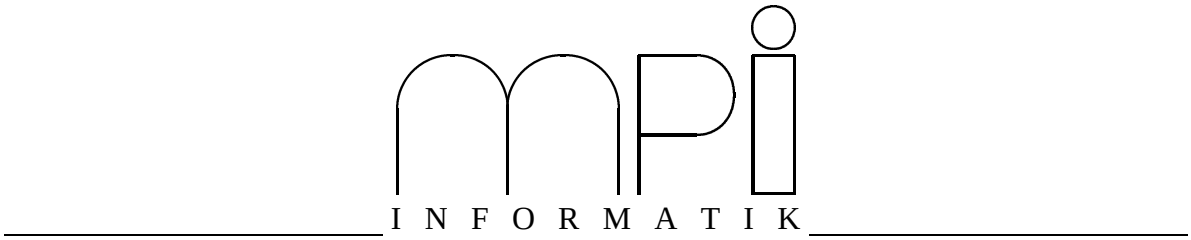


Online Scheduling with Bounded Migration

Peter Sanders Naveen Sivadasan
Martin Skutella

MPI–I–2004–1–004 April 2004

FORSCHUNGSBERICHT RESEARCH REPORT

MAX-PLANCK-INSTITUT
FÜR
INFORMATIK

Stuhlsatzenhausweg 85 66123 Saarbrücken Germany

Authors' Addresses

Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
Email: {sanders,ns,skutella}@mpi-sb.mpg.de

Publication Notes

An extended abstract of this report was accepted for publication in the proceedings of the 31st International Colloquium on Automata, Languages, and Programming (ICALP), 2004.

Acknowledgements

This work was partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT) and by the EU Thematic Network APPOL II, Approximation and Online Algorithms, IST-2001-30012.

Abstract

Consider the classical online scheduling problem where jobs that arrive one by one are assigned to identical parallel machines with the objective of minimizing the makespan. We generalize this problem by allowing the current assignment to be changed whenever a new job arrives, subject to the constraint that the total size of moved jobs is bounded by β times the size of the arriving job.

Our main result is a linear time 'online approximation scheme', that is, a family of online algorithms with competitive ratio $1 + \epsilon$ and constant migration factor $\beta(\epsilon)$, for any fixed $\epsilon > 0$. This result is of particular importance if considered in the context of sensitivity analysis: While a newly arriving job may force a complete change of the entire structure of an optimal schedule, only very limited 'local' changes suffice to preserve near-optimal solutions. We believe that this concept will find wide application in its own right.

We also present simple deterministic online algorithms with migration factors $\beta = 2$ and $\beta = 4/3$, respectively. Their competitive ratio $3/2$ beats the lower bound on the performance of any online algorithm in the classical setting without migration. We also present improved algorithms and similar results for closely related problems. In particular, there is a short discussion of corresponding results for the objective to maximize the minimum load of a machine. The latter problem has an application for configuring storage servers that was the original motivation for this work.

1 Introduction

A classical scheduling problem. One of the most fundamental scheduling problems asks for an assignment of jobs to m identical parallel machines so as to minimize the makespan. (The makespan is the completion time of the last job that finishes in the schedule; it also equals the maximum machine load.) In the standard classification scheme of Graham, Lawler, Lenstra, & Rinnooy Kan [15], this scheduling problem is denoted by $\mathrm{P}\,||\,C_{\max}$ and it is well known to be strongly NP-hard [12].

The *offline* variant of this problem assumes that all jobs are known in advance whereas in the *online* variant the jobs are incrementally revealed by an adversary and the online algorithm can only choose the machine for the new job without being allowed to move other jobs. Note that dropping this radical constraint on the online algorithm yields the offline situation.

A new online scheduling paradigm. We study a natural generalization of both offline and online problems. Jobs arrive incrementally but, upon arrival of a new job j, we are allowed to migrate *some* previous jobs to other machines. The total size of the migrated jobs however must be bounded by βp_j where p_j is the size of the new job. For *migration factor* $\beta = 0$ we get the online setting and for $\beta = \infty$ we get the offline setting.

Approximation algorithms. For an offline optimization problem, an *approximation algorithm* efficiently (in polynomial time) constructs schedules whose values are within a constant factor $\alpha \geq 1$ of the optimum solution value. The number α is called *performance guarantee* or *performance ratio* of the approximation algorithm. A family of polynomial time approximation algorithms with performance guarantee $1 + \epsilon$ for all fixed $\epsilon > 0$ is called a *polynomial time approximation scheme* (PTAS).

Competitive analysis. In a similar way, *competitive analysis* evaluates solutions computed in the online setting. An online algorithm achieves *competitive ratio* $\alpha \geq 1$ if it always maintains solutions whose objective values are within a factor α of the offline optimum. Here, in contrast to offline approximation results, the achievable values α are not determined by limited computing power but by the apparent lack of information about parts of the input that will only be revealed in the future. As a consequence, for all interesting classical online problems it is rather easy to come up with lower bounds that create a gap between the best possible competitive ratio α and 1. In particular, it is usually impossible to construct a family of $(1 + \epsilon)$-competitive online algorithms for such problems.

2 Related Work

For the online machine scheduling problem, Graham's *list scheduling* algorithm keeps the makespan within a factor $2 - 1/m$ of the offline optimum [13]: Schedule a newly arriving job on the least loaded machine. It can also easily be seen that this bound is tight: adversarial sequence consists of $m(m-1)$ jobs of size $\frac{1}{m}$ followed by one job of size 1. The optimal makespan in this case is 1.

For the offline setting, Graham showed three years later that sorting the jobs in the order of non-increasing size before feeding them to the list scheduling algorithm yields an approximation algorithm with performance ratio $4/3 - 1/(3m)$ [14]. Later, exploiting the relationship

between the machine scheduling problem under consideration and the binpacking problem, algorithms with improved approximation ratios have been obtained in a series of works [9, 11, 19].

Finally, polynomial time approximation schemes for a constant number of machines and for an arbitrary number of machines are given in [14, 24] and by Hochbaum & Shmoys [17], respectively. The latter PTAS partitions jobs into large and small jobs. The sizes of large jobs are rounded such that an optimum schedule for the rounded jobs can be obtained via dynamic programming. The small jobs are then added greedily using Graham's list scheduling algorithm. This approach can be refined to an algorithm with linear running time (see, e.g., [16]): replace the dynamic program with an integer linear program on a fixed number of variables and constraints which can be solved in constant time [20].

In a series of papers, increasingly complicated online algorithms with better and better competitive ratios beating the Graham bound 2 have been developed [6, 18, 2]. The best result known to date is a 1.9201-competitive algorithm due to Fleischer and Wahl [10]. The best lower bound 1.88 on the competitive ratio of any deterministic online algorithm currently known is due to Rudin [23]. For randomized online algorithms there is a lower bound of $e/(e-1) \approx 1.58$ [8, 27]. For more results on online algorithms for scheduling we refer to the recent survey articles by Albers [3] and Sgall [28].

Strategies that reassign jobs were studied in the context of online load balancing, jobs arrive in and depart from a system of m machines online and the scheduler has to assign each incoming job to one of the machines. Deviating from the usual approach of comparing against the optimal *peak load* seen so far, Westbrook [29] introduced the notion of competitiveness against *current load*: An algorithm is α–competitive if after every round the makespan is within α factor of the optimal makespan for the current set of jobs. Each incoming job u has size p_u and reassignment cost r_u. For a job, the reassignment cost has to be paid for its initial assignment and then every time it is reassigned. Observe that the optimal strategy has to pay this cost once for each job for its initial assignment. Thus the optimal (re)assignment cost S is simply the sum of reassignment costs of all jobs scheduled till now. Westbrook showed a 6-competitive strategy for identical machines with reassignment cost $3S$ for proportional reassignments, i.e., r_u is proportional to p_u, and $2S$ for unit reassignments, i.e., $r_u = 1$ for all jobs. Later Andrews et al. [4] improved it to 3.5981 with the same reassignment factors. They also showed $3+\epsilon$ and $2+\epsilon$ competitive strategies respectively for the proportional and unit case, the reassignment factor depending only on ϵ. For arbitrary reassignment costs they achieve 3.5981 competitiveness with 6.8285 reassignment factor. They also present a 32–competitive strategy with constant reassignment factor for related machines. Job deletions is an aspect that we do not consider in our work, our focus is primarily on achieving competitive ratios close to 1. Our results can also be interpreted in this framework of online load balancing, with proportional reassignments and without job deletions. We show strategies with better competitive ratios, at the same time achieving reassignment factor strictly less than three. We also show $(1+\epsilon)$–competitive strategies, for any $\epsilon > 0$, with constant reassignment factor $f(\epsilon)$. Our results are also stronger in the sense that a strategy with reassignment factor β ensures that when a job u arrives, the total reassignment cost incurred (for scheduling it) is at most βr_u. This is different from the more relaxed constraint that after t rounds, the total reassignment cost incurred is at most $\beta \sum r_u$ (summing over all jobs seen till round t). Most of our strategies are *robust*, they convert *any* α–competitive schedule to an α–competitive schedule after assigning the newly arrived job, whereas in [29, 4] it is required that the schedule so far is carefully constructed in order to ensure the competitiveness after assigning/deleting a job in the next round.

3 Our Contribution

In Section 5 we describe a simple online algorithm which achieves approximation ratio $3/2$ using a moderate migration factor $\beta = 2$. Notice that already this result beats the lower bound 1.88 (1.58) on the competitive ratio of any classical (randomized) online algorithm without migration. Using a more sophisticated analysis, the migration factor can be decreased to $4/3$ while maintaining competitive ratio $3/2$. On the other hand we show that our approach does not allow for migration factor 1 and competitive ratio $3/2$. Furthermore, an improved competitive ratio $4/3$ can be achieved with migration factor 4. For two machines, we can achieve competitive ratio $7/6$ with a migration factor of one. This ratio is tight for migration factor one.

Our main result can be found in Section 6. We present a family of online algorithms with competitive ratio $1 + \epsilon$ and constant migration factor $\beta(\epsilon)$, for any fixed $\epsilon > 0$. On the negative side, no constant migration factor suffices to maintain competitive ratio one, i.e., optimality. We provide interpretations of these results in several different contexts:

Online algorithms. Online scheduling with bounded job migration is a relaxation of the classical online paradigm. Obviously, there is a tradeoff between the desire for high quality solutions and the requirement to compute them online, that is, to deal with a lack of information. Our result can be interpreted in terms of the corresponding tradeoff curve: Any desired quality can be guaranteed while relaxing the online paradigm only moderately by allowing for a constant migration factor.

Sensitivity analysis. Given an optimum solution to an instance of an optimization problem and a slightly modified instance, can the given solution be turned into an optimum solution for the modified instance without changing the solution too much? This is the impelling question in sensitivity analysis. As indicated above, for the scheduling problem under consideration one has to answer in the negative. Already one additional job can change the entire structure of an optimum schedule. However, our result implies that the answer is positive if we only require near-optimum solutions.

Approximation results. Our result yields a new PTAS for the scheduling problem under consideration. Due to its online background, this PTAS constructs the solution incrementally. That is, it reads the input little by little always maintaining a $(1+\epsilon)$-approximate solution. Indeed, it follows from the analysis of the algorithm that every update only takes constant time. In particular, the overall running time is linear and thus matches the previously best known approximation result.

We believe that each of these interpretations constitutes an interesting motivation for results like the one we present here in its own right and can therefore lead to interesting results for many other optimization problems.

The underlying details of the presented online approximation scheme have the same roots as the original PTAS by Hochbaum & Shmoys [17] and its refinements [16]. We distinguish between small and large jobs; a job is called large if its size is of the same order of magnitude as the optimum makespan. Since this optimum can change when a new job arrives, the classification of jobs must be updated dynamically. The size of every large job is rounded such that the problem of computing an optimum schedule for the subset of large jobs can be formulated as an integer linear program of constant size. A newly arriving job causes a small change in the right hand side of this program. This enables us to use results from sensitivity analysis of integer programs in order to prove that the schedule of large jobs needs to be changed only slightly.

Our PTAS is very simple, it uses only this structural result and does not use any algorithms from integer programming theory. For a detailed account of linear and integer programming theory we refer to the books [26, 21].

In Section 8 we discuss an application of bounded migration to configuring storage servers. This was the original motivation for our work. In this application, the objective is to maximize the minimum load. It is well-known [5] that any online deterministic algorithm for this *machine covering problem* has competitive ratio at least m (the number of machines). There is also a lower bound of $\Omega(\sqrt{m})$ for any randomized online algorithm. We develop a simple deterministic online strategy which is 2-competitive already for migration factor $\beta = 1$.

4 Preliminaries

Let the set of *machines* be denoted by $M = \{1, \ldots, m\}$. The set of *jobs* is $\{1, \ldots, n\}$ where job j arrives in round j. Let p_j denote the positive *processing time* or the *size* of job j. For a subset of jobs N, the *total processing time* of jobs in N is $p(N) := \sum_{j \in N} p_j$; let $p_{\max}(N) := \max_{j \in N} p_j$. For a schedule on the set of jobs N, let $S(i)$ denote the set of *jobs scheduled on machine* i. For a subset of machines $Y \subseteq M$, let $S(Y) := \bigcup_{i \in Y} S(i)$. For a subset of jobs N, let $\mathrm{opt}(N)$ denote the *optimal makespan.* If the subset of jobs N and a newly arrived job j are clear from the context, we sometimes also use the shorter notation $\mathrm{opt} := \mathrm{opt}(N)$ and $\mathrm{opt}' := \mathrm{opt}(N \cup \{j\})$. It is easy to observe that $\mathrm{lb}(N) := \max\{p(N)/m, p_{\max}(N)\}$ is a lower bound on $\mathrm{opt}(N)$ satisfying

$$\mathrm{lb}(N) \;\leq\; \mathrm{opt}(N) \;\leq\; 2\mathrm{lb}(N) \;. \tag{1}$$

The following well-known fact is used frequently in the subsequent sections.

Observation 1. *For a set of jobs N, consider an arbitrary schedule with makespan κ. Assigning a new job j to the least loaded machine yields a schedule with makespan at most* $\max\{\kappa, \mathrm{opt}(N \cup \{j\}) + (1 - 1/m)p_j\}$.

Proof. We need to show that if the makespan changes after scheduling job j, then the new makespan is at most $\mathrm{opt}(N \cup \{j\}) + p_j(1 - 1/m)$. Since job j is scheduled on the least loaded machine, the new makespan is at most $p(N)/m + p_j$. This combined with the following inequality yields the fact; $\mathrm{opt}(N \cup \{j\}) \geq p(N \cup \{j\})/m = p(N)/m + p_j/m$. □

5 Strategies with Small Migration Factor

We consider the problem of scheduling jobs arriving one after another on m parallel machines so as to minimize the makespan. We first show a very simple $3/2$-competitive algorithm with migration factor 2. The algorithm is as follows:

Procedure FILL_1:
Upon arrival of a new job j, choose one of the following two options minimizing the resulting makespan.

Option 1: Assign job j to the least loaded machine.

Option 2: Let i be the machine minimizing the maximum job size. Repeatedly remove jobs from this machine; stop before the total size of removed jobs exceeds $2p_j$. Assign job j to machine i. Assign the removed jobs successively to the least loaded machine.

Theorem 1. *Procedure* FILL_1 *is* $\left(\frac{3}{2}-\frac{1}{2m}\right)$*-competitive with migration factor* 2.

Proof. From the description of FILL_1, it is clear that the migration factor is at most 2. In order to prove competitiveness, we consider an arbitrary $\left(\frac{3}{2}-\frac{1}{2m}\right)$-approximate schedule for a set of jobs N and show that incorporating a new job j according to FILL_1 results in a new schedule which is still $\left(\frac{3}{2}-\frac{1}{2m}\right)$-approximate. In the following, a job is called *small* if its processing time is at most $\mathrm{opt}'/2$, otherwise it is called *large*. If the new job j is small, then the first option yields makespan at most $\left(\frac{3}{2}-\frac{1}{2m}\right)\mathrm{opt}'$ by Observation 1. Thus, we can assume from now on that j is large.

Since there can be at most m large jobs in $N \cup \{j\}$, all jobs on the machine chosen in the second option are small. Thus, after removing jobs from this machine as described above, the machine is either empty or the total size of removed jobs exceeds the size of the large job j. In both cases, assigning job j to this machine cannot increase its load above $\left(\frac{3}{2}-\frac{1}{2m}\right)\mathrm{opt}'$. Thus, using the same argument as above, assigning the removed small jobs successively to the least loaded machine yields again a $\left(\frac{3}{2}-\frac{1}{2m}\right)$-approximate schedule. □

Next we show that the migration factor can be decreased to $4/3$ without increasing the competitive ratio above $3/2$. This result is achieved by carefully modifying FILL_1.

Procedure FILL_2 :
Upon arrival of j, choose the one of the following $m+1$ options that minimizes the resulting makespan. (Break ties in favor of option 0.)

Option 0: Assign job j to the least loaded machine.

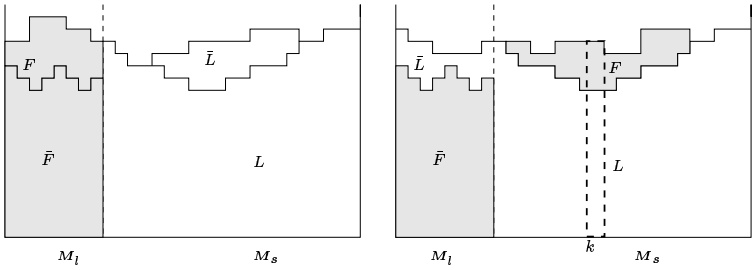
Option i [for $i \in \{1,\ldots,m\}$]: Ignoring the largest job on machine i, consider the remaining jobs in the order of non-increasing size and remove them from the machine; stop before the total size of removed jobs exceeds $\frac{4}{3}p_j$. Assign job j to machine i. Assign the removed jobs successively to the least loaded machine.

Theorem 2. *Procedure* FILL_2 *is* $\frac{3}{2}$*-competitive with migration factor* $\frac{4}{3}$.

Proof. The migration factor is clear from the description of FILL_2 . To show the competitive ratio, we consider an arbitrary $\frac{3}{2}$–approximate schedule of N, also denoted as *input schedule*, that additionally satisfies the following property; the total load on any machine excluding its largest job is at most opt. We show that incorporating the new job j results in a $\frac{3}{2}$–approximate schedule. As shown by the following fact, the resulting schedule also satisfies the above additional property.

Fact 1. *After scheduling job* j*, the total load on any machine excluding its largest job is at most* opt'.

Proof. Let M' be the subset of machines 'touched' by FILL_2 for scheduling job j. It suffices to show that the total load in such a machine excluding its 'last' job (the job that entered last) is at most opt'. Fix any machine in M' and consider its last job. If it is not j then it was assigned as part of the redistribution phase. Since redistribution is always performed on the current least loaded machine, the load in this machine excluding this last job is at most opt'. If the last job is j then, as option 0 is always preferred, the total load of this machine can only be smaller than the load in the least loaded machine of the input schedule together with the size of job j. Thus the fact follows. □



Optimal Schedule of jobs in N Input Schedule of jobs in N

Figure 1: Figure showing the differences between the optimal schedule and the input schedule of jobs in N. The sets $F, \bar{F}, L$ and $\bar{L}$ capture these differences. The 'common sets' in both schedule are $\bar{F}$ and L. On both schedules, exactly the machines in M_l contain one *large* job each and such jobs belong to $\bar{F}$.

We distinguish three cases depending on p_j. If $p_j \leq \text{opt}'/2$ then option 0 already yields a schedule of makespan at most $\frac{3}{2}\text{opt}'$ by Observation 1.

Case $\text{opt}'/2 < p_j \leq \frac{3}{4}\text{opt}'$: let

$$p_j = \text{opt}'/2 + \delta \quad \text{where} \quad 0 < \delta \leq \text{opt}'/4 \tag{2}$$

To handle this case, it suffices find a $k \in \{1, \ldots, m\}$ such that Option k yields a $\frac{3}{2}$–approximate schedule. We call it a 'good' option. We fix k as follows. Denote a job as *large* if its size is more than $\text{opt}' - \delta$ and *small* otherwise. With respect to the input schedule, partition the set of machines M as machines with only small jobs denoted as M_s, and the rest denoted as M_l. That is,

$$M_s = \{i \in M \mid p_{\max}(S(i)) \leq \text{opt}' - \delta\} \quad \text{and} \quad M_l = M - M_s \tag{3}$$

Observe that on any $\frac{3}{2}$–approximate schedule of N, a machine has at most one large job. Hence there are $|M_l|$ large jobs; one on each machine in M_l. Consequently, we fix an optimal schedule of N such that large jobs are scheduled on M_l. We now compare the input schedule with this optimal schedule in the following way. As shown in Figure 1, partition N as $F, \bar{F}, L, \bar{L}$. In the optimal schedule, the jobs scheduled on M_l and M_s are $F \cup \bar{F}$ and $L \cup \bar{L}$ respectively and, the respective sets in the input schedule are $\bar{F} \cup \bar{L}$ and $F \cup L$. It follows that in the input schedule there is a machine in M_s with total load at most opt with respect to L. We fix k as this machine. That is,

$$p(S(k) - F) = p(S(k) \cap L) \leq \text{opt} \tag{4}$$

It remains to show that option k is good.

By the choice of optimal schedule, every large job belongs to the 'common set' $\bar{F}$. Hence each job in set F has size at most δ; as it is scheduled along with a large job (from $\bar{F}$), in the optimal schedule. As a consequence, the set of jobs in machine k with each having size at most δ includes the subset of jobs induced by F. That is, if we partition the set $S(k)$ as K_δ and K_r, where

$$K_\delta = \{\text{All jobs in machine } k \text{ with size at most } \delta\}; \quad K_r = S(k) - K_\delta$$

then

$$p(K_r) \leq \text{opt} \quad (\text{as } F \subseteq K_\delta \text{ and } p(S(k) - F) \leq \text{opt} \text{ by (4)}) \tag{5}$$

This readily implies that there is at most one job with size greater than $\text{opt}'/2$ in machine k; as such jobs belongs to set K_r. Consequently, every job removed during option k has size at most $\text{opt}'/2$; as the largest job is untouched. This ensures by Observation 1 that, reassignment of the removed jobs still yields a $\frac{3}{2}$–approximate schedule, if the schedule before reassignment is $\frac{3}{2}$–approximate. Hence we are done with this case analysis if we show that the total load of machine k *before redistribution*, i.e., after removal of jobs and assignment of job j, is at most $\frac{3}{2}\text{opt}'$. That is, it is enough to ensure that for machine k,

$$p(\text{initial jobs}) - p(\text{removed jobs}) + p_j \leq \frac{3}{2}\text{opt}' \tag{6}$$

The interesting case is when the set of removed jobs excludes more jobs in addition to the largest job; the largest job has size at most $\text{opt}' - \delta$ (recall that machine k belongs to M_s). Even then, it is not a problem if an unremoved job belongs to K_δ as this readily implies that the total size of removed jobs is at least $\frac{4}{3}p_j - \delta \geq p_j$, where $\frac{4}{3}p_j$ is the total removal limit. The remaining situation is that every unremoved job (apart from the largest job) is from K_r. An unremoved job from K_r, by the greedy removal, immediately implies that there is a removed job from K_r. This is because, (a) the removal limit, i.e. $\frac{4}{3}p_j$, is at least $\text{opt}'/2$ and (b) all jobs except the largest job have size at most $\text{opt}'/2$ in machine k. Thus $p(\text{removed jobs}) \geq p(K_\delta) + \delta$, which in turn satisfies (6). The additional δ accounts for the removed job from K_r.

Case $p_j > \frac{3}{4}\text{opt}'$: let

$$p_j = \frac{3}{4}\text{opt}' + \delta \quad \text{where} \quad \delta \leq \text{opt}'/4 \tag{7}$$

Since there can be at most m jobs in $N \cup \{j\}$ each with size greater than $\text{opt}'/2$, there is a machine k in the input schedule where all jobs have size at most $\text{opt}'/2$. We show that option k is good. Observe that the reassignment of removed jobs yield $\frac{3}{2}\text{opt}'$ makespan schedule by Observation 1, if the schedule before has $\frac{3}{2}\text{opt}'$ makespan. Recall that in the input schedule, on any machine, the load excluding its largest job is at most opt. Hence all jobs, except the largest job, are removed and thus yielding a total load of at most $\frac{3}{2}\text{opt}'$ after assigning job j. This concludes the proof. □

Robustness

Most of our scheduling strategies for minimizing the makespan discussed in this chapter are robust in the following sense. The only invariant that we require in their analyses is that before the arrival of a new job the current schedule is α-approximate. Job j can then be incorporated yielding again an α-approximate schedule. In other words, we do not require that the current schedule is carefully constructed so far, to maintain the competitiveness in the next round. Only for Procedure FILL_2 , the schedule should additionally satisfy that, on any machine, the load excluding the largest job in it is at most the optimum makespan. We further remark that this is not an unreasonable requirement as even the simple list scheduling algorithm ensures this.

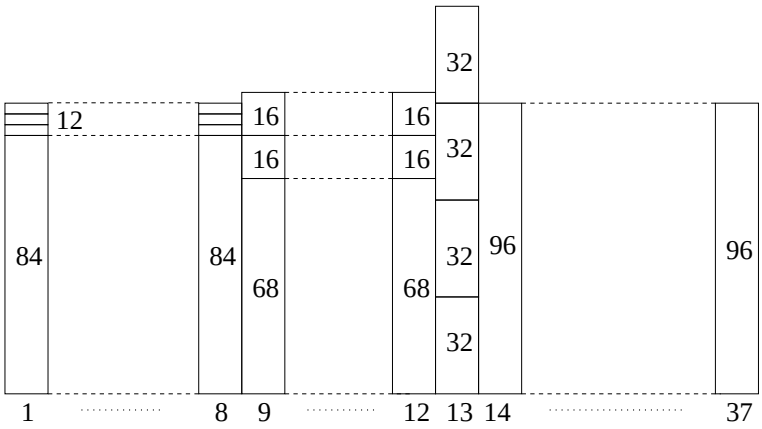


Figure 2: A $\frac{3}{2}$-approximate schedule ($m = 37$). If a new job of size 86.1 arrives, jobs of total size at least 96 have to be moved in order to construct a schedule that is still $\frac{3}{2}$-approximate.

5.1 Negative Results

Theorems 1 and 2 raise the question of which migration factor is really necessary to achieve competitive ratio $3/2$. We can prove that any robust strategy needs migration factor greater than 1 in order to maintain competitive ratio $3/2$.

Lemma 1. *There exists a $3/2$-approximate schedule such that, upon arrival of a particular job, migration factor 1.114 is needed to achieve $3/2$-competitiveness. Moreover, migration factor 1 only allows for competitive ratio 1.52 in this situation.*

Proof. The situation is depicted in Figure 2. There are 37 machines. Machines 1 to 8 are identically packed. Each of them has one job of size 84 and three jobs of size 4. Machines 9 to 12 are also identical. Each of them has one job of size 68 and two jobs of size 16. Machine 13 contains four jobs of size 32. Machines 14 to 37 contain one job of size 96 each. The size of the newly arriving job is 86.1. The optimal makespan is 100. To achieve a makespan of at most 150, it is necessary to migrate at least 3 jobs of size 32 from machine 13 to other machines. Hence, the migration factor is at least $96/86.1 > 1.114$. To show the lower bound on the competitiveness (the second part of the lemma), we set the size of the newly arriving job to 88 instead of 86.1. It is straightforward to check that the final optimal makespan is still 100 and the best possible makespan achievable for any strategy with migration factor $\beta \leq 1$ is 152. □

An additional feature of FILL_1 and FILL_2 is that they are *local* in the sense that they migrate jobs only from the machine where the newly arrived job is assigned to. There is a class of optimal schedules for which, upon arrival of a new job, it is not possible to achieve a better competitive ratio than $3/2$ using only local migration. This holds even if an arbitrary migration factor is allowed. The following optimal schedule on m machines, upon the arrival of a new job, enforces a competitive ratio of at least $3/(2 + \frac{2}{m})$ for any amount of migration. This bound converges to $3/2$ for large m. The example looks as follows: Machines 1 and 2 each contain one job of size $1/2$ and $m/2$ jobs of size $1/m$. All other machines contain a single job of size 1. The newly arriving job has size 1. The optimum makespan is $1 + 1/m$ and the makespan achievable by any local strategy is $3/2$ (by scheduling the new job on say, machine 1 and migrating all small jobs to other machines).

5.2 A $\frac{4}{3}$–Competitive Strategy

In this section, we show that an improved competitive ratio of $\frac{4}{3}$ can be achieved by a more sophisticated algorithm, FILL_3, with migration factor 4.

Procedure FILL_3:
Upon arrival of j, choose one of the following $m+1$ options that minimizes the resulting makespan.

Option 0: Assign job j to the least loaded machine.

Option i [for $i \in \{1, \ldots, m\}$]: Skip phase one if either $2p_j < p(S(i))$ or $p_j < p_{\max}(S(i))$.

Phase one: Let ℓ denote the largest job in machine i. Remove all jobs from machine i and schedule job j there. Except job ℓ, assign the removed jobs successively in the least loaded machine.

Phase two: We assign the unassigned job ℓ in this phase. If phase one was skipped then ℓ is simply job j. Consider $m+1$ sub-options and choose the one that minimizes the resulting makespan.

Sub-option 0: Assign job ℓ to the least loaded machine.

Sub-option k [for $k \in \{1, \ldots, m\}$]: Ignoring the largest job in machine k, consider the remaining jobs in order of non-increasing size and repeatedly remove them; stop before the total size of removed jobs exceeds $2p_j$. Assign job ℓ to machine k and assign the removed jobs successively to the least loaded machine.

Theorem 3. FILL_3 *is* $\frac{4}{3}$*–competitive with factor* 4 *migration.*

Proof. The migration factor is clear from the description of FILL_3. In both phases the migration factor is 2. To show the competitive ratio, we consider any arbitrary *input schedule* on the set of jobs N that is $\frac{4}{3}$–approximate. We show that FILL_3 yields a $\frac{4}{3}$–approximate schedule on $N \cup \{j\}$. We call a job b as either *small*, *medium* or *large* where

$$\text{small}: \ p_b \leq \text{opt}'/3 \qquad \text{medium}: \ \text{opt}'/3 \ < p_b \leq \frac{2}{3}\text{opt}' \qquad \text{large}: \ p_b > \frac{2}{3}\text{opt}'$$

If job j is small then option 0 yields a $\frac{4}{3}$–approximate schedule by Observation 1.

Case $p_j > \text{opt}'/3$: With respect to the input schedule, we partition the set of machines M as M_L and M_S, where

$$M_L \ = \ \{i \in M \mid \text{machine } i \text{ contains a large job}\} \ \text{ and } \ M_S = M - M_L$$

Since $p_j > \text{opt}'/3$, the number of large jobs in N is at most $m-1$; otherwise the optimal makespan of $N \cup \{j\}$ exceeds opt'.

Observation 2. *Consider the set of jobs* $A \cup \{b\}$ *with optimal makespan* opt' *on* m *machines. Let* $p_b > \text{opt}'/3$. *If there are* m' *large jobs* ($m' < m$) *in* A *then there are at most* $2(m-m')-1$ *medium jobs in* A.

Proof. Otherwise the optimal makespan of $A \cup \{b\}$ exceeds opt'. □

Clearly Observation 2 implies that there is a machine z with at most one medium job and no large jobs. Thus

$$p_{\max}(S(z)) \;\leq\; \frac{2}{3}\mathrm{opt}' \tag{8}$$

We show that option z is *good*, i.e., it yields a $\frac{4}{3}$–approximate schedule for $N \cup \{j\}$.

By Observation 2, all jobs in machine z, except possibly the largest, are small jobs. Hence reassigning them during phase one after scheduling job j is fine by Observation 1. Thus after phase one of option z, the 'intermediate schedule' is $\frac{4}{3}$–approximate.

We now show that the same holds after phase two of option z. Recall that the job to be assigned in this phase is denoted as ℓ. If phase one was skipped then ℓ is simply job j. Second phase is entered either

- skipping phase one because $2p_j < p(S(z)) \leq \frac{4}{3}\mathrm{opt}$ or $p_j < p_{\max}(S(z)) \overset{(8)}{\leq} \frac{2}{3}\mathrm{opt}'$. Job j is same as ℓ for phase two.
- or after phase one. Hence $p_\ell = p_{\max}(S(z)) \overset{(8)}{\leq} \frac{2}{3}\mathrm{opt}'$ and $p_\ell \leq p_j$.

In either case it follows that $p_\ell \leq p_j$ and job ℓ is either small or medium. If job ℓ is small then the sub-option 0 yields $\frac{4}{3}$–approximate schedule by Observation 1.

We complete the proof by handling the case that job ℓ is medium. As $p_\ell > \mathrm{opt}'/3$, by Observation 2, in the intermediate schedule after phase one there is a machine z' with at most one medium job and no large jobs. We show that sub-option z' yields a $\frac{4}{3}$–approximate schedule. Let the largest job in machine z' be ℓ'. Since ℓ' is untouched in sub-option z', all the removed jobs are small. Hence by Observation 1, the makespan of schedule after reassignment of removed jobs is $\frac{4}{3}\mathrm{opt}'$ if the schedule before has $\frac{4}{3}\mathrm{opt}'$ makespan. Clearly this is true if all jobs except ℓ' were removed, as both ℓ and ℓ' are non-large jobs. At least one unremoved job in addition to ℓ' also ensures this as the total size of removed jobs in this case is at least $2p_j - \mathrm{opt}'/3$, where $2p_j$ is the removal limit. Thus the total load in machine z' after assigning job ℓ is at most $\frac{4}{3}\mathrm{opt}'$ as $p_j \geq p_\ell$ and $p_\ell > \mathrm{opt}'/3$. □

Even better results are possible for two machines. In section 7, we discuss a specialized algorithm with competitive ratio $\frac{7}{6}$ and migration factor of one. We also show that this ratio is tight for any deterministic strategy with migration factor one.

6 An Online Approximation Scheme with Constant Migration

The results presented in the last section raise the question how far the competitive ratio for online algorithms with constant migration factor can be decreased. We first prove that optimality (i.e., competitive ratio 1) cannot be achieved. However, for any fixed $\epsilon > 0$ we can get down to competitive ratio $1 + \epsilon$.

Lemma 2. *Any online algorithm computing optimal solutions needs migration factor $\Omega(m)$.*

Proof. Consider a scheduling instance with m machines and $2m - 2$ jobs, two of size i/m for all $i = 1, \ldots, m-1$. Up to permutations of machines, any optimum schedule has the structure depicted in the left part of Figure 3. The optimum makespan is $(m-1)/m$. When a new job of size 1 arrives, the optimum makespan increases to 1. Again, the structure of an optimum

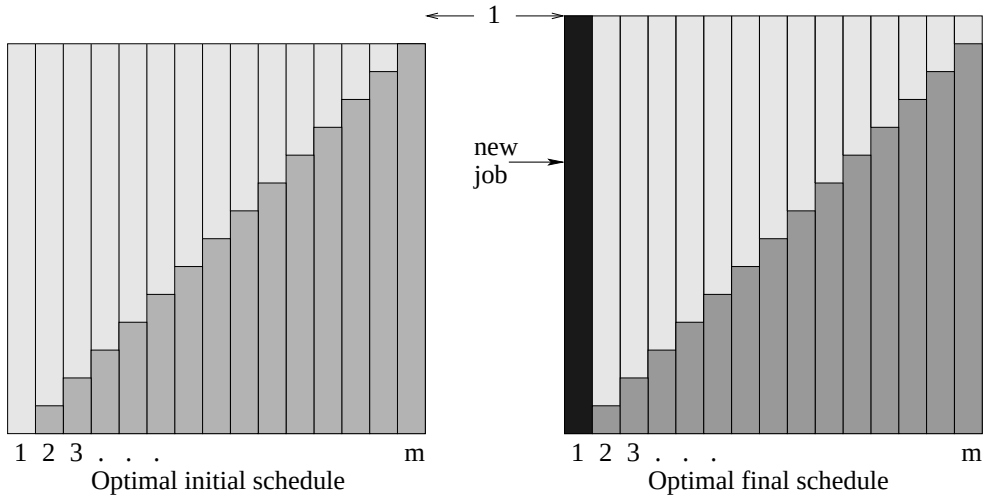


Figure 3: An instance where all machine configurations have to change to maintain optimality.

schedule for the enlarged instance is unique; see the right hand side of Figure 3. From each machine in $\{2, \ldots, m-1\}$, at least one job from the pair has to move. Hence the minimum total size of jobs that have to be migrated is at least

$$\frac{1}{m} \sum_{i=1}^{m-2} \min\{i,\ m-1-i\} \ \geq \ \frac{1}{m} \sum_{i=1}^{\lfloor (m-2)/2 \rfloor} i \ ,$$

which is $\Omega(m)$. □

In the following, $\epsilon > 0$ is a fixed constant. We assume without loss of generality that $\epsilon \leq 1$. The following observation belongs by now to the folklore in the field of scheduling; see, e.g., [1].

Observation 3. *Rounding up each job's processing time to the nearest integer power of* $1+\epsilon$ *increases the makespan of an arbitrary schedule at most by a factor* $1+\epsilon$*. In particular, in specifying a* $(1+O(\epsilon))$*-competitive algorithm we can assume that all processing times are integer powers of* $1+\epsilon$*.*

The current set of jobs is denoted by N. A job in N is called *large* if its processing time is at least $\epsilon \operatorname{lb}(N)$; otherwise, it is called *small*. The subset of large and small jobs is denoted by N_L and N_S, respectively. We partition N into classes N_i, $i \in \mathbb{Z}$, with

$$N_i \ := \ \{j \in N \mid p_j = (1+\epsilon)^i\} \ .$$

Let $I := \{i \in \mathbb{Z} \mid \epsilon \operatorname{lb}(N) \leq (1+\epsilon)^i \leq p_{\max}(N)\}$ such that $N_L = \bigcup_{i \in I} N_i$. Thus, the number of different job sizes for large jobs is bounded by $|I|$ and therefore constant:

$$|I| \ \leq \ 1 + \log_{1+\epsilon} \frac{p_{\max}(N)}{\epsilon \operatorname{lb}(N)} \ \leq \ 1 + \log_{1+\epsilon} \frac{1}{\epsilon} \ \leq \ \frac{2}{\epsilon} \log \left(\frac{1+\epsilon}{\epsilon} \right) \ . \tag{9}$$

Given an assignment of jobs N_L to machines, we say that a particular machine obeys *configuration* $k : I \to \mathbb{N}_0$ if, for all $i \in I$, exactly $k(i)$ jobs from N_i are assigned to this machine. The

set of configurations that can occur in any schedule for N_L is

$$\mathbf{K} \;:=\; \{k : I \to \mathbb{N}_0 \mid k(i) \leq |N_i| \text{ for all } i \in I\} \;.$$

Up to permutations of machines, an arbitrary schedule for N_L can be described by specifying, for each $k \in \mathbf{K}$, the number y_k of machines that obey configuration k. Conversely, a vector $y \in \mathbb{N}_0{}^{\mathbf{K}}$ specifies a feasible m-machine-schedule for N_L if and only if

$$\sum_{k \in \mathbf{K}} y_k \;=\; m \qquad \text{and} \tag{10}$$

$$\sum_{k \in \mathbf{K}} k(i)\, y_k \;=\; |N_i| \qquad \text{for all } i \in I. \tag{11}$$

We denote the set of vectors $y \in \mathbb{N}_0{}^{\mathbf{K}}$ satisfying (10) and (11) by $\mathbf{S}$. Thus, $\mathbf{S}$ represents the set of all schedules (up to permutations of machines and up to permutations of equal size jobs) for N_L. For a configuration $k \in \mathbf{K}$ let

$$\text{load}(k) \;:=\; \sum_{i \in I} (1+\epsilon)^i \, k(i)$$

denote the load of a machine obeying configuration k. The makespan of a schedule $y \in \mathbf{S}$ is equal to $\max\{\text{load}(k) \mid y_k > 0\}$. For $\mu \geq 0$, let

$$\mathbf{K}(\mu) \;:=\; \{k \in \mathbf{K} \mid \text{load}(k) \leq \mu\} \;.$$

The set of all schedules with makespan at most μ is denoted by

$$\mathbf{S}(\mu) \;:=\; \{y \in \mathbf{S} \mid y_k = 0 \text{ if } \text{load}(k) > \mu\} \;.$$

In the following, we usually interpret a schedule $y \in \mathbf{S}(\mu)$ as a vector in $\mathbb{N}_0{}^{\mathbf{K}(\mu)}$ by ignoring all zero-entries corresponding to configurations not contained in $\mathbf{K}(\mu)$.

The minimum makespan for N_L can be obtained by determining the minimum value μ with $\mathbf{S}(\mu) \neq \emptyset$. Checking whether $\mathbf{S}(\mu)$ is empty and, otherwise, finding a schedule $y \in \mathbf{S}(\mu)$ can be done by finding a feasible solution to an integer linear program. We can write

$$\mathbf{S}(\mu) \;=\; \{y \in \mathbb{N}_0{}^{\mathbf{K}(\mu)} \mid A(\mu)\, y = b\} \;,$$

where $A(\mu)$ is a matrix in $\mathbb{N}_0{}^{(1+|I|)\times|\mathbf{K}(\mu)|}$ and b is a vector in $\mathbb{N}_0{}^{1+|I|}$. The first row of the linear system $A(\mu)\, y = b$ corresponds to constraint (10); the remaining $|I|$ rows correspond to constraints (11).

Lemma 3. *Let N be a set of jobs and let j be a new job of size $p_j \geq \epsilon\,\text{lb}(N)$. Any schedule for N_L with makespan $\mu \leq (1+\epsilon)\text{opt}(N)$ can be turned into a schedule for $N_L \cup \{j\}$ by touching only a constant number of machines such that the makespan of the resulting schedule is at most $\max\{\mu, \text{opt}(N_L \cup \{j\})\}$.*

Proof. We distinguish two cases. If $p_j \geq 2\mu$, then it is easy to observe that $\text{opt}(N_L \cup \{j\}) = p_j$ and an optimal schedule for $N_L \cup \{j\}$ can be obtained by assigning job j to an arbitrary machine and moving all jobs that are currently on this machine to any other machine.

In the remainder of the proof we can assume that $p_j = (1+\epsilon)^{i'} < 2\mu$ and therefore $\mathrm{opt}(N_L \cup \{j\}) \leq 2\mu$. Let $y \in \mathbf{S}(\mu)$ denote the given schedule for N_L. Then, y satisfies

$$A(\mu)\, y \;=\; b \;, \qquad y \in {\mathbb{N}_0}^{\mathbf{K}(\mu)} \;. \tag{12}$$

Let $I' := I \cup \{i'\}$ and let $\mathbf{K}'$ denote the set of configurations $k : I' \to \mathbb{N}_0$ that can occur in any schedule for $N_L \cup \{j\}$. Then, $\mathbf{K}'(\mu)$, $\mathbf{S}'(\mu)$, $A'(\mu)$, and b' are defined analogously to $\mathbf{K}(\mu)$, $\mathbf{S}(\mu)$, $A(\mu)$, and b, respectively, with $\mathbf{K}$ replaced by $\mathbf{K}'$ and I replaced by I'.

Let $\mu' := \max\{\mu, \mathrm{opt}(N_L \cup \{j\})\} \leq 2\mu$. We are looking for a schedule $y' \in \mathbf{S}'(\mu')$, that is, y' must satisfy

$$A'(\mu')\, y' \;=\; b' \;, \qquad y' \in {\mathbb{N}_0}^{\mathbf{K}'(\mu')} \;. \tag{13}$$

Moreover, y' should be 'similar' to y. In order to compare the two vectors, we first 'lift' y to a vector in ${\mathbb{N}_0}^{\mathbf{K}'(\mu')}$ as follows. A configuration $k \in \mathbf{K}(\mu)$ can be interpreted as an element of $\mathbf{K}'(\mu')$ by defining $k(i) := 0$ for all $i \in I' \setminus I$. We then define

$$y_k \;:=\; 0 \qquad\qquad \text{for all } k \in \mathbf{K}'(\mu') \setminus \mathbf{K}(\mu) \;.$$

It follows from (12) that the extended vector y satisfies

$$A'(\mu')\, y \;=\; \hat{b} \;, \qquad y \in {\mathbb{N}_0}^{\mathbf{K}'(\mu')} \;. \tag{14}$$

The right hand side $\hat{b} \in {\mathbb{N}_0}^{1+|I'|}$ is defined as follows: If $I' = I$, then $\hat{b} = b$; otherwise, $I' = I \cup \{i'\}$ and we define the entry of vector $\hat{b}$ corresponding to i' to be zero and all other entries as in vector b.

Thus, y and y' are solutions to essentially the same integer linear program ((14) and (13), respectively) with slightly different right hand sides $\hat{b}$ and b', respectively. More precisely, the right hand sides are equal for all but one entry (the one corresponding to i') where they differ by 1.

Theorem 4 ([26, Corollary 17.2a]). *Let A be an integral $m \times n$-matrix, such that each sub-determinant of A is at most Δ in absolute value, let $\hat{b}$ and b' be column m-vectors, and let c be a row n-vector. Suppose $\max\{cx \mid Ax \leq \hat{b};\ x \text{ integral}\}$ and $\max\{cx \mid Ax \leq b';\ x \text{ integral}\}$ are finite. Then for each optimum solution y of the first maximum there exists an optimum solution y' of the second maximum such that*

$$\|y - y'\|_\infty \;\leq\; n\Delta\,(\|\hat{b} - b'\|_\infty + 2).$$

By Theorem 4 (choosing c to be zero vector), there exists a solution y' to (13) satisfying

$$\|y - y'\|_\infty \;\leq\; 3\,\big|\mathbf{K}'(\mu')\big|\,\Delta \;, \tag{15}$$

where Δ is an upper bound on the absolute value of any sub-determinant of the matrix $A'(\mu')$. To complete the proof, we have to show that the right hand side of (15) is constant.

First we give an upper bound on the number of columns $|\mathbf{K}'(\mu')|$, i.e., on the number of machine configurations with load at most μ'. Since each job has size at least

$$\epsilon\,\mathrm{lb}(N) \;\overset{(1)}{\geq}\; \frac{\epsilon}{2}\,\mathrm{opt}(N) \;\geq\; \frac{\epsilon}{2(1+\epsilon)}\,\mu \;\geq\; \frac{\epsilon}{4(1+\epsilon)}\,\mu' \;,$$

there are at most $\gamma := \lfloor 4(1+\epsilon)/\epsilon \rfloor \leq \frac{8}{\epsilon}$ jobs in any configuration $k \in \mathbf{K}'(\mu')$. In particular, $k(i) \leq \gamma$ for all $i \in I'$. This yields

$$\left|\mathbf{K}'(\mu')\right| \;\leq\; \gamma^{|I'|} \;\leq\; \gamma^{|I|+1} \leq \left(\frac{8}{\epsilon}\right)^{|I|+1} \overset{(9)}{\leq} \left(\frac{8}{\epsilon}\right)^{\frac{3}{\epsilon}\log(\frac{1+\epsilon}{\epsilon})} \leq \left(\frac{1+\epsilon}{\epsilon}\right)^{\frac{3}{\epsilon}\log(\frac{8}{\epsilon})} .$$

Finally, all entries in the first row of $A'(\mu')$ are 1 and the remaining entries are of the form $k(i) \leq \gamma$. Hence we bound Δ as follows. The maximum dimension of a square sub-matrix inside $A'(\mu')$ is at most the number of rows, i.e., $2 + |I|$ and, each entry in it has value at most γ. Hence the value of its determinant is upper-bound by $((2+|I|)\gamma)^{2+|I|}$. Thus

$$\Delta \;\leq\; \left((2+|I|)\,\gamma\right)^{2+|I|} \overset{(9)}{\leq} \left(\frac{8}{\epsilon^2}\right)^{2+|I|} \cdot \left(\frac{8}{\epsilon}\right)^{2+|I|} \overset{(9)}{\leq} \left(\frac{1+\epsilon}{\epsilon}\right)^{\frac{12}{\epsilon}\log(\frac{4}{\epsilon})}$$

Hence by (15) the number of machines touched is at most

$$3\left|\mathbf{K}'(\mu')\right|\Delta \;\leq\; 3\left(\frac{1+\epsilon}{\epsilon}\right)^{\frac{15}{\epsilon}\log(\frac{8}{\epsilon})} \tag{16}$$

and therefore is a constant. This concludes the proof. □

Theorem 5. *Let N be a set of jobs and let j be a new job not contained in N. Any $(1+\epsilon)$-approximate schedule for N can be turned into a $(1+\epsilon)$-approximate schedule for $N \cup \{j\}$ such that the total size of jobs that have to be moved is bounded by a constant $\beta(\epsilon)$ times p_j.*

Proof. We distinguish two cases. If the newly arrived job is small, i.e., $p_j < \epsilon\,\mathrm{lb}(N)$, then j can simply be assigned to the least loaded machine by Observation 1 and no job in N has to be moved.

It remains to consider the case $p_j \geq \epsilon\,\mathrm{lb}(N)$. The given schedule for N induces a schedule for N_L with makespan $\mu \leq (1+\epsilon)\,\mathrm{opt}(N) \leq (1+\epsilon)\,\mathrm{opt}(N \cup \{j\})$. By Lemma 3, the latter schedule can be turned into a schedule for $N_L \cup \{j\}$ with makespan at most

$$\max\{\mu, \mathrm{opt}(N_L \cup \{j\})\} \;\leq\; (1+\epsilon)\,\mathrm{opt}(N \cup \{j\}) \;,$$

by touching only a constant number of machines. In the following, this subset of machines of constant size is denoted by M'. We construct a schedule for $N \cup \{j\}$ as follows:

i) Start with the schedule for $N_L \cup \{j\}$ discussed above.

ii) The jobs in N_S that were assigned, by the given schedule for N, to one of the machines in $M \setminus M'$ are assigned to the same machine again.

iii) The remaining jobs in N_S are assigned one after another to the least loaded machine.

The makespan of the partial schedule constructed in steps i) and ii) is bounded by the maximum of the makespan of the given schedule for N and the optimal makespan of the schedule for $N_L \cup \{j\}$. It is thus bounded by $(1+\epsilon)\mathrm{opt}(N \cup \{j\})$. Assigning small jobs greedily to the least loaded machine in step iii) therefore results in a $(1+\epsilon)$-approximate schedule for $N \cup \{j\}$ by Observation 1.

Finally, notice that, in the whole process, only jobs that have initially been scheduled on machines M' are moved. The total size of these jobs is at most

$$(1+\epsilon)\mathrm{opt}(N)\,|M'| \overset{(1)}{\leq} 2(1+\epsilon)\,\mathrm{lb}(N)\,|M'| \leq 2p_j\left(\frac{1+\epsilon}{\epsilon}\right)|M'| \overset{(16)}{=} p_j\left(\frac{1+\epsilon}{\epsilon}\right)^{O\left(\frac{\log(2/\epsilon)}{\epsilon}\right)}.$$

This concludes the proof. □

Theorem 6. *There exists a $(1+\epsilon)$-competitive online algorithm with constant migration factor $\beta(\epsilon)$ such that the running time for scheduling a newly arrived job is constant.*

In particular, it follows from the last property mentioned in the theorem that the algorithm has linear running time.

Proof. The result on the competitive ratio follows from Theorem 5. It remains to show that upon arrival of a new job j, the schedule can be updated in constant time. We consider only the non-trivial case $\epsilon < 1$. We assume that for the current set of jobs N the following information is given:

- The total size of jobs $p(N)$, the maximum job size $p_{\max}(N)$, and the lower bound $\mathrm{lb}(N)$.
- For each machine, its load rounded down to the nearest integer power of $1+\frac{\epsilon}{2}$.

We argue that this information can be updated in constant time for the new set of jobs $N\cup\{j\}$. It is certainly easy to compute $p(N\cup\{j\}) := p(N)+p_j$, $p_{\max}(N\cup\{j\}) := \max\{p_{\max}(N), p_j\}$, and $\mathrm{lb}(N\cup\{j\}) := \max\{p(N\cup\{j\})/m, p_{\max}(N\cup\{j\})\}$. Since only constant number of machines are touched to incorporate the new job j, approximating the modified machine loads can also be done in constant time.

From Lemma 3, we recall the notion of a machine configuration $k(\cdot)$ with respect to the set of large jobs. In the following, we call a job small if its size is less than $\frac{\epsilon}{2}\mathrm{lb}(N)$ and large otherwise. This slightly modified definition is just a technicality and it only affects the bound on I and γ in Lemma 3, by a constant factor. Thus it only changes the bound (16), by a constant factor.

Similar to the arguments in Lemma 3, we argue that the number of machine configurations with each configuration having load at most $4\mathrm{lb}(N)$ is a constant. That is $|\mathbf{K}(4\mathrm{lb}(N))|$ is a constant. Each large job (belonging to N_L) has size at least $\frac{\epsilon}{2}\mathrm{lb}(N)$. Hence a machine configuration with total load at most $4\mathrm{lb}(N)$ has at most $\frac{8}{\epsilon}$ jobs from N_L. Each such job belongs to one of the job classes from I. Since the total number of large job classes $|I|$ is also a constant and the job classes are indexed from $\lceil\log_{1+\epsilon}(\frac{\epsilon}{2}\mathrm{lb}(N))\rceil =: i$ to $i+|I|$, we get:

Observation 4. *There are at most a constant number (say c) of configurations with each configuration having total load at most $4\mathrm{lb}(N)$, and we can enumerate them in constant time.*

The given schedule on N is represented using the following simple data structure. We assume that initially the schedule is given to us in this form. Later we show how to update it in constant time while scheduling job j. The machine configurations are represented using structures as shown in Figure 4. There is an array of *Config Head*s of dimension c, one for each possible configuration, and each *Config Head* points to the list of machines (list of *Machine*

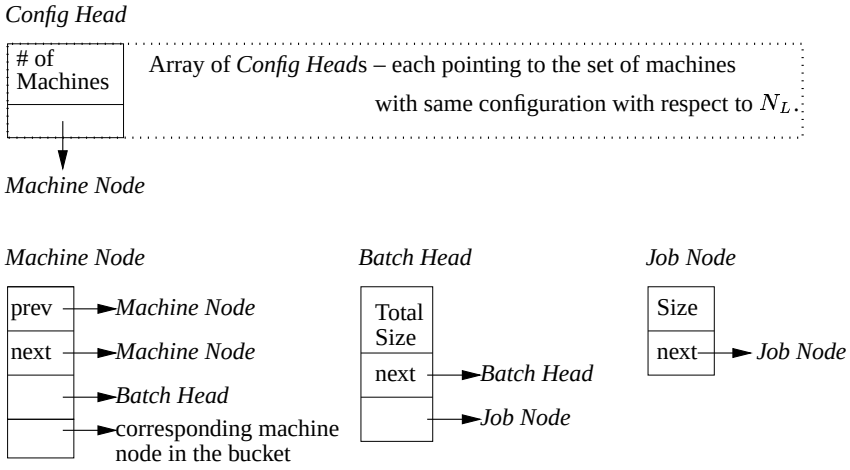


Figure 4: The different structures used in representing the schedule.

*Node*s) obeying that configuration. A *Machine Node* points to the list of jobs in it grouped as batches. The small jobs (belonging to N_S) in it are grouped into batches of size at most $\frac{\epsilon}{2}\mathrm{lb}(N)$ and the large jobs remain as a batch with single node. Clearly there are only a constant number of such batches in any machine. Each batch has a *Batch Head* that points to the list of jobs (*Job Node*s) in it.

Since $\mathrm{lb}(\cdot)$ is monotonically increasing, the machines belonging to the same configuration list still belong together in future, possibly in a different configuration list, as long as they are untouched. Thus while incorporating job j, *Config Head* array and its associated machine lists can be updated in constant time if a constant number of machines are touched and the pointers to their corresponding *Machine Node*s are available.

To find a machine with load at most $(1+\frac{\epsilon}{2})\mathrm{lb}(N)$ in constant time, we use bucketing. This is needed for assigning small jobs. The machines are partitioned into buckets based on the exponent of $1+\frac{\epsilon}{2}$ to which machine loads are rounded. All machines with approximate load at most $\mathrm{lb}(N)$ belong to bucket 0. Each machine belonging to bucket $i > 0$ has its approximate load ℓ such that $\frac{\mathrm{lb}(N)}{1+\epsilon/2} < \frac{\ell}{(1+\epsilon/2)^i} \leq \mathrm{lb}(N)$. Since the maximum load of a machine is at most $2\mathrm{opt}(N) \leq 4\mathrm{lb}(N)$, the number of buckets is at most a constant. A machine from bucket 0 can be found in constant time (bucket 0 is always non-empty). Observe that the untouched machines in a bucket stay together even in future (possibly in a new bucket) as $\mathrm{lb}(\cdot)$ is monotonically increasing. Thus the bucket structure can be updated in constant time while assigning job j if only a constant number of machines are touched. We assume that the bucket representation for N is also available in the beginning.

If the new job j is small ($p_j < \frac{\epsilon}{2}\mathrm{lb}(N)$), then

i) Consider any machine from bucket 0. Assigning job j to it changes its load to at most $(1+\epsilon)\mathrm{lb}(N) \leq (1+\epsilon)\mathrm{opt}(N \cup \{j\})$, as any machine in bucket 0 has load at most $(1+\frac{\epsilon}{2})\mathrm{lb}(N)$.

ii) Assign j to a batch of small jobs such that the total load in it does not exceed $\frac{\epsilon}{2}\mathrm{lb}(N)$. If no such batch exists, create a new batch for job j. Update the batches (merge small batches) with respect to $N \cup \{j\}$. There are only constant number of batches initially.

iii) Update the bucket structure with respect to $N \cup \{j\}$.

If the new job j is large then with respect to the input schedule y

a) Generate feasible schedules $z \in \mathbf{S}'\big(4\,\mathrm{lb}(N \cup \{j\})\big)$ by enumerating all vectors with constant distance $\|y - y'\|_\infty$ (see (15) and (16) of Lemma 3), where $y' \in \mathbf{S}'\big((1+\epsilon)\,\mathrm{opt}(N \cup \{j\})\big)$. Observe that $\mathbf{S}'\big((1+\epsilon)\,\mathrm{opt}(N \cup \{j\})\big) \subseteq \mathbf{S}'\big(4\,\mathrm{lb}(N \cup \{j\})\big)$. There are only constant number of such vectors and they can be generated in constant time by Observation 4. For each feasible vector do the following and choose the one minimizing the makespan.

b) The component wise difference between y and z specifies a subset of configurations and non-zero number of machines from each such configuration that should be modified. For each such configuration, we remove the required number of machines from the front of the machine list pointed to by the respective *Config Head*.

   – Remove small job batches (with respect to $N \cup \{j\}$) from these machines.
   – Reschedule the remaining jobs among these machines.
   – Assign these machines to the appropriate configuration lists.
   – Update the bucket structure with respect the new machine loads.
   – Each of the small batches are reassigned as in the small job case discussed above.

It is straightforward to verify that all of the above steps take only constant time. Thus we conclude the proof. □

7 The Two Machine Case

In this section we show a tight competitive ratio of $\frac{7}{6}$ for the two machine case. Consider the following procedure FILL_4.

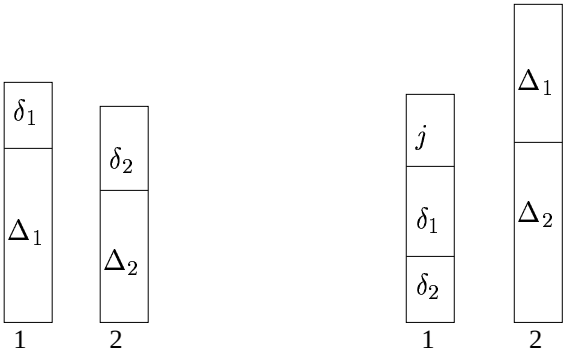
Procedure FILL_4:
Upon arrival of j, choose the option minimizing the makespan from the following options.

> For each fixed machine $i \in \{1, 2\}$, we define multiple options in the following way. Let L be the largest 3 jobs in machine i. Set L could possibly have less than three jobs. Let the remaining jobs in machine i be B. That is $B = S(i) \setminus L$. Let $\mathcal{L} \subseteq 2^L$ be set of all possible subsets (including ϕ) of L such that for each $L_k \in \mathcal{L}$, $p(L_k) \leq p_j$. Each set $L_k \in \mathcal{L}$ gives rise to a new
>
> *Option* $(i.k)$: Migrate jobs in L_k to the other machine. Consider the jobs in B in non-increasing order order of size and repeatedly remove them; stop before the total size of removed jobs exceeds $p_j - p(L_k)$. Let B_k denote these removed jobs. Assign job j to machine i and assign the removed jobs successively to the least loaded machine.
>
> *Option* 0: Assign job j to the least loaded machine.

Theorem 7. FILL_4 *is* $\frac{7}{6}$*–competitive with factor 1 migration.*

Proof. The migration factor is clear from the FILL_4 description. To show the competitive ratio, we consider an arbitrary *input schedule* on the set of jobs N that is $\frac{7}{6}$–approximate. We show that FILL_4 yields a $\frac{7}{6}$–approximate schedule for $N \cup \{j\}$.

If job j is such that $p_j \leq \mathrm{opt}'/3$ then option 0 already yield a $\frac{7}{6}$–competitive schedule by Observation 1. It remains to handle the *case* $p_j > \mathrm{opt}'/3$. From now on we assume that Option 0 does not suffice. As the largest three jobs are not included in set B we have,



Input Schdule on N Optimal Schedule on $N \cup \{j\}$

Figure 5: Comparison of input schedule on N and optimal schedule on $N \cup \{j\}$, for two machines.

Observation 5. *For any machine i consider the set B as defined in* FILL_4*. Every job in it has size at most* $\mathrm{opt}'/3$.

Observation 6. *Migrating jobs of total size at most $p_j - \mathrm{opt}'/3$ from any fixed machine and assigning job j there yields a $\frac{7}{6}$–approximate schedule for $N \cup \{j\}$.*

Proof. Let machine 2 be the higher loaded machine and let $p(S(2)) = \mathrm{opt}' - p_j/2 + y$. This implies that $p(S(1)) \leq \mathrm{opt}' - p_j/2 - y$. Merely assigning job j to machine 1 fail only if the final load difference exceeds $\mathrm{opt}'/3$. For this it is necessary that $y < p_j/2 - \mathrm{opt}'/6$ and $p_j > \mathrm{opt}'/3$. The total migration amount needed is either $p_j/2 - \mathrm{opt}'/6$ (from machine 1 to 2) or $p_j/2 + y - \mathrm{opt}'/6$ (from 2 to 1), which is at most $p_j - \mathrm{opt}'/3$. □

Though above observation is true, it is possible that every feasible set of jobs that needs to be migrated have total size more than $p_j - \mathrm{opt}'/3$.

Fact 2. *There is a subset of jobs of total size at most p_j residing in a machine such that scheduling job j here and migrating this subset to the other machine yields a $\frac{7}{6}$–approximate schedule.*

It might take exponential time in the worst case to identify the subset that needs to be migrated. Using Observation 6 we show that our polynomial time strategy also works. Let X denote the *set with the smallest total size* that needs to be migrated from machine i according to Fact 2. Consider the sets L and B for machine i as defined in FILL_4. Let $X \cap L = L_k$. Observe that $L_k \in \mathcal{L}$. We show that Option $(i.k)$ yields a $\frac{7}{6}$–approximate schedule. The set of jobs that are either migrated or removed from machine i before assigning j is $L_k \cup B_k$. Observe that $p(L_k \cup B_k)$ is at least $p_j - \mathrm{opt}'/3$ unless $B_k = B$ as size of any job in B is at most $\mathrm{opt}'/3$ by Observation 5. Thus in any case $p(X) \leq p(L_k \cup B_k)$ as $p(X) \leq p_j - \mathrm{opt}'/3$ by Observation 6. Thus by Observation 6 the total load in machine i after assigning job j, i.e., $p(S(i) \setminus (L_k \cup B_k)) + p_j$, is at most $\frac{7}{6}\mathrm{opt}'$. The reassignment of jobs in B_k (each have size at most $\mathrm{opt}'/3$) is also fine by Observation 1. □

Proof. (Fact 2) It is clear that there is no need to migrate a total size more than p_j (simply assign p_j on the destination machine instead). Fix any optimal schedule of $N \cup \{j\}$. We capture the difference between this optimal schedule and the input schedule on N by sets $\delta_1, \delta_2, \Delta_1$ and

Δ_2. As shown in Figure 5, the above four sets define a partition of N. The set δ_1 is the set of all jobs assigned to machine 1 on both schedules. Similarly, set Δ_2 is the set of all jobs assigned to machine 2 on both schedules. The remaining two sets capture the differences between these two schedules except for job j, which is only present in the optimal schedule.

We consider only the interesting case that assigning j to any machine without migration fails. From now, for convenience, we denote the size of any set X as simply X. For job j we denote its size as j. We also normalize opt' to 1. The optimal schedule (Figure 5) and the fact that assigning j without migration fails implies

$$(1)\ \delta_1 + \delta_2 + j \leq 1 \qquad (3)\ \delta_1 + \Delta_1 + j > 7/6$$
$$(2)\ \Delta_1 + \Delta_2 \leq 1 \qquad (4)\ \delta_2 + \Delta_2 + j > 7/6$$

It is straightforward to verify that these four inequalities yield $j > 2/3$ and hence $\delta_1 + \delta_2 \leq 1/3$ (Inequalities 1 and 3 imply $\Delta_1 > 1/6 + \delta_2$. Inequalities 2 and 4 imply $\delta_2 + j > 1/6 + \Delta_1$). W.l.o.g let $\delta_1 \leq 1/6$. Hence the migration strategy is; migrate Δ_2 to machine 1 and schedule j on machine 2. □

Tight Lower Bound

Theorem 8. *Let A be any deterministic algorithm that is c–competitive on two machines with migration factor at most one. Then $c \geq \frac{7}{6(1+\epsilon)}$ for any sufficiently small positive $\epsilon \in \mathbb{R}^+$.*

Proof. The adversary initially issues four jobs with the following size: $1/6+\epsilon/2, 1/6+\epsilon/2, 1/2$ and $1/2$. It is easy to verify that the only $\frac{7}{6(1+\epsilon)}$–approximate way of scheduling them is to assign in each machine one job of size $1/6 + \epsilon/2$ and one job of size $1/2$.

Assume that both machines contain one job of size $1/2$ and one job of size $1/6 + \epsilon/2$. Now adversary issues a new job of size $2/3$. The optimal makespan is $1 + \epsilon$. But if the migration factor is restricted to 1, then the best possible makespan by A is $7/6$. Hence $c = \frac{7}{6(1+\epsilon)}$. □

8 Maximizing the Minimum Machine Load

An alternative, yet less frequently used objective for machine scheduling is to maximize the minimum load. However, we have a concrete application using this objective function that was the original motivation for our interest in bounded migration: Storage area networks (SAN) usually connect many disks of different capacity and grow over time. A convenient way to hide the complexity of a SAN is to treat it as a single big, fault tolerant disk of huge capacity and throughput [7, 25]. A simple scheme with many nice properties implements this idea if we manage to partition the SAN into several sub-servers [25] of about equal size. Mapping to the scheduling framework, the contents of disks correspond to jobs and the sub-servers correspond to machines. Each sub-server stores the same amount of data. For example, if we have two sub-servers, each of them stores all the data to achieve a fault tolerance comparable to mirroring in ordinary RAID level 0 arrays [22]. More sub-servers allow for a more flexible tradeoff between fault tolerance, redundancy, and access granularity. In any case, the capacity of the server is determined by the *minimum* capacity of a sub-server. Moreover, it is not acceptable to completely reconfigure the system when a new disk is added to the system or when a disk fails. Rather, the user expects a "proportionate response", i.e., if she adds a disk of x GByte she will not be astonished if the system moves data of this order of magnitude but she would

complain if much more is moved. Our theoretical investigation confirms that this 'common sense' expectation is indeed reasonable.

We concentrate on the case without job departures (disk failures). We show that the following simple strategy, which is very similar to FILL_1, is 2–competitive already for migration factor $\beta = 1$.

Procedure FILL_5:
Upon arrival of a new job j, do the following. Repeatedly remove jobs from the least loaded machine $i_{\min}$; stop before the total size of removed jobs exceeds p_j. Assign job j to machine $i_{\min}$. Assign the removed jobs successively to the least loaded machine.

Theorem 9. FILL_5 *is* 2*–competitive with migration factor* 1.

Proof. The migration factor is clear from the description of FILL_5. In the input schedule on N, consider the *maximum loaded machine among those containing multiple jobs*. If there is no such machine (i.e, every machine has at most one job) then the schedule after assigning job j is 1–approximate (optimal). Hence the interesting case is that such machines exist. We call such machines as *multi-job* machines.

We assume that the following property holds for the input schedule;

$$\text{maximum load of a multi-job machine} \;\leq\; 2 \cdot \text{ minimum load} \tag{17}$$

Later we show that the above property is preserved on the output schedule. An output schedule with above property is a 2–approximate schedule as the optimal *minimum* load is at most the maximum load of a multi-job machine. Thus it remains to show that property (17) holds for the output schedule. If $p_j \leq p(S(i_{\min}))$ then this is straightforward. In case $p_j > p(S(i_{\min}))$, initially all jobs from the least loaded machine are removed and j is assigned there. This intermediate schedule (before reassigning the removed jobs) satisfies property (17). Observe that each of the removed jobs has size at most the intermediate minimum load. Hence reassigning them still preserves the property as shown above. □

Negative Result

The following lemma shows that it is not possible to start with an arbitrary 2–approximate schedule on N and obtain a 2–approximate schedule for $N \cup \{j\}$ with constant migration factor.

Lemma 4. *There is a* 2*–approximate schedule on* m *machines such that upon the arrival of a new job, it is not possible for any strategy to obtain* 2*–approximate schedule with migration factor less than* $m - 2$.

Proof. In the initial schedule machine 1 has $2m$ jobs of size 1. All the remaining $m-1$ machines have one job of size 1. In total there are $3m - 1$ jobs. The optimal minimum load is 2. Hence this is a 2–approximate schedule. A new job of size 1 arrives. The new optimal minimum is 3. To achieve minimum load greater than 1, any strategy has to move at least $m - 2$ jobs from machine 1. □

Acknowledgments. We would like to thank Gerhard Woeginger for interesting discussions and helpful comments on the topic of this paper.

References

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 32–43, New York City, NY, 1999.

[2] S. Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29:459–473, 1999.

[3] S. Albers. Online algorithms: a survey. *Mathematical Programming*, 97:3–26, 2003.

[4] M. Andrews, M.X. Goemans, and L. Zhang. Improved bounds for on-line load balancing. *Algorithmica*, 23:278–301, 1999.

[5] Y. Azar and L. Epstein. On-line machine covering. *Journal of Algorithms*, 1:67–77, 1998.

[6] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. *Journal of Computer and System Sciences*, 51:359–366, 1995.

[7] A. Brinkmann, K. Salzwedel, and C. Scheideler. Compact, adaptive placement schemes for non-uniform requirements. In *14th ACM Symposium on Parallel Algorithms and Architectures*, pages 53–62, 2002.

[8] B. Chen, A. van Vliet, and G. J. Woeginger. Lower bounds for randomized online scheduling. *Information Processing Letters*, 51:219–222, 1994.

[9] E. G. Coffman Jr., M. R. Garey, and D. S. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7:1–17, 1978.

[10] R. Fleischer and M. Wahl. Online scheduling revisited. *Journal of Scheduling*, 3:343–353, 2000.

[11] D. K. Friesen. Tighter bounds for the multifit processor scheduling algorithm. *SIAM Journal on Computing*, 13:170–181, 1984.

[12] M. R. Garey and D. S. Johnson. Strong np-completeness results: Motivation, examples and implications. *Journal of the ACM*, 25:499–508, 1978.

[13] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[14] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:263–269, 1969.

[15] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[16] D. S. Hochbaum. Various notions of approximation: Good, better, best, and more. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, chapter 9, pages 346–398. Thomson, 1996.

[17] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM*, 34:144–162, 1987.

[18] D. R. Karger, S. J. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 20:400–430, 1996.

[19] M. A. Langston. *Processor scheduling with improved heuristic algorithms.* PhD thesis, Texas A&M University, 1981.

[20] H. W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.

[21] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley & Sons, New York, 1988.

[22] D. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks (RAID). *Proceedings of ACM SIGMOD'88*, pages 109–116, 1988.

[23] J. F. Rudin III. *Improved bounds for the on-line scheduling problem*. PhD thesis, The University of Texas at Dallas, 2001.

[24] S. Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM*, 23:116–127, 1976.

[25] P. Sanders. Algorithms for scalable storage servers. In *SOFSEM 2004: Theory and Practice of Computer Science*, volume 2932, pages 82–101, 2004.

[26] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Chichester, 1986.

[27] J. Sgall. A lower bound for randomized on-line multiprocessor scheduling. *Information Processing Letters*, 63(1):51–55, 14 July 1997.

[28] J. Sgall. On-line scheduling — a survey. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*, pages 196–231. Springer, Berlin, 1998.

[29] J. Westbrook. Load balancing for response time. *J. Algorithms*, 35(1):1–16, 2000.