m}$.*

Any job sequence for the m-processor scheduling problem (in the traditional sense) is an m-block sequence: just let every job have width $\frac{1}{m}$.

While this definition may be useful for a discussion that is close to the original scheduling problem, it seems still rather discrete and therefore somewhat unnatural for something that will be considered a limit problem. It is more intuitive not to differentiate between one block of length h and width r and three consecutive blocks of length h and width $\frac{r}{3}$. The natural way to do this is to simply replace the notion of a block sequence by that of a step function.

2.2.4 Definition (Job Step Function)
A job step function *(or just* step function*) is a function $J : [0, s) \to \mathbb{R}$ for which there are finitely many rational points $0 = t_0 < t_1 < \cdots < t_n = s$ such that J is constant on each $[t_i, t_{i+1})$.*



(a) Block Sequence (b) Job Step Function (c) Continuous Job Function

Figure 2.1: Three different models for job sequences

Remark. Every block sequence has an associated step function: and vice versa. We will refer to the t_i as *job boundaries.*

However, at times this might still seem to be an unnatural restriction, especially when trying to construct lower bounds. If we are striving for a continuous version of the scheduling problem, we might as well let the jobs, as they come in, vary continuously. This leads to the following definition.

2.2.5 Definition (Continuous Job Function)
A continuous job function *is a function $J : [0, s) \to \mathbb{R}$ for which there are finitely many rational points $0 = t_0 < t_1 < \cdots < t_n = s$ such that J is uniformly continuous on each $[t_i, t_{i+1})$, and such that $\inf_t J(t) > 0$.*

Every job step function is also a continuous job function. See figure 2.1 for an illustration of the three models. We will abbreviate these models as B-, S- and C-inputs (the letters stand for “block”, “step”, and “continuous). Any of these inputs will be called a “job sequence”.

After defining three notions of job sequences, we should now turn to the question of what it means to *schedule* these jobs. The idea is that we are allowed to split these jobs arbitrarily and then schedule them on our continuous set of processors. Again, there are two ways in which we can make the definition; one closer to the discrete case and one in the spirit of non-discrete methods.

As the continuous job function is the most general notion, we will make the definition of a schedule for these.

2.2.6 Definition (Discrete Schedule)
If J is a continuous job function, then a discrete schedule *or* Δ-schedule *for J is given by a finite set of pairs $(t_1, x_1), \ldots, (t_n, x_n) \in [0, s) \times [0, 1)$ with $0 = t_1 < t_2 < \cdots < t_n < t_{n+1} = s$.*

The configuration *of this schedule is the function* $C : [0,1) \to \mathbb{R}$,

$$C(x) := \sum_{i : x \in [x_i + t_{i+1} - t_i)} J(x - x_i + t_i).$$

The makespan *of this schedule is* $\sup_x C(x)$.

The second version of the defintion of a schedule may seem somewhat complicated, but often turns out to actually be easier to work with.

2.2.7 Definition (Continuous Schedule)
A continuous schedule *or* Γ-schedule *for J is a Borel function $S : [0,s) \to [0,1]$ which is locally area preserving; i.e., $\lambda(S(a, a+\varepsilon)) = \varepsilon$ for any a and ε small enough, where λ is Lebesgue measure.*

The corresponding configuration *is the function $C : [0,1] \to \mathbb{R}$ which satisfies $\int_a^b C = \int_{S^{-1}(a,b)} J$. The* makespan *of the schedule is the essential supremum of C.*

Remark. Every Δ-schedule has a corresponding Γ-schedule: If $t \in [t_i, t_{i+1})$, then $S(t) := x_i + t - t_i$.

We can now define continuous versions of the scheduling problem.

2.2.8 Definition (Continuous Scheduling Problem)
Let $x \in \{B, S, C\}$ and $\xi \in \{\Delta, \Gamma\}$. Then the $(x|\xi)$-scheduling problem *has as input a job sequence J of type x. The goal is to produce a ξ-schedule for J such that the makespan is minimized.*

In the online version of the problem, the schedule up to time t_0 may only depend on $J|[0,t_0)$, where t_0 ranges over the job boundaries in the case $a = \mu$, and over all of $[0,s)$ otherwise.

We have thus defined six different candidates for a limit problem. In the following, we will show that their offline versions are all equivalent. We believe that the same is true in the online case, but have so far been unable to establish this fact. This will be discussed in the next section.

Let us elaborate on the definition of the online versions, as there are some clarifications to be made. If the input is a block sequence, then the situation is fairly simple. The algorithm determines the schedule one block at a time, without using information about any following blocks. However, the algorithm is allowed to take the length of the last job into account. If the input is a step function, then the schedule must be built continuously, i.e., the length of the steps is not known at the beginning of a step.

If the input is of C-type, *then we assume that at the beginning of each interval of uniform continuity, the constants for uniform continuity is known.* (In most cases, this might simply mean giving the algorithm a bound on the derivative on this interval.) Note, however, that the length of this interval is *not* known.

Let us remark on the nature of the different online problems, for it is here that the differences really appear. An online algorithm that receives block sequences as input can, in deciding how to split a block, it use the information about this blocks width. An online algorithm which has a step function as input, however, must start scheduling whenever a new step appears, without any information on the width of this step.

A similar effect occurs when we compare discrete and continuous schedules. An online algorithm that produces a Γ-schedule might, at the beginning of a step, start distributing this job on different areas in the schedule, whereas if it had to produce a Δ-schedule, then — if the step is small enough — all of the job will be scheduled at the same point of the schedule.

As a final comment, note that, for these problems, an "algorithm" is nothing but a function that associates schedules to job sequences of the corresponding type. When $x \neq C$ and $\xi \neq \Gamma$, one could still ask the question of computability; in the fully continuous versions, this question does not even make sense any longer. The questions asked, however, are about possibility and impossibility of such an assignment, and we will not consider computability considerations. On the other hand, all methods used here will easily give actually applicable methods for the discrete cases.

2.3 Equivalence of the Limit Problems

Let us show that the offline versions of the continuous scheduling problems are all equivalent. This process involves two steps. First, we see that, for any job sequence, we can convert any Γ-schedule to a Δ-schedule with the same makespan. Then we will see that we can discretize a job flow to a job sequence that is not too different. (Note that there is obviously no difference between B- and S-inputs in the offline case!)

2.3.1 Proposition
If $J = (R_1, \ldots, R_k)$ is a block sequence and S is a Γ-schedule for J, then there exists a Δ-schedule for J with the same makespan. The the schedule up to R_i does not depend on R_j, $j > i$.

Proof. We can construct the schedule inductively. For any time i, let C be the configuration of S up to job R_i, and denote the configuration of the Δ-schedule S' we are constructing by C'. Our construction will ensure that, for any h, $\lambda(\{t : C(t) = h\}) = \lambda(\{t : C'(t) = h\}$.

Suppose we have constructed S' for $R_1, \ldots, R_i$; we show that we can extend it to a schedule for $R_1, \ldots R_{i+1}$ As before, let C and C' be the configurations after the scheduling of R_i. There are only finitely many h_n such that $\{t : C(t) = h_n\}$ has positive measure.

Then, for each n, consider the portion p_n of R_{i+1} that is being distributed over processors that of height h_n in C. These portions add up to the full width of R_{i+1}. We can cut R_{i+1} into finitely many pieces $P_1, \ldots, P_n$, of length $p_1, \ldots, p_n$. Then each P_i is cut into finitely many pieces to be scheduled at processors in C' that have load h_n. This completes the induction step. ■

2.3.2 Proposition
If J is a continuous job function, then for any $\varepsilon > 0$ there exists a job step function J' such that $J(t) \leq J'(t)$, with the following property: For any schedule (of Δ- or of Γ-type) for J of makespan M, the makespan M' of the corresponding schedule for J' satisfies

$$M \leq M' \leq (1+\varepsilon)M.$$

For any t, $J'|[0,t)$ depends only on $J|[0,t)$.

Proof. We may assume that J is uniformly continuous; otherwise we will simply construct J' by glueing together the job sequences obtained from the uniformly continuous pieces of J.

Let $h := \inf J(t) > 0$. There exists a $\delta > 0$ such that, if $t_1 - t_2 < \delta$, then $|J(t_1) - J(t_2)| < \frac{\varepsilon}{2}h$. Let J' be the step function all of whose steps are of width δ and whose value on the interval $[n\delta, (n+1)\delta)$ is $J(n\delta) + \frac{\varepsilon}{2}h$. Then by definition,

$$0 \leq J'(t) - J(t) \leq \varepsilon h \leq \varepsilon J(t).$$

Thus

$$J(t) \leq J'(t) \leq (1+\varepsilon)J(t).$$

Therefore J' has the required properties. ∎

2.3.3 Corollary
For any job sequence J,

$$OPT^{\infty}(J) := \inf\{\textit{makespan of } C : C \textit{ is the configuration of a schedule for } J\}$$

is independent of whether Δ- or Γ-schedules are used.

Proof. By Proposition 2.3.1, the claim is true for block sequences and step functions. By Proposition 2.3.2, reducing a continuous job function to a step function multiplies OPT by at most $1+\varepsilon$. ∎

Let us now investigate the equivalence of the different online versions of the limit problems. Competitiveness of an "algorithm" for the continuous problems is defined as usual. For abbreviation, we introduce the following notation:

2.3.4 Definition
We write $(x|\xi) \prec (x'|\xi')$ if for any c-competitive algorithm for the online $(x'|\xi')$-scheduling problem, there is, for every ε, a $c+\varepsilon$-competitive algorithm for the online $(x|\xi)$-scheduling problem.

We write $(x|\xi) \asymp (x'|\xi')$ if both $(x|\xi) \prec (x'|\xi')$ and $(x'|\xi') \prec (x|\xi)$.

Clearly if we are given an online algorithm that uses C- or S-inputs, this algorithm immediately restricts to one with S- or B-inputs, respectively, and which has the same competitive ratio. In the same way, an online algorithm that produces a Δ-schedule online is in particular one that produces an Γ-schedule. (Recall that the offline optimum does *not* depend on the scheduling type!) In other words,

$$
\begin{tikzcd}
(B|\Delta) \arrow[r, leftrightarrow] & (B|\Gamma) \\
(S|\Delta) \arrow[u] \arrow[r] \arrow[d, leftrightarrow] & (S|\Gamma) \arrow[u] \arrow[d, leftrightarrow] \\
(C|\Delta) \arrow[r] & (C|\Gamma)
\end{tikzcd}
$$

Figure 2.2: Relationships between the continuous online scheduling problems

2.3.5 Fact
For any $\xi \in \{\Delta, \Gamma\}$,

$$(B|\xi) \prec (S|\xi) \prec (C|\xi).$$

For any $x \in \{B, S, C\}$,

$$(x, \Gamma) \prec (x, \Delta).$$

Furthermore, as we have seen in the previous section, continuous job functions and step functions are not very different.

2.3.6 Lemma
For any $\xi \in \{\Delta, \Gamma\}$,

$$(C|\xi) \asymp (S|\xi).$$

PROOF. This follows directly from Proposition 2.3.2: In order to obtain a $c(1+\varepsilon)$-competitive algorithm for the (C, ξ)-problem, run the c-competitive algorithm on the corresponding sequence J'. Then the makespan of the resulting schedule will be

$$M \leq M' \leq c\mathrm{OPT}(J') \leq c(1+\varepsilon)\mathrm{OPT}(J).$$

This proves $c(1+\varepsilon)$-competitiveness. ∎

2.3.7 Lemma
$(B|\Delta) \asymp (B|\Gamma)$.

PROOF. This is an immediate consequence of Proposition 2.3.1. ∎

These relationships are illustrated in Figure 2.2.
The remaining problem is to prove the following conjecture.

2.3.8 Conjecture
$(S|\Delta) \prec (B|\Delta)$.

If this is the case, then all online versions are equivalent.

2.4 Lower Bounds and Generalized Competitive Ratios for the Limit Problem

In this short section, we will make definitions of the limit problem analogous to those we made in Chapter 1. As for the m-processor problem, we believe it is useful to study *makespan lower bounds*; i.e., lower bounds on the optimum makespan for the limit problem. Recall that OPT does not depend on which of the limit problems we consider, so neither does the notion of a makespan lower bound. Again, we think that it will be useful to consider also the notion of competitiveness with respect to such a lower bound.

The notion of layered forcing sequences also carries over to the limit problem. A layered c-forcing sequence is an input sequence consisting of a consecutive number of "layers" of width 1. Any online schedule that schedules any two jobs a layer on the same machine immediately has competitive ratio at least c. Again, such forcing sequences can be defined with respect to some makespan lower bound. Note that, by the very nature of forcing sequences, a forcing sequence for any version of the continuous online scheduling problem is one for all of these versions.

Let us carry the elementary lower bounds for the classical problem over to the limit problem. First recall that if J is a job step function all of whose step widths are multiples of $\frac{1}{m}$, then there is an associated job sequence for the m-processor scheduling problem. Let us refer to this sequence as J_m.

Note that, for any k, the makespan bound $B_k(J_m)$ depends only on J but not on m. We set $B_k(J) := B_k(J_m)$. Now consider the sharpened bound $B'_k(J_m)$. This bound *does* depend on m, however, it is nonincreasing in m and is therefore constant for m large enough. We set $B'_k(J) := \inf_m B'_k(J_m)$.

2.4.1 Proposition
B and B' are makespan lower bounds for the limit problem.

The proof is analogous to the discrete case and will be omitted. □

We also write $B_k^m(J), {B'}_k^m(J)$ etc. for the makespan lower bounds we defined in the beginning. As noted before, $B_k^m(J)$ is actually independent of m, and so $B_k^\infty(J) := B_k^m(J)$ is a lower bound on $\mathrm{OPT}^\infty(J)$.

Similarly, ${B'}_k^m(J)$ is nonincreasing in m, and thus we can define

$$ {B'}_k^\infty(J) := \lim_{m\to\infty} {B'}_k^m(J). $$

Chapter 3

Comparing the Limit Problem to the m-processor problem

3.1 Overview

Even while the continuous scheduling problem appears to be a rather natural limit problem of the m-machine scheduling problems, it is difficult to prove exact theorems about the approximation. The reason is mainly the fact that, as discussed below, splitting jobs in a schedule, even if it only contains steps of a fixed width, may be advantageous. Thus, even for an entry to the original m-processor problem, the competitive ratio of a given schedule may drop as m increases.

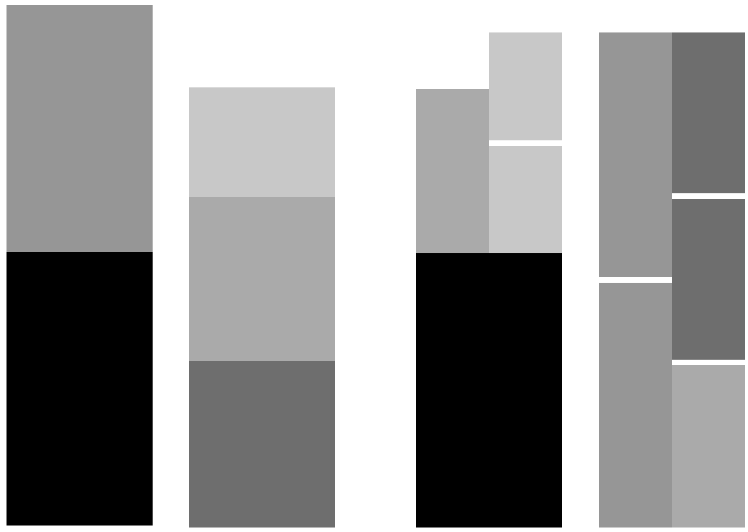
On the other hand, even if we can schedule a job sequence well in the continuous model, there will usually be much splitting involved. It seems very hard to get any bounds on the advantage that this splitting provides, without any assumptions on the special form of the algorithm.

Finally, a sequence of c-competitive algorithms for increasing m is certainly far from allowing us to find an algorithm for the limit problem.

The methods currently applied to obtain either algorithms or lower bounds for the scheduling problems are, however, *not* affected by these problems. In this chapter, we first discuss the influence of the ability to split jobs on the optimum makespan. We see that, if J is a job step function, then $\mathrm{OPT}(J_m)$ will in general depend on m. $\mathrm{OPT}(J_m)$ will generally not be monotone in m. However, we also show that the fact that an algorithm for the limit problem might split the job into pieces of irrational width does not reduce the optimum.

Furthermore, we will introduce a notion of well-behaved families of makespan lower bounds that will allow the passing of information between the m-processor scheduling problem and the limit problem. We illustrate the use of this notion by proving the equivalence of existence of layered forcing sequences for large m and for the limit problem.

For any of the elementary lower bounds B discussed previously, and any job step function, let us agree to define $B^m(J) := B(J_m)$ and $B^\infty := B(J)$.



(a) An optimal schedule for J_2

(b) An optimal schedule for J_4

Figure 3.1: A job sequence J for which $\mathrm{OPT}^2(J) > \mathrm{OPT}^4(J)$

3.2 Splitting of Job Sequences

Before studying the limit problem, one could first ask what can be said about the problems for increasing m. Unfortunately, not much is known; it is even an open question whether $\sigma(m)$ is increasing in m. We will now discuss some of the problems that appear when one studies the splitting of jobs.

3.2.1 Proposition (Decrease of $\mathbf{OPT}_m$ Possible)
There exists a job sequence J for two processors such that $OPT^2(J) > OPT^4(J)$.

Proof. Consider the sequence J with $J_2 = (10, 9, 6, 6, 4)$. One easily checks that $\mathrm{OPT}_2(J) = 19$. Indeed, a schedule of J with makespan 19 is obtained by placing the two largest jobs on one processor, and the rest of the jobs on the other (Figure 3.1(a)). Conversely, suppose that the jobs of length 10 and 9 are scheduled on different machines. To achieve a makespan of at most 19, there may be at most one of the three smaller jobs on the same machine as that which has received the job of length 10. So the other processor has a load of at least $9 + 6 + 4 = 19$.

Now consider the same sequence for 4 processors, i.e., the one in which each job has been split into two jobs of the same length: $J_4 = (10, 10, 9, 9, 6, 6, 6, 6, 4, 4)$. A schedule with makespan 18 is given in Figure 3.1(b). ■

In fact, a careful consideration of this job sequence J even shows the following result.

3.2.2 Proposition
For the job seqence of the previous proposition, $OPT^{(2k-1)2}(J) = 19$ *and* $OPT^{4k}(J) = 18$ *for all* $k \geq 1$.

PROOF. We have $17.5 = A(J) \leq \mathrm{OPT}^{4k}(J) \leq \mathrm{OPT}^{4}(J) = 18$; since all item sizes are integers, $\mathrm{OPT}^{4k}(J) = 18$.

It remains to show that $\mathrm{OPT}^{(2k-1)2}(J) \neq 18$ for all k. So let us suppose that there existed a schedule for J on $(2k-1)2$ processors that has makespan 18. Then clearly all jobs of length 10 must be scheduled on different processors. So consider the other jobs on these $2k-1$ processors. Clearly, all of these must have length 4 or 6 (or the schedule would have makespan ≥ 19). Also, their total area must be at least $A(J) - 5 - 9 = 3.5$. In other words, they would give a schedule on $2k-1$ processors which has makespan ≤ 8, contains only jobs of length 4 and 6 and on which no more than $\frac{2k-1}{2}$ processors have a height of 6 or less. Thus at least $\lceil \frac{2k-1}{2} \rceil$ processors must have two jobs of length 4 on them, which is not possible. ∎

Note, however, that OPT^{m} only takes values in a discrete set, and therefore, for every J, there exists m_0 such that $\mathrm{OPT}^{m_0}(J)$ is minimal. The only chance that an algorithm for the limit problem might have to improve this even further is by splitting jobs into pieces of non-rational length. It is not surprising that this does not happen.

3.2.3 Theorem
For every continuous job sequence J*, there exists* m_0 *such that*

$$OPT^{m_0}(J) = OPT^{\infty}(J).$$

PROOF. We need to prove that, for any continuous schedule of J, we can split the jobs into pieces of rational width while achieving the same makespan. Consider any schedule of J, and the corresponding configuration. By a *processor configuration* we mean a collection $(l_1, \ldots, l_k)$ of job lengths that are present in J. There are only finitely many such configurations $C_1, \ldots, C_k$ that are used in the schedule of J. For any C_k and any job length l, let $n_{C_k,l}$ denote the number of times that l appears in C_k. Also denote by θ_l the total width of jobs of length l in J; note that θ_l is by definition rational. For each processor configuration C, introduce a variable x_i and consider the linear system of equations

$$\sum_{l \in C} n_{C,l} * x_C = \theta_l.$$

All coefficients of these equations are rational. By assumption there exists a positive real solution vector, so there must also be a positive rational solution vector. ∎

3.3 Families of Lower Bounds

We now discuss lower bounds that respect the transition to the limit problem as m increases.

3.3.1 Definition (Proper Families of Lower Bounds)
For every m, $2 \leq m \leq \infty$, let B^m be a makespan lower bound for the m-processor scheduling problem, where $m = \infty$ means the continuous scheduling problem. The family (B^m) is called proper *if $B^\infty \leq B^{km} \leq B^m$ for all k, m and J.*

Remark.

1. OPT and all B_k (and B'_k) are proper families of lower bounds.
2. From *any* family B of lower bounds, we can obtain a proper family B' by setting

$$B'^m(J) := \sup_{k \in \mathbb{N}} B^{mk}(J), B^\infty(J)$$

   for $m \in \mathbb{N}$. Therefore, when we talk of a *family of lower bounds* in the following, we always assume it is proper.

3.3.2 Definition (Stationary and Continuous Bounds)
Let (B^m) be a (proper) family of lower bounds. The family B^m is called stationary *if, for any job sequence J, $B^m(J)$ does not depend on m. It is called* weakly stationary *if, for every J, there exists some m_0 such that*

$$B^{km_0}(J) = B^\infty(J) \tag{*}$$

for all $k \in \mathbb{N}$. B^m is called continuous *if, for any job sequence J,*

$$B^m(J) \to B^\infty(J)$$

as $m \to \infty$. It is called weakly continuous *if*

$$\inf_k B^{mk}(J) = B^{\infty(J)}.$$

3.3.3 Remark
Clearly, a stationary bound is also weakly stationary, and a continuous bound is weakly continuous (because it is assumed to be proper). Also a (weakly) stationary bound is (weakly) continuous.

Furthermore, a weakly continuous bound which takes on only a discrete set of values is weakly stationary. If a weakly continuous bound is nonincreasing in m, then it is in fact continuous.

3.3.4 Proposition

- *OPT is weakly stationary.*
- *The bound B_k is stationary.*
- *The bound B'_k is continuous and weakly stationary.*

PROOF. B_k is stationary by definition; B'_k is weakly continuous by definition. Since it takes on only a discrete set of values (namely sums of job sizes), it is also weakly stationary. Furthermore, B'_k is nonincreasing in m; thus it is also continuous. OPT is weakly stationary by Theorem 3.2.3. ■

3.3.5 Definition (Discrete Lower Bounds)
Let B be a makespan lower bound for the continuous scheduling problem. B is called weakly discrete *if, for every job step function $J : [0, s) \to (0, \infty)$, the (nondecreasing) function*

$$(0, s] \to (0, \infty), t \mapsto B\left(J|_{[0,t)}\right) \tag{*}$$

has finitely many points of discontinuity, all of which are rational. B is called discrete *if it is weakly discrete, and the function (*) is piecewise constant. We call a family (B^m) of lower bounds discrete or weakly discrete if the bound B^∞ is.*

3.3.6 Proposition
OPT is discrete. The bounds B_k and B'_k are weakly discrete.

PROOF. For the bounds B_k and B'_k, the claim is easily verified. Let J be any job step function. Then the function

$$t \mapsto \mathrm{OPT}\left(J|_{[0,t)}\right)$$

has only finitely many points of discontinuity, because it is nondecreasing and takes on only finitely many values. By the same argument as in the proof of 3.2.3, OPT only changes at rational points. ■

3.4 Transferring Lower Bounds

Recall that a *layered forcing sequence* consists of several layers, so that — in order to avoid a competitive ratio of c — all jobs of one layer have to be scheduled on different machines.

3.4.1 Theorem
Let B be any proper family of lower bounds, and let J be a layered forcing sequence for the m-processor problem, forcing a B-competitive ratio of c. Then J is also a layered forcing sequence for the km-processor problem, as well as for the continuous problem, forcing the same ratio.

PROOF. Recall first that $B^k m(J)$ does not increase as k increases, although it might decrease. Consider the first time that a job (or part of a job) is scheduled on a machine that already contains a job from the same layer. Then, by the definition of layered forcing sequences, the makespan at this time is larger than the optimum at the end of the job. The bound for the optimum at the end of the job is at least as large as it is up to the point where the job is split, which completes the proof. ■

3.4.2 Theorem

Let B be a weakly continuous and weakly discrete family of lower bounds.

Suppose that the step function J is a (not necessarily layered) forcing sequence, forcing a B-competitive ratio c for the limit problem. Then for every $\varepsilon > 0$, there exists m_0 such that, if m is a multiple of m_0, then J forces a B-competitive ratio of $c - \varepsilon$ for the m-processor problem.

If B is weakly stationary and discrete (in particular if $B = OPT$), then $c - \varepsilon$ can be replaced by c.

Proof. Define

$$f(t) := B\left(J|_{[0,t)}\right).$$

Because B is weakly stationary, we can find m_1 such that, for all k,

$$\inf\left\{f(t) : t \in \left(\frac{k-1}{m_1}, \frac{k}{m_1}\right]\right\} \geq f\left(\frac{k}{m_1}\right) - \varepsilon.$$

Because B is weakly continuous, we can then find a multiple m_0 of m_1 such that for all k,

$$B^{m_0}\left(J|_{\left[0,\frac{k}{m_1}\right)}\right) \leq B^{\infty}\left(J|_{\left[0,\frac{k}{m_1}\right)}\right) + \varepsilon.$$

Now consider any online algorithm for the m_0-scheduling problem. Any schedule for J in the m_0-processor problem is in particular a schedule for the continuous scheduling problem. Because J is a forcing sequence for the continuous problem, there exists a t such that the makespan M of the schedule at time t satisfies

$$M \geq c \cdot B^{\infty}\left(J|_{[0,t)}\right).$$

Now let k such that $(k-1) < t \cdot m_1 \leq k$. Then

$$B^{\infty}\left(J|_{[0,t)}\right) = f(t) \geq f\left(\frac{k}{m_1}\right) - \varepsilon \geq B^{m_0}\left(J|_{\left[0,\frac{k}{m_1}\right)}\right) - 2\varepsilon.$$

Therefore, after scheduling all of $J|_{[0,k/m_1)}$ — or, in other words, after scheduling k jobs — the makespan is

$$M \geq c \cdot \left(B^{m_0}\left(J|_{[0,\frac{k}{m_1})}\right) - 2\varepsilon\right).$$

This proves the theorem. ■

3.4.3 Theorem

If the continuous job function J is a layered forcing sequence for competitive ratio c, then there is a step function that is a layered forcing sequence for competitive ratio $c - \varepsilon$ (for any ε).

PROOF. This follows from the discussion in Section 2.3, where we saw how to discretize a continuous job sequence without increasing its optimum too much. ■

REMARK. The only reason that we formulated this theorem only for the optimum is that we did not discuss how families of lower bounds should be extended to continuous job functions.

Finally, let us remark that Theorem 3.4.2 generalizes — with the same proof — to a theorem about adaptive adversaries for the $(B|\Gamma)$ problem.

3.4.4 Theorem

Let B be a weakly continuous and weakly discrete family of lower bounds.

Suppose that A is a finite (adaptive) adversary forcing a B-competitive ratio c for the $(B|\Gamma)$ online scheduling problem. Then for every $\varepsilon > 0$, there exists m so that A also forces a B-competitive ratio of $c - \varepsilon$ for the m-processor problem.

Chapter 4

Conclusion

We presented an approach to the scheduling problem which focuses on the study of B-competitive ratios for elementary makespan lower bounds B. As we noted in the introduction, this approach has recently been independently taken by Susanne Albers with spectacular results. We expect that there will be further improvements in the future.

We also presented a family of limit problems for the m-processor scheduling problem. We believe that, like the offline versions, the online versions of all these problems are equivalent, and proving the remaining reduction of Conjecture 2.3.8 is a priority in ongoing research.

In our discussion of the relation between the discrete and the continuous scheduling problems, we have focused only on carrying lower bounds on the competitive ratio between the two problems. However, there are also notions that allow the translation of certain families of algorithms for the m-processor problems to the limit problem. In particular, this is true for the current best algorithms. In the other direction, conditions can be formulated that allow to transform an online algorithm for the limit problem to one for the original problems.

As announced in the introduction, let us present, in closing, a deterministic algorithm for the limit problem that might have a chance at being an improvement over known algorithms for large m. This algorithm has not been analyzed, and there are several parameters that can be addjusted.

Let c be the target competitive ratio. The algorithm will attempt to keep the schedule close to some target curve $c : [0, 1]$ (with $\int_0^1 c(t)dt = 1.$) For lack of a better choice, let us take $c(t) := \exp(1-t)/(e-1)$; the optimal distribution we may try to achieve in the randomized case. For any x, let $A(x) := \int_0^x c(t)dt$.

At any point of time, consider the current schedule C. Let us assume, for simplicity, that the processors are ordered with nonincreasing loads, i.e., so that $C : [0, 1]$ is nonincreasing. Let L be the total load scheduled so far, and $L(x) := \int_0^x c(t)dt$ the total load over the x largest loaded processors. Then the algorithm schedules a job of length ℓ by the following rule:

Schedule the job at the processor corresponding to the smallest x such that

- $C(X) + \ell \leq c * L$ and
- $\frac{L(x)}{L} \leq A(x)$.

If no such x exists, schedule ℓ on a least loaded processor.

An algorithm of this kind generalizes the idea of the algorithm from [A1], but removes the very discrete nature of that algorithm, in which only $x = 0$ and $x = \frac{1}{2}$ are allowed as choices for scheduling. The analysis of the algorithm is the subject of ongoing research.

Bibliography

[A1] Susanne Albers, *Better Bounds for On-Line Scheduling*, STOC'1997 Proceedings, 130-139

[A2] Susanne Albers, *On Randomized Online Scheduling*, Proc. 34th ACM Symposium on Theory of Computing, 2002

[BFK] Y. Bartal, H. Fiat, H. Karloff and R. Vohra, *New algorithms for an ancient scheduling problem*, Journal of Computer and System Sciences, 51 (1995), 359-366

[CGJ] E.G. Coffman, Jr., M. R. Garey, and D. S. Johnson, *Approximation Algorithms for Bin Packing: A Survey*, in: Approximation Algorithms for NP-hard problems, edited by D.S. Hochbaum (1997), 46-93

[FW] R. Fleischer and M. Wahl, *Online scheduling revisited*, Proc. 8th Annual European Symposium on Algorithm, Springer LNCS (2000)

[G1] R.L. Graham, *Bounds for Certain Multiprocessor Anomalies*, Bell System Technical Journal, 45 (1966), 1563-1581

[G2] R.L. Graham, *Bounds on Multiprocessor Timing Anomalies*, SIAM J. Appl. Math., 17 (1969), 416-429

[GJ] Michael R. Garey, David S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness*; Freeman and Company, 1979

[HS] Dorit Hochbaum and David Shmoys, *A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach*, SIAM J. Computing 17 (1988, no.3), 539-551

[K] R. M. Karp, *Reducibility among combinatorial problems*, in *Complexity of Computer Computations*, Miller/Thatcher (eds), Plenum Press, 1972, 85-103

[KPT] D.R. Karger, S.J. Phillips and E. Torng, *A Better Algorithm for an Ancient Scheduling Problem*, Journal of Algorithms, 20 (1996), 400-430

[R] John F. Rudin, *Improved Bounds for the Online Scheduling Problem*, PhD Thesis, 2001

[RC] John F. Rudin, R. Chandrasekaran, *Improved Bounds for the Online Scheduling Problem*, Manuscript, 2002

[S1] Jiří Sgall, *A Lower Bound for Randomized On-Line Multiprocessor Scheduling*, Information Processing Letter 63 (1997), 51-55

[S2] Jiří Sgall, *On-Line Scheduling: A Survey*

[T1] Tomáš Tichý, *Pravděpodobnostní on-line algoritmy pro rozvrhování*, Diploma Thesis, Univerzita Karlova v Praze, 2001

[T2] Tomáš Tichý, *Multiprocessor Probabilistic Online Scheduling*, Manuscript, 2002