

Applying Extra-Resource Analysis to Load Balancing

Mark Brehob, Eric Torng, and Patchrawat Uthaisombut

Department of Computer Science and Engineering
Michigan State University

March 24, 2000

Coresponding author:

Eric Torng

Department of Computer Science and Engineering

3115 Engineering Building

Michigan State University

East Lansing, MI 48824

USA

torng@cse.msu.edu

A preliminary version of this work appeared in SODA2000 Proceedings.

This work was supported by NSF CAREER grant CCR-9701679.

Applying Extra-Resource Analysis to Load Balancing^{1 2}

Abstract

Previously, extra-resource analysis has been used to argue that certain on-line algorithms are good choices for solving specific problems because these algorithms perform well with respect to the optimal off-line algorithm when given extra resources. We now introduce a new application for extra-resource analysis: deriving a qualitative divergence between off-line and on-line algorithms. We do this for the load balancing problem, the problem of assigning a list of jobs on m identical machines to minimize the makespan, the maximum load on any machine. We analyze the worst-case performance of on-line and off-line approximation algorithms relative to the performance of the optimal off-line algorithm when the approximation algorithms have k extra machines. Our main results are the following: The **Longest-Processing-Time** (\mathcal{LPT}) algorithm will produce a schedule with makespan no larger than that of the optimal off-line algorithm if \mathcal{LPT} has at least $(4m - 1)/3$ machines while the optimal off-line algorithm has m machines. In contrast, no on-line algorithm can guarantee the same with any number of extra machines.

Keywords: load balancing, extra-resource analysis, longest-processing-time algorithm, on-line algorithm, approximation algorithm

¹A preliminary version of this work appeared in SODA2000 Proceedings.

²This work was supported by NSF CAREER grant CCR-9701679.

1 Introduction

In the past few years, the extra-resource analysis technique popularized by Kalyanasundaram and Pruhs [19] has been used for analyzing the performance of on-line algorithms for problems where traditional competitive analysis yields non-constant competitive ratios. In many cases, the traditional competitive analysis does not differentiate between good and bad algorithms. For these problems, extra-resource analysis has been used to argue that certain on-line algorithms are good choices for solving specific problems because they perform well with respect to the optimal off-line algorithm when given extra resources [18, 25, 5, 20, 22, 24, 2, 11].

In this paper, we use extra-resource analysis to provide us with more insight into the behavior of specific load balancing algorithms. This leads us to a qualitative divergence between on-line and off-line algorithms for the load balancing problem. This result also reemphasizes the value of sorting by job size before performing list scheduling.

1.1 Problem Definitions

In the load balancing problem, we are given m identical machines and a job list I of n jobs. All jobs arrive at time 0. For $j = 1, \dots, n$, job j has a processing time p_j . Let $p_{\min} = \min_{1 \leq j \leq n} p_j$. An assignment algorithm chooses a machine i for each job j , and that job runs on that machine until completion. Let $s_j^{\mathcal{A}}(m, I)$ and $C_j^{\mathcal{A}}(m, I)$ denote the starting time and the completion time, respectively, of job j in the schedule produced by algorithm \mathcal{A} for job list I on m machines. Note that $C_j^{\mathcal{A}}(m, I) = s_j^{\mathcal{A}}(m, I) + p_j$. Let $C_{\max}^{\mathcal{A}}(m, I)$ denote the makespan, the maximum load on any machine, of the schedule produced by algorithm \mathcal{A} for job list I on m machines, which is defined as $C_{\max}^{\mathcal{A}}(m, I) = \max_j C_j^{\mathcal{A}}(m, I)$. We will drop the arguments m and/or I if there is no ambiguity. The goal of the load balancing problem is to minimize the makespan.

An algorithm is *on-line* if it must permanently assign the current job to a machine before it is aware of the next job in the list. An algorithm is *off-line* if it is aware of the entire job list in advance. We say that an algorithm \mathcal{A} is an (m, k) -machine ρ -approximation algorithm if

$$\sup_I \frac{C_{\max}^{\mathcal{A}}(m+k, I)}{C_{\max}^{\mathcal{OPT}}(m, I)} \leq \rho$$

where \mathcal{OPT} denotes the optimal off-line algorithm. We say that an algorithm \mathcal{A} is ρ -approximation if \mathcal{A} is an $(m, 0)$ -machine ρ -approximation for all $m \geq 1$. We say that an algorithm \mathcal{A} is (m, k) -optimal if \mathcal{A} is an (m, k) -machine 1-approximation algorithm.

We study two algorithms in particular. The **List-Scheduling** algorithm (\mathcal{LS}) runs the next job in the list on the machine with the smallest load. The **Longest-Processing-Time** algorithm (\mathcal{LPT}) runs the *longest* unscheduled job on the machine with the smallest load. \mathcal{LPT} is an off-line algorithm while \mathcal{LS} is an on-line algorithm.

1.2 Related Results

The load balancing problem is NP-hard [12]. Graham showed that \mathcal{LS} is an $(2 - \frac{1}{m})$ -approximation algorithm [14] and \mathcal{LPT} is an $(\frac{4}{3} - \frac{1}{3m})$ -approximation algorithm [15] for any $m \geq 1$. Hochbaum and Shmoys devised a polynomial time approximation scheme (PTAS) for this problem [16]. For the on-line setting, Albers introduced a deterministic on-line 1.923-approximation algorithm, and

she proved a deterministic on-line lower bound of 1.852 [1]. Recently, the lower bound was improved to 1.85358 by Gormley, Reingold, Torng, and Westbrook [13].

Kalyanasundaram and Pruhs were the first to explicitly use *extra-resource analysis* of on-line algorithms [19] with the goal of identifying good on-line algorithms in settings where traditional competitive analysis fails to do so. The extra-resource analysis technique is a relaxed notion of competitive analysis in which the on-line algorithm has more resources than the optimal off-line algorithm to which it is compared. Following this paper, several more studies have used the extra-resource analysis technique in a wide variety of settings [18, 25, 5, 20, 22, 24, 11, 2].

Azar, Epstein, and van Stee have independently and in parallel considered the problem of load balancing with extra resources [4]. In contrast to our work, they only consider on-line algorithms. Their main result is an on-line load balancing algorithm with an approximation ratio which decays exponentially in $(m+k)/m$ as well as a nearly matching lower bound. They also considered other problem features such as allowing a job to be scheduled on multiple machines, temporary tasks, and related and unrelated processors. We only consider identical machines and permanent tasks which must be scheduled on only a single machine.

Another approach for identifying good on-line algorithms when competitive analysis fails to yield useful results is to restrict the set of legal input instances in some form. This has been proposed and used in a variety of settings [6, 23, 7, 8]. In contrast, extra-resource analysis compares the performance of on-line/off-line approximation algorithms to that of the optimal off-line algorithm on *all* possible input instances; however, the approximation algorithms are given more resources than the optimal off-line algorithm.

Our problem is also related to the well studied and NP-hard bin packing problem [9, 12]. In the bin packing problem, we have to pack items into a minimum number of bins of known size. An item can be placed into a bin only if it fits in the available space. Johnson *et al.* prove that two simple on-line bin packing algorithms, **First-Fit** and **Best-Fit**, are $(\frac{17}{10}m + 2)$ -approximation algorithms [17]. Richey introduced an on-line algorithm **Harmonic+1** which has an asymptotic worst case ratio smaller than 1.5888 [26]. Karp and Karmarkar devised an asymptotic fully polynomial-time approximation scheme (asymptotic FPTAS) for off-line bin packing [21].

Dell’Olmo *et al.* studied off-line bin packing with extendable bins [10]. In this problem, we have to pack items into a fixed number of bins of known size. However, the size of each bin can be extended if needed. The “final” size of a bin is the maximum between the original bin size and the total size of items in that bin. The goal is to minimize the *sum* of the final size of all bins. In contrast, the goal of load balancing is to minimize the *maximum* load on any machine. Speranza and Tuza studied the on-line version of the bin-packing problem with extendable bins [27].

Azar and Regev studied the on-line bin-stretching problem [3]. In this problem, the number of bins is fixed, and we wish to pack all the items into the bins while we stretch the size of the bins as little as possible. The on-line algorithm is presented with one item at a time, and it has to pack the item before the next item is presented to it. This problem is equivalent to the load balancing problem except that the optimal makespan (maximum load) is known in advance. In contrast, we study the load balancing problem where approximation algorithms do not know the optimal makespan. Furthermore, in our study, approximation algorithms have more machines (bins) than the optimal off-line algorithm. In a way similar to the work by Azar and Regev, we compare the makespan (bin stretch) of the schedules produced by approximation algorithms and the optimal off-line algorithm.

The property of an algorithm being (m, k) -optimal for the load balancing problem with extra

machines is similar to being an $\frac{m+k}{m}$ -approximation algorithm for the bin packing problem. Both types of algorithms can pack items into $m + k$ bins when the optimal off-line algorithm needs m bins. Load balancing algorithms know the number of bins used, but they do not know the optimal makespan. In contrast, bin packing algorithms know the bin size but do not know the number of bins used by the optimal algorithm. Thus, an (m, k) -optimal load balancing algorithm can be thought of as an algorithm for solving a bin packing problem with unknown bin size but with a specified number of bins.

1.3 Summary of Results

The extra-resource analysis technique has been used to derive insight into the behavior of on-line and off-line algorithms. Previously, these insights have been used to identify good on-line algorithms in settings where traditional competitive analysis fails to do so. In this work, we show that these insights can also be used to derive a qualitative divergence between off-line and on-line load balancing algorithms.

In Section 2, we give on-line results. We begin by extending Graham's results [14] on the performance of \mathcal{LS} for the load balancing problem to the case when \mathcal{LS} has $m + k$ machines while \mathcal{OPT} has m machines. We show that \mathcal{LS} is a (m, k) -machine $(1 + \frac{m-1}{m+k})$ -approximation algorithm. We give a lower bound instance to show that this result is tight. The lower bound for \mathcal{LS} can be generalized to apply to all on-line algorithms. In particular, we prove that no on-line algorithm is (m, k) -optimal for any k . It should be noted that although no on-line algorithm is (m, k) -optimal for any k , the makespan of the on-line schedule could be asymptotically close to the optimal makespan as can be seen from the performance guarantee of \mathcal{LS} .

In Section 3, we give off-line upper bound results. By extending Graham's result [15] on \mathcal{LPT} , we show that \mathcal{LPT} is an (m, k) -machine $(\max\{\frac{4m+k-1}{3m+3k}, 1\})$ -approximation algorithm. This implies that \mathcal{LPT} is (m, k) -optimal when $k \geq \frac{m-1}{2}$; that is, if \mathcal{LPT} has at least $\frac{3m-1}{2}$ machines and \mathcal{OPT} has m machines, then \mathcal{LPT} will produce a schedule with a makespan no larger than that of \mathcal{OPT} .

Next, we refine this result to show that \mathcal{LPT} is (m, k) -optimal when $k \geq \frac{m-1}{3}$. The proof is based on a seemingly trivial but important fact that the sum of the processing times of jobs on each machine in the optimal schedule is no larger than the optimal makespan. To exploit this fact, new ideas are required which we now briefly summarize. The main idea is to not classify jobs by their absolute sizes, but to classify each job in a schedule by (1) the number of jobs on the same machine and (2) the order of its size among jobs on the same machine. The classification allows us to establish some crucial relationships between the optimal schedule and the \mathcal{LPT} schedule. These relationships enable us to identify a most heavily loaded machine y in the \mathcal{LPT} schedule and some machine z in the optimal schedule such that the i th largest job on machine y is no larger than the i th largest job on machine z . Therefore, the makespan of the \mathcal{LPT} schedule is no larger than the makespan of the optimal schedule.

Results in Sections 2 and 3 imply a divergence between off-line and on-line algorithms because simple off-line algorithms such as \mathcal{LPT} guarantees (m, k) -optimality with only a few extra machines whereas no on-line algorithm can guarantee (m, k) -optimality with any number of extra machines. This result also underscores the value of sorting before performing list scheduling. Namely, if the list is sorted in non-increasing order (\mathcal{LPT}), then we can achieve (m, k) -optimality with $k = \frac{m-1}{3}$. If, however, the list is not sorted in non-increasing order, (m, k) -optimality through list scheduling cannot be guaranteed for any value of k .

Results in Section 2 also imply a difference between load balancing and bin packing. Consider an (m, k) -optimal load balancing algorithm and an $\frac{m+k}{m}$ -approximation bin packing algorithm. Both algorithms can pack all items into $m + k$ bins without overpacking any bin whereas the optimal algorithm needs m bins. However, each of them knows different pieces of information. A load balancing algorithm knows how many bins is needed by the optimal algorithm, but it does not know the optimal makespan (bin size). A bin packing algorithm knows the bin size but it does not know how many bins is needed by the optimal algorithm. The performance achievable by on-line algorithms in each of the problems are quite different. No on-line load balancing algorithm can guarantee not to overpack the bins. In contrast, simple on-line bin packing algorithms such as **First-Fit** and **Best-Fit** need only $\frac{17}{10}m + 2$ bins to pack all items [17]. This result indicates that the optimal makespan (bin size) is a more important piece of information than the optimal number of bins.

In Section 4, we describe a procedure to compute a good, though not necessarily optimal, lower bound of the performance of \mathcal{LPT} for any m and k .

2 On-line Results

We begin this section by proving the following lemma which is an extension of Graham's analysis for \mathcal{LS} [14] and \mathcal{LPT} [15] to the case when list scheduling algorithms have k extra machines. Note that the lemma applies to any list scheduling algorithm.

Lemma 2.1. *For any instance I and any $k \geq 0$, if job l is the last job to finish in the \mathcal{LS} schedule,*

$$\text{then} \quad C_{\max}^{\mathcal{LS}}(m+k, I) \leq \left(\frac{m}{m+k}\right) C_{\max}^{\mathcal{OPT}}(m, I) + \left(\frac{m+k-1}{m+k}\right) p_l.$$

Proof. All machines must be busy up to time $s_l^{\mathcal{LS}}$ when job l is scheduled, otherwise job l could have been started earlier. The starting time $s_l^{\mathcal{LS}}$ of job l is upper bounded by the average load on $m+k$ machines just before job l is scheduled. Thus, $s_l^{\mathcal{LS}} \leq \frac{1}{m+k}((\sum_{j=1}^n p_i) - p_l) = \frac{1}{m+k} \sum_{j=1}^n p_i - \frac{1}{m+k} p_l$. The optimal makespan on m machines is lower bounded by the average load on m machines after all jobs are scheduled. Therefore, $C_{\max}^{\mathcal{OPT}}(m) \geq \frac{1}{m}(\sum_{j=1}^n p_i)$.

$$\begin{aligned} C_{\max}^{\mathcal{LS}}(m+k, I) &= s_l^{\mathcal{LS}} + p_l \\ &\leq \frac{1}{m+k} \sum_{j=1}^n p_i - \frac{1}{m+k} p_l + p_l \\ &\leq \left(\frac{m}{m+k}\right) C_{\max}^{\mathcal{OPT}}(m, I) + \left(\frac{m+k-1}{m+k}\right) p_l. \end{aligned}$$

□

Theorem 2.1. *For $m \geq 1, k \geq 0$, and any job list I ,*

$$C_{\max}^{\mathcal{LS}}(m+k, I) \leq \left(1 + \frac{m-1}{m+k}\right) C_{\max}^{\mathcal{OPT}}(m, I),$$

and this is tight.

Proof. Suppose job l is the last job to finish in the \mathcal{LS} schedule. By Lemma 2.1 and the fact that no job has a processing time larger than $C_{\max}^{\mathcal{OPT}}$,

$$\begin{aligned} C_{\max}^{\mathcal{LS}}(m+k, I) &\leq \left(\frac{m}{m+k}\right) C_{\max}^{\mathcal{OPT}}(m, I) + \left(\frac{m+k-1}{m+k}\right) C_{\max}^{\mathcal{OPT}}(m, I) \\ &= \left(1 + \frac{m-1}{m+k}\right) C_{\max}^{\mathcal{OPT}}(m, I). \end{aligned}$$

The tightness of this result can be seen by considering an instance that begins with $(m-1)(m+k)$ jobs of size 1 and ends with a job of size $m+k$. \mathcal{LS} would schedule $m-1$ jobs of size 1 on each of the $m+k$ machines and schedule the job of size $m+k$ on one of the machines. Thus, its makespan would be $2m+k-1$. The optimal schedule would schedule $m+k$ jobs of size 1 on $m-1$ machines and schedule the job of size $m+k$ on the remaining machine. Thus, its makespan would be $m+k$. \square

The lower bound for \mathcal{LS} can be generalized to apply to all on-line algorithms.

Theorem 2.2. *No on-line algorithm is (m, k) -optimal for $m \geq 2$ and $k \geq 0$.*

Proof. The idea of the proof is that the adversary will keep generating new jobs until the on-line algorithm schedules a new job on a non-empty machine. Fix $m \geq 2, k \geq 0$, and an on-line algorithm \mathcal{A} . The size of jobs generated by the adversary can be described as follows. The first job has size 4. The size of each subsequent job is equal to the size of all the previous jobs combined. Therefore, $p_1 = 4$ and $p_j = 2^j$ for $j \geq 2$. If the adversary has generated l jobs, then the optimal schedule for these l jobs is to schedule job l by itself and to schedule the other jobs arbitrarily on the other machines. In fact, the adversary can schedule jobs 1 through $l-1$ on the same machine. The optimal makespan for these l jobs is 2^l which is the size of the latest job generated by the adversary and is also equal to the sum of the size of jobs 1 through $l-1$.

Suppose job l is the first job that the on-line algorithm schedules on a non-empty machine. Since the on-line algorithm has only $m+k$ machines, then $l \leq m+k+1$. The adversary would generate no more jobs after job l . From the argument above, the optimal makespan is 2^l , the size of job l . Since the on-line algorithm schedules job l on a non-empty machine, then the on-line makespan is strictly greater than 2^l . Thus, the result follows. \square

3 Off-Line Upper Bounds

We begin this section by extending Graham's analysis for \mathcal{LPT} [15] to the case when \mathcal{LPT} has k extra machines.

Theorem 3.1. *For $m \geq 1, k \geq 0$, and any job list I ,*

$$\frac{C_{\max}^{\mathcal{LPT}}(m+k, I)}{C_{\max}^{\mathcal{OPT}}(m, I)} \leq \begin{cases} \frac{4m+k-1}{3m+3k}, & 0 \leq k \leq \frac{m-1}{2} \\ 1, & k \geq \frac{m-1}{2}. \end{cases}$$

Proof. The proof is very similar to the case where both \mathcal{LPT} and \mathcal{OPT} have m machines [15]. Suppose job l is the last job to finish in the \mathcal{LPT} schedule. Thus, $C_{\max}^{\mathcal{LPT}} = s_l^{\mathcal{LPT}} + p_l$. We can

assume that job l is the last job to *start* in the \mathcal{LPT} schedule. If this is false, we can remove all jobs i such that $s_i^{\mathcal{LPT}} \geq s_l^{\mathcal{LPT}}$. This does not change the makespan of the \mathcal{LPT} schedule because these jobs must have run on machines other than the one job l does. Moreover, this cannot increase the optimal makespan. Since \mathcal{LPT} schedules job l last, then job l is the *smallest* job, *i.e.*, $p_l = p_{\min}$. There are two cases.

Case 1: $p_{\min} \leq \frac{1}{3}C_{\max}^{\mathcal{OPT}}$.

Since l is the last job to finish in the \mathcal{LPT} schedule, then by Lemma 2.1,

$$C_{\max}^{\mathcal{LPT}} \leq \left(\frac{m}{m+k}\right) C_{\max}^{\mathcal{OPT}} + \left(\frac{m+k-1}{m+k}\right) \frac{1}{3} C_{\max}^{\mathcal{OPT}} = \left(\frac{4m+k-1}{3m+3k}\right) C_{\max}^{\mathcal{OPT}}.$$

Case 2: $p_{\min} > \frac{1}{3}C_{\max}^{\mathcal{OPT}}$.

Thus, $p_i > \frac{1}{3}C_{\max}^{\mathcal{OPT}}$ for all i . Therefore, in the optimal schedule, there are at most 2 jobs on each machine. If $n \leq m$, then the optimal algorithm schedules one job per machine. If $m+1 \leq n \leq 2m$, we claim that for $j = 1, \dots, m$, the optimal algorithm schedules job j on the same machine as job $2m+1-j$ if $2m+1-j \leq n$, and schedules job j by itself otherwise. The optimality can be proven by an interchange argument. Furthermore, \mathcal{LPT} would produce a similar schedule on $m+k$ machines. If $n \leq m+k$, then \mathcal{LPT} schedules one job per machine. If $m+k+1 \leq n \leq 2(m+k)$, then for $j = 1, \dots, m+k$, \mathcal{LPT} schedules job j on the same machine as job $2(m+k)+1-j$ if $2(m+k)+1-j \leq n$, and schedules job j by itself otherwise. Thus, $C_{\max}^{\mathcal{LPT}} \leq C_{\max}^{\mathcal{OPT}}$.

$$\begin{aligned} \text{From Cases 1 and 2,} \quad C_{\max}^{\mathcal{LPT}} &\leq \max\left\{\frac{4m+k-1}{3m+3k}, 1\right\} C_{\max}^{\mathcal{OPT}} \\ \text{and} \quad \frac{4m+k-1}{3m+3k} > 1 &\iff k < \frac{m-1}{2} \end{aligned}$$

□

Corollary 3.1. \mathcal{LPT} is (m, k) -optimal if $k \geq \frac{m-1}{2}$.

Proof. Immediate from Theorem 3.1. □

We can refine the above result and get the following theorem.

Theorem 3.2. \mathcal{LPT} is (m, k) -optimal if $k \geq \frac{m-1}{3}$.

Proof. Assume that the theorem is not true. Then there must be some values of k and m for which the theorem statement does not hold. Let us fix a value of k for which the theorem statement does not hold, and let us fix m to be the minimum possible m such that the theorem does not hold for m and k . Note that $m \leq 3k+1$ because otherwise the theorem is true. Finally, given k and m , define I to be a smallest counterexample; that is, an input instance with the minimum possible number of jobs such that $C_{\max}^{\mathcal{LPT}}(m+k, I) > C_{\max}^{\mathcal{OPT}}(m, I)$. We will now show that this counterexample cannot exist, and thus the theorem follows.

Suppose that jobs in I are labeled according to the order that they are scheduled by \mathcal{LPT} . Thus, $p_1 \geq p_2 \geq \dots \geq p_n$. We first observe that job n is the only job in \mathcal{LPT} 's schedule that finishes later than the last job in \mathcal{OPT} 's schedule; that is, $C_{\max}^{\mathcal{LPT}}(m+k, I) = C_n^{\mathcal{LPT}}(m+k, I) > C_{\max}^{\mathcal{OPT}}(m, I)$ and $C_i^{\mathcal{LPT}}(m+k, I) \leq C_{\max}^{\mathcal{OPT}}(m, I)$ for $i = 1, \dots, n-1$. Otherwise, there exists a job i such that

$C_i^{\mathcal{LPT}}(m+k, I) > C_{\max}^{\mathcal{OPT}}(m, I)$ and $1 \leq i \leq n-1$. Create an instance I' from I by removing jobs $i+1$ through n from I . This does not affect the completion time of job i in the \mathcal{LPT} schedule because jobs $i+1$ through n are scheduled after job i is. Moreover, this cannot increase the optimal makespan. Thus, $C_{\max}^{\mathcal{LPT}}(m+k, I') \geq C_i^{\mathcal{LPT}}(m+k, I') = C_i^{\mathcal{LPT}}(m+k, I) > C_{\max}^{\mathcal{OPT}}(m, I) \geq C_{\max}^{\mathcal{OPT}}(m, I')$, and I' has fewer jobs than I . This contradicts our assumption that I is a smallest counterexample.

We next observe that \mathcal{OPT} cannot produce a schedule where some machine has only one job i . Suppose this is not the case. Create an instance I' from I by setting p_i to $C_{\max}^{\mathcal{OPT}}(m, I)$. This only possibly increases p_i . Clearly, $C_{\max}^{\mathcal{OPT}}(m, I') = C_{\max}^{\mathcal{OPT}}(m, I)$. Furthermore, $C_{\max}^{\mathcal{LPT}}(m+k, I') \geq C_{\max}^{\mathcal{LPT}}(m+k, I) > C_{\max}^{\mathcal{OPT}}(m, I) = C_{\max}^{\mathcal{OPT}}(m, I')$. Thus, I' is also a smallest counterexample. We can assume that, in the schedule created by \mathcal{LPT} for instance I' , job i is on a machine by itself. Create an instance I'' from I' by removing job i . Clearly, $C_{\max}^{\mathcal{OPT}}(m-1, I'') \leq C_{\max}^{\mathcal{OPT}}(m, I')$. Furthermore, $C_{\max}^{\mathcal{LPT}}(m-1+k, I'') = C_{\max}^{\mathcal{LPT}}(m+k, I') > C_{\max}^{\mathcal{OPT}}(m, I') \geq C_{\max}^{\mathcal{OPT}}(m-1, I'')$. Thus, I'' is a counterexample to the theorem statement where there are $m-1$ base machines and k extra machines. This contradicts the minimality of m .

We now observe that the schedule produced by \mathcal{LPT} just before job n arrives also cannot have any machines with only one job i . Suppose this is not the case. Assign job n to the machine with only job i . Since \mathcal{OPT} has no machines with only one job, job i must be scheduled with some other job j in the schedule produced by \mathcal{OPT} . Since job n is the smallest job in the input instance, $p_n \leq p_j$. This implies that $C_{\max}^{\mathcal{LPT}}(m+k, I) = C_n^{\mathcal{LPT}}(m+k, I) \leq p_i + p_n \leq p_i + p_j \leq C_{\max}^{\mathcal{OPT}}(m, I)$.

We now proceed with the remainder of the proof. There are 2 cases based on the size of p_n . In both cases, we show that $C_{\max}^{\mathcal{LPT}}(m+k, I) \leq C_{\max}^{\mathcal{OPT}}(m, I)$ which is a contradiction to the assumption that I is a counterexample.

Case 1: $p_n \leq \frac{1}{4}C_{\max}^{\mathcal{OPT}}$.

Job n is the last to finish in the \mathcal{LPT} schedule, and $p_n = p_{\min} \leq \frac{1}{4}C_{\max}^{\mathcal{OPT}}$. Thus, from Lemma 2.1,

$$\begin{aligned} C_{\max}^{\mathcal{LPT}} &\leq \left(\frac{m}{m+k} \right) C_{\max}^{\mathcal{OPT}} + \left(\frac{m+k-1}{m+k} \right) \frac{1}{4} C_{\max}^{\mathcal{OPT}} \\ &= \left(\frac{4m+m+k-1}{4m+4k} \right) C_{\max}^{\mathcal{OPT}} \\ &\leq C_{\max}^{\mathcal{OPT}} \quad \text{because } m \leq 3k+1. \end{aligned}$$

Case 2: $p_n > \frac{1}{4}C_{\max}^{\mathcal{OPT}}$.

Before we continue, we provide some definitions.

- Let ϕ be an optimal schedule on m machines for instance I .
- Let σ be the schedule produced by \mathcal{LPT} on $m+k$ machines for instance I .
- Let π be the partial schedule produced by \mathcal{LPT} on $m+k$ machines for instance I when exactly the first $n-1$ jobs are scheduled.
- Let H and K be the set of machines in schedule ϕ with exactly 2 and 3 jobs, respectively.
- Let E and F be the set of machines in schedule π with exactly 2 and 3 jobs, respectively.

Since for $i = 1, \dots, n$, $p_i \geq p_{\min} > \frac{1}{4}C_{\max}^{\phi}$, and $C_i^{\phi} \leq C_{\max}^{\phi}$, then there are at most 3 jobs per machine in schedule ϕ . Similarly, there are at most 3 jobs per machine in schedule π because $C_i^{\pi} \leq C_{\max}^{\phi}$ for $i = 1, \dots, n-1$. From earlier observations, there are at least two jobs per machine in both schedule ϕ and schedule π .

For the remainder of this proof, we introduce the following notation. When describing a machine, we will use a distinct letter such as u or v . When describing jobs, we will use two different notations. First, we will still call the last job scheduled by \mathcal{LPT} job n , and we will still refer to its length as p_n . However, we will also refer to jobs by how they are scheduled by the optimal algorithm. Specifically, if u is one of the m machines for the optimal schedule, u_i is the i 'th largest job scheduled on machine u with ties broken arbitrarily. Also, we will use $p(u_i)$ to denote the processing time of job u_i .

We now classify jobs according to

- (1) the number of jobs on the machine on which they are scheduled in ϕ and
- (2) the order of their size among jobs on the same machine in ϕ .

- Let $H_i = \{ u_i \mid u \in H \}$ for $i = 1, 2$.
- Let $H_{12} = H_1 \cup H_2$.
- Definitions of K with subscripts are analogous to the two definitions above.

We also classify jobs in a second manner with respect to the schedule π produced by \mathcal{LPT} .

- Let $E_i = \{ u_j \mid \text{job } u_j \text{ is the } i\text{th job on some machine } v \text{ in schedule } \pi \text{ and } v \in E \}$ for $i = 1, 2$.
- Let $E_{12} = E_1 \cup E_2$.
- Definitions of F with subscripts are analogous to the two definitions above.

Figure 1 illustrates the two job classification schemes described. To further clarify these definitions, consider the following statements which are implied by $u_i \in K_{23}$.

- (1) Either $u_i = u_2$ or $u_i = u_3$.
- (2) Job u_i is on machine u in schedule ϕ .
- (3) $u \in K$.
- (4) There are 3 jobs on machine u in schedule ϕ , namely u_1, u_2 , and u_3 .
- (5) $u_1 \in K_1, u_2 \in K_2, u_3 \in K_3$.
- (6) $p(u_1) \geq p(u_2) \geq p(u_3)$.
- (7) $p(u_1) + p(u_2) + p(u_3) \leq C_{\max}^\phi$.

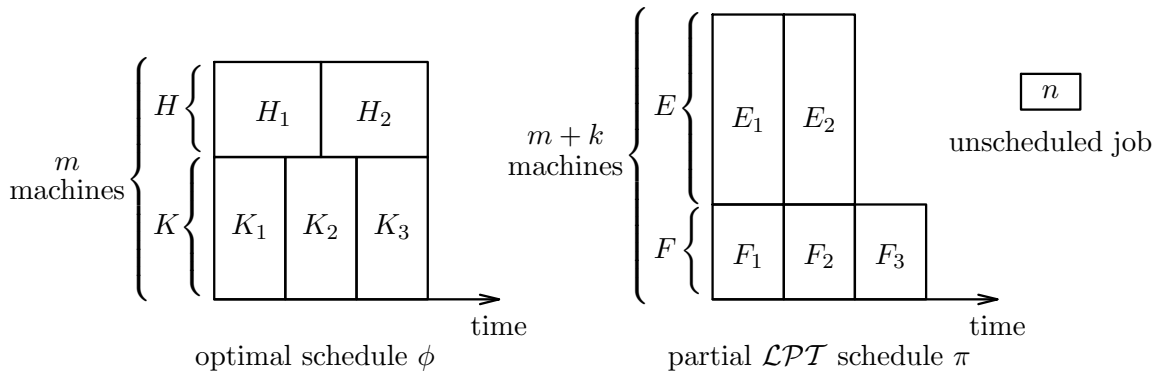


Figure 1: This figure illustrates the two job classification schemes used in the proof. Note, jobs in the same class may have different sizes.

From earlier arguments, there are either 2 or 3 jobs per machine in schedule ϕ and in schedule π . Therefore, we have the following equalities.

$$n = 2|E| + 3|F| + 1 \quad (1)$$

$$n = 2|H| + 3|K| \quad (2)$$

$$m = |H| + |K| \quad (3)$$

$$m + k = |E| + |F| \quad (4)$$

$$2|E| + 3|F| + 1 = 2|H| + 3|K| \quad \text{from (1) and (2)} \quad (5)$$

$$4(|H| + |K|) \leq 3(|E| + |F|) + 1 \quad \text{from (3),(4), and } m \leq 3k + 1 \quad (6)$$

We show that there exists a machine y in π and a machine z in ϕ such that if job n is scheduled on y , there is a pairing of jobs from the two machines such that the jobs from y are no larger than the jobs from z . We consider two cases.

Case 2.1: $E_1 \cap K_{23} \neq \emptyset$.

This case is shown pictorially in Figure 2. Relation $E_1 \cap K_{23} \neq \emptyset$ means that there exists a job v_i such that $v_i \in E_1$ and $v_i \in K_{23}$. Let job u_j be the job in E_2 on the same machine as v_i in schedule π . By definition of E_1 and E_2 , we have that $p(u_j) \leq p(v_i)$. In the optimal schedule, job v_i is scheduled on the same machine with exactly two other jobs, and job v_i is not the largest job on the machine. Thus, $p(u_j) \leq p(v_i) \leq p(v_2) \leq p(v_1)$. Finally, since job n is the smallest job, then $p_n \leq p(v_3)$. Combining these facts, it follows that job n can be scheduled on the same machine as v_i and u_j in the \mathcal{LPT} schedule so that the total load on the machine is no larger than the optimal makespan.

Algebraically, since $p(u_j) \leq p(v_i) \leq p(v_2) \leq p(v_1)$ and $p_n \leq p(v_3)$, then $C_{\max}^\sigma = C_n^\sigma \leq p(v_i) + p(u_j) + p_n \leq p(v_1) + p(v_2) + p(v_3) \leq C_{\max}^{\mathcal{OPT}}$.

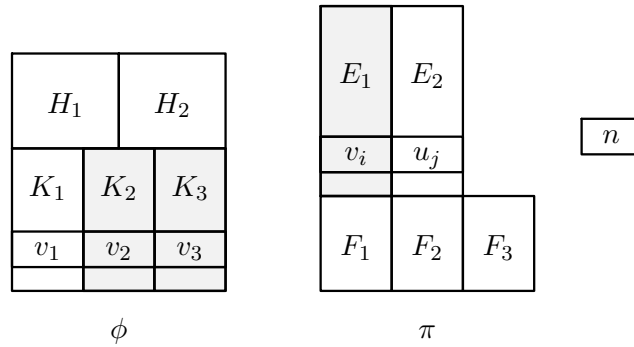


Figure 2: Case 2.1 of Theorem 3.2: $E_1 \cap K_{23} \neq \emptyset$.

Case 2.2: $E_1 \cap K_{23} = \emptyset$.

First, we establish some relations between schedule ϕ and schedule π .

$$E_1 \subseteq K_{23}^c = H_{12} \cup K_1 \quad \text{because } E_1 \cap K_{23} = \emptyset \quad (7)$$

$$2|H| + |K| \geq |E| \quad \text{from (7)} \quad (8)$$

$$2|H| + |K| \leq |E| \quad \text{from (5)+(6)} \quad (9)$$

$$E_1 = H_{12} \cup K_1 \quad \text{from (7), (8) and (9)} \quad (10)$$

$$E_2 \cup F_{123} \cup \{n\} = E_1^c = (H_{12} \cup K_1)^c = K_{23} \quad \text{from (10)} \quad (11)$$

Equalities (10) and (11) are the critical equalities we need and are shown graphically in Figure 3.

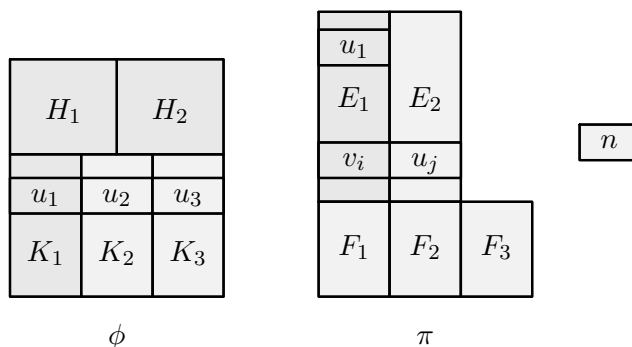


Figure 3: Case 2.2 of Theorem 3.2: $K_{23} = E_2 \cup F_{123} \cup \{n\}$ and $H_{12} \cup K_1 = E_1$.

Set E_2 is not empty because from equalities (10) and (3), $|E_2| = |E| = 2|H| + |K| \geq m \geq 1$. Let u_j be the first job in E_2 to be scheduled by \mathcal{LPT} . Suppose v_i is the job in E_1 which is on the same machine as u_j in schedule π . From relation (11), $u_j \in K_{23}$. Thus, there are 3 jobs on machine u in ϕ . If we can show that $p_n \leq p(u_3)$, $p(u_j) \leq p(u_2)$, and $p(v_i) \leq p(u_1)$, then $C_{\max}^\sigma = C_n^\sigma \leq p(v_i) + p(u_j) + p_n \leq p(u_1) + p(u_2) + p(u_3) \leq C_{\max}^{OPT}$. Obviously, $p_n \leq p(u_3)$ because job n is the smallest job. Since $u_j \in K_{23}$, then job u_j is either u_2 or u_3 . In any case, $p(u_j) \leq p(u_2)$.

We now prove $p(v_i) \leq p(u_1)$. By definition, job u_1 is in K_1 and u_1 is on the same machine as u_j in schedule ϕ . From relation (10), $u_1 \in E_1$. Since u_j is the first job to be scheduled in E_2 , and \mathcal{LPT} always schedules the next job on the least loaded machine, then the fact that u_j is scheduled on the same machine as v_i implies that v_i is a smallest job in E_1 . In particular, $p(v_i) \leq p(u_1)$, and this completes case 2.2.

Since this is the last possibility to consider, we have proven that there are no counterexamples to the theorem and thus the theorem is true. \square

The result of the previous theorem is tight in 2 aspects. First, with $\frac{m-1}{3}$ extra machines, \mathcal{LPT} does not always produce a schedule with makespan *strictly smaller* than that of the optimal algorithm. In fact, no algorithm can guarantee this due to the fact that an input instance can contain a job whose length is equal to the optimal makespan. Second, as will be shown in the next theorem, if \mathcal{LPT} has fewer than $\frac{m-1}{3}$ extra machines, there exist instances such that \mathcal{LPT} will produce a schedule with the makespan *strictly larger* than that of the optimal algorithm.

Theorem 3.3. *For any pair of non-negative integers m and k such that $k = \frac{m-2}{3}$, there exist instances such that $C_{\max}^{\mathcal{LPT}}(m+k) \geq \frac{8k+7}{8k+6} C_{\max}^{OPT}(m) > C_{\max}^{OPT}(m)$.*

Proof. For $k \geq 0$, the instance I_k with $8k + 5$ jobs is defined as follow:

$$\begin{aligned} & 2k + 2 \text{ jobs of size } 4k + 3 \\ & \quad 2 \text{ jobs of size } 2k + 2 + i \text{ for } i = 2k, \dots, 1 \text{ (a total of } 4k \text{ jobs)} \\ & 2k + 3 \text{ jobs of size } 2k + 2 \end{aligned}$$

Figure 4 illustrates a lower bound instance, its optimal schedule on m machines, and its \mathcal{LPT} schedule on $m + k$ machines. The optimal schedule on $m = 3k + 2$ machines can be described as follows:

$$\begin{aligned} & k + 1 \text{ machines with jobs of sizes } 4k + 3 \quad 4k + 3 \\ & \quad 2 \text{ machines with jobs of sizes } 4k + 2 \quad 2k + 2 \quad 2k + 2 \\ & \quad 2 \text{ machines with jobs of sizes } 4k + 2 - i \quad 2k + 2 + i \quad 2k + 2 \text{ for } i = 1, \dots, k - 1 \\ & \hspace{15em} \text{(a total of } 2k - 2 \text{ machines)} \\ & \quad 1 \text{ machine with jobs of sizes } 3k + 2 \quad 3k + 2 \quad 2k + 2 \end{aligned}$$

All machines in the optimal schedule have load $8k + 6$. The schedule produced by \mathcal{LPT} on $m + k = 4k + 2$ machines can be described as follows:

$$\begin{aligned} & \quad 1 \text{ machine with jobs of sizes } 4k + 3 \quad 2k + 2 \quad 2k + 2 \\ & 2k + 1 \text{ machines with jobs of sizes } 4k + 3 \quad 2k + 2 \\ & \quad 2 \text{ machines with jobs of sizes } 4k + 3 - i \quad 2k + 2 + i \quad \text{for } i = 1, \dots, k \\ & \hspace{15em} \text{(a total of } 2k \text{ machines)} \end{aligned}$$

The machine with 3 jobs in the \mathcal{LPT} schedule have load $8k + 7$. All other machines have load $6k + 5$. Note that \mathcal{LPT} could schedule the last job (of size $2k + 2$) on any machine. \square

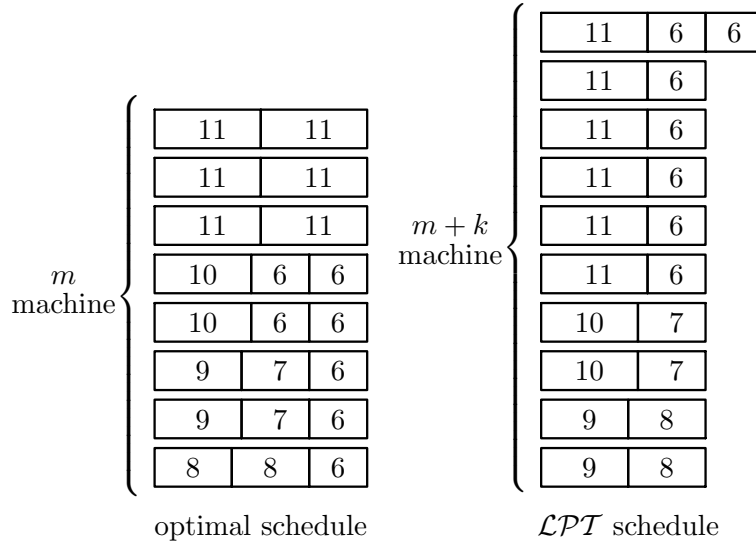


Figure 4: A lower bound instance when $k = 2$ and $m = 3k + 2 = 8$. Each job is labeled with its size.

4 Lower Bounds on \mathcal{LPT}

In this section, we describe a procedure to compute a good, though not necessarily optimal, lower bound of the performance of \mathcal{LPT} for any m and k . Given m and k , we use a linear program that assumes that the optimal schedule and the \mathcal{LPT} schedule have the specific structures shown in Figure 5 where jobs are ordered by non-increasing size. The variables of the linear program are the job sizes and the starting time of the last job in the \mathcal{LPT} schedule. This procedure considers only a very restricted set of possible instances. The best lower bound instance from this restricted set can be obtained by optimizing the linear program.

We describe our assumptions in detail next. We assume that in the \mathcal{LPT} schedule, there are 2 jobs on all machines except for one machine on which there are 3 jobs. Suppose $p_1 \leq p_2 \leq \dots \leq p_n$. We assume that in the \mathcal{LPT} schedule, job i is paired with job $n - i$, and job n is the last job to finish. We assume that the optimal makespan is no larger than 1. Finally, we assume that there are either 2 or 3 jobs on each machine in the optimal schedule, and the optimal schedule has a structure as illustrated in Figure 5.

The linear program can be described algebraically as follows. Let m_2 and m_3 be the number of machines in the optimal schedule with 2 and 3 jobs respectively. From the above assumptions,

$$\begin{aligned}
 n &= 2(m + k) + 1, \\
 n &= 2m_2 + 3m_3, & \text{and} \\
 m &= m_2 + m_3. \\
 \text{Thus, } m_2 &= m - 2k - 1 & \text{and} \\
 m_3 &= 2k + 1.
 \end{aligned}$$

We denote the starting time of job n in the \mathcal{LPT} schedule by s . Given $m \geq 1$ and $0 \leq k \leq (m - 1)/2$, we construct the linear program $LB(m, k)$ as follows.

$$\begin{array}{ll}
 \text{maximize } s + p_n & \text{subject to} \\
 p_i \geq p_{i+1}, & i = 1, \dots, n - 1 \\
 s \leq p_i + p_{n-i}, & i = 1, \dots, (n - 1)/2 \\
 p_i + p_j \leq 1, & \forall (i, j) \in H \\
 p_i + p_j + p_l \leq 1, & \forall (i, j, l) \in K
 \end{array}$$

where

$$\begin{aligned}
 H &= \{ (i, 2m_2 + 1 - i) \mid i = 1, \dots, m_2 \} \\
 K &= \{ (2m_2 + i, 2m_2 + 2m_3 + 2 - i, 2m_2 + 2m_3 + 1 + i) \mid i = 1, \dots, m_3 - 1 \} \cup \\
 &\quad \{ (2m_2 + m_3, 2m_2 + m_3 + 1, 2m_2 + m_3 + 2) \}
 \end{aligned}$$

Notice the difference between the layout of jobs in the optimal schedules in Figures 4 and 5. In Theorem 3.3, we chose to use the layout in Figure 4 rather than the layout in Figure 5 because it is easier to describe. Table 1 shows the computed lower bounds for $m = 1, \dots, 50$ and $k = 0, \dots, 12$. Figure 6 shows the plot of this table.

Figure 7 shows the plot of $LB(m, k)$ when $m = 100$. Points in Figure 7 are the lower bounds obtained from our linear program when $m = 100$. The upper curve is the performance guarantee

of \mathcal{LPT} from Theorem 3.1. It is important to note that the lower bounds found when $m \geq 1$ and $k = 0$ or $k \geq \frac{m-1}{3}$ match the upper bounds given by Graham [15] and Theorem 3.2, respectively. The lower curve is the performance guarantee of \mathcal{LPT} when it is given m machines with speed $\frac{m+k}{m}$ instead of $m+k$ machines with unit speed. When an algorithm is given machines with speed s , it can guarantee a makespan which is $\frac{1}{s}$ times the guaranteed makespan using machines with unit speed. Intuitively, faster machines should be utilized more efficiently than extra machines. In Figure 7, the lower bounds obtained from the linear program mostly lie well above the lower curve. However, there is one point which lies below the lower curve. It is possible that this point is a counterexample to our result, but more likely we have not found an optimal extra machine lower bound for this particular combination of m and k .

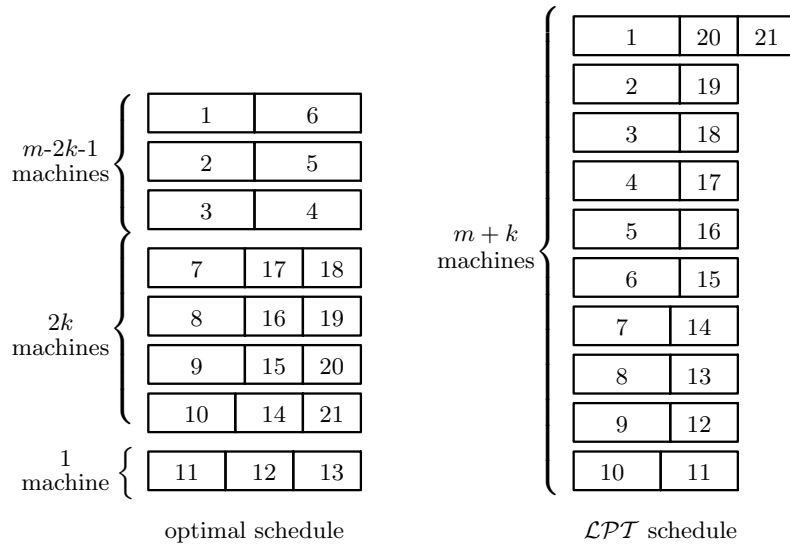


Figure 5: A lower bound instance when $m = 8$ and $k = 2$. Each job is labeled with its job id.

5 Future Work

Extra-resource analysis is a promising analysis technique which can be used to derive greater insight into the behavior of both on-line and off-line algorithms. Previously, these insights have been used to identify good on-line algorithms in settings where traditional competitive analysis fails to do so. In this work, we show that these insights can also be used to derive a divergence between off-line and on-line load balancing algorithms. We believe that future work should, in part, search for even more applications of the extra-resource analysis technique.

6 Acknowledgement

The authors would like to thank Kirk Pruhs for pointing out several studies involving extra-resource analysis.

$m \setminus k$	0	1	2	3	4	5	6	7	8	9	10	11	12
1	1	-	-	-	-	-	-	-	-	-	-	-	-
2	7/6	-	-	-	-	-	-	-	-	-	-	-	-
3	11/9	1	-	-	-	-	-	-	-	-	-	-	-
4	5/4	1	-	-	-	-	-	-	-	-	-	-	-
5	19/15	15/14	1	-	-	-	-	-	-	-	-	-	-
6	23/18	10/9	1	-	-	-	-	-	-	-	-	-	-
7	9/7	7/6	1	1	-	-	-	-	-	-	-	-	-
8	31/24	7/6	23/22	1	-	-	-	-	-	-	-	-	-
9	35/27	27/23	15/14	1	1	-	-	-	-	-	-	-	-
10	13/10	25/21	11/10	1	1	-	-	-	-	-	-	-	-
11	43/33	11/9	10/9	31/30	1	1	-	-	-	-	-	-	-
12	47/36	11/9	7/6	19/18	1	1	-	-	-	-	-	-	-
13	17/13	11/9	7/6	15/14	1	1	1	-	-	-	-	-	-
14	55/42	70/57	7/6	11/10	39/38	1	1	-	-	-	-	-	-
15	59/45	5/4	7/6	53/48	23/22	1	1	1	-	-	-	-	-
16	21/16	5/4	27/23	10/9	19/18	1	1	1	-	-	-	-	-
17	67/51	5/4	19/16	7/6	15/14	47/46	1	1	1	-	-	-	-
18	71/54	5/4	25/21	7/6	11/10	31/30	1	1	1	-	-	-	-
19	25/19	19/15	11/9	7/6	11/10	19/18	1	1	1	1	-	-	-
20	79/60	19/15	11/9	7/6	10/9	19/18	55/54	1	1	1	-	-	-
21	83/63	19/15	11/9	7/6	10/9	15/14	31/30	1	1	1	1	-	-
22	29/22	19/15	11/9	7/6	7/6	11/10	23/22	1	1	1	1	-	-
23	91/69	23/18	11/9	27/23	7/6	11/10	19/18	63/62	1	1	1	1	-
24	95/72	23/18	27/22	19/16	7/6	76/69	19/18	39/38	1	1	1	1	-
25	33/25	23/18	70/57	19/16	7/6	10/9	15/14	31/30	1	1	1	1	1
26	103/78	23/18	5/4	25/21	7/6	10/9	11/10	19/18	71/70	1	1	1	1
27	107/81	9/7	5/4	11/9	7/6	7/6	11/10	19/18	43/42	1	1	1	1
28	37/28	9/7	5/4	11/9	7/6	7/6	11/10	19/18	31/30	1	1	1	1
29	115/87	9/7	5/4	11/9	7/6	7/6	53/48	15/14	23/22	79/78	1	1	1
30	119/90	9/7	5/4	11/9	27/23	7/6	10/9	11/10	19/18	47/46	1	1	1
31	41/31	31/24	5/4	11/9	19/16	7/6	10/9	11/10	19/18	31/30	1	1	1
32	127/96	31/24	5/4	11/9	19/16	7/6	7/6	11/10	19/18	31/30	87/86	1	1
33	131/99	31/24	19/15	11/9	25/21	7/6	7/6	11/10	15/14	19/18	55/54	1	1
34	45/34	31/24	19/15	27/22	25/21	7/6	7/6	10/9	11/10	19/18	39/38	1	1
35	139/105	35/27	19/15	27/22	11/9	7/6	7/6	10/9	11/10	19/18	31/30	95/94	1
36	143/108	35/27	19/15	70/57	11/9	7/6	7/6	10/9	11/10	19/18	23/22	55/54	1
37	49/37	35/27	19/15	5/4	11/9	27/23	7/6	7/6	11/10	15/14	19/18	43/42	1
38	151/114	35/27	19/15	5/4	11/9	19/16	7/6	7/6	53/48	11/10	19/18	31/30	103/102
39	155/117	13/10	19/15	5/4	11/9	19/16	7/6	7/6	10/9	11/10	19/18	31/30	63/62
40	53/40	13/10	23/18	5/4	11/9	19/16	7/6	7/6	10/9	11/10	19/18	19/18	43/42
41	163/123	13/10	23/18	5/4	11/9	25/21	7/6	7/6	10/9	11/10	15/14	19/18	31/30
42	167/126	13/10	23/18	5/4	11/9	25/21	7/6	7/6	7/6	11/10	11/10	19/18	31/30
43	57/43	43/33	23/18	5/4	11/9	11/9	7/6	7/6	7/6	53/48	11/10	19/18	23/22
44	175/132	43/33	23/18	5/4	27/22	11/9	27/23	7/6	7/6	10/9	11/10	19/18	19/18
45	179/135	43/33	23/18	5/4	27/22	11/9	19/16	7/6	7/6	10/9	11/10	15/14	19/18
46	61/46	43/33	23/18	5/4	70/57	11/9	19/16	7/6	7/6	10/9	11/10	11/10	19/18
47	187/141	47/36	9/7	19/15	70/57	11/9	19/16	7/6	7/6	7/6	76/69	11/10	19/18
48	191/144	47/36	9/7	19/15	5/4	11/9	19/16	7/6	7/6	7/6	10/9	11/10	19/18
49	65/49	47/36	9/7	19/15	5/4	11/9	25/21	7/6	7/6	7/6	10/9	11/10	15/14
50	199/150	47/36	9/7	19/15	5/4	11/9	25/21	7/6	7/6	7/6	10/9	11/10	11/10

Table 1: The lower bounds computed by our linear program.

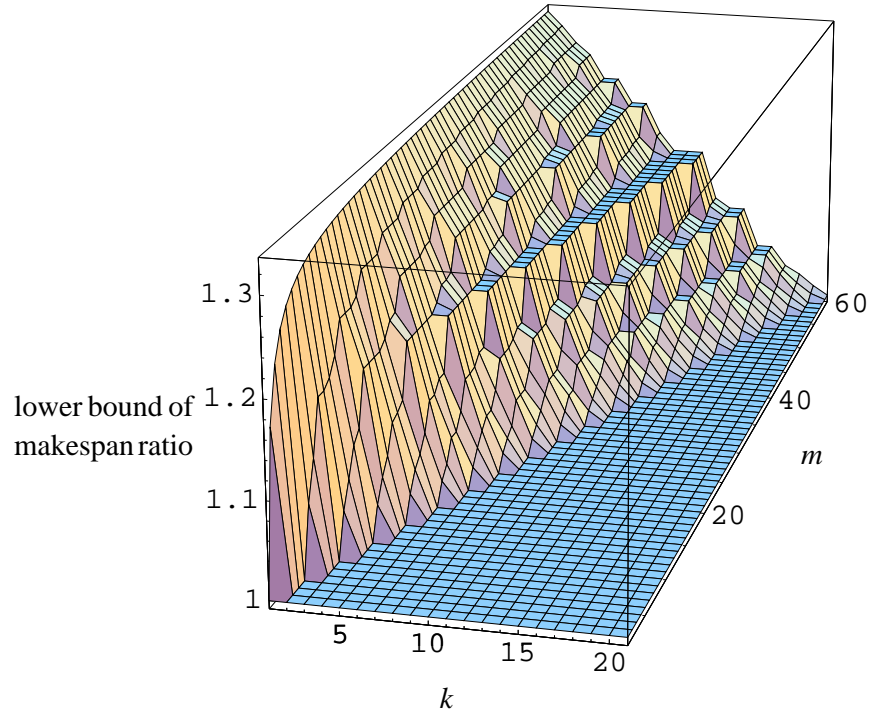


Figure 6: The plot of the lower bounds computed by our linear program for $m = 1, \dots, 60$ and $k = 0, \dots, 20$.

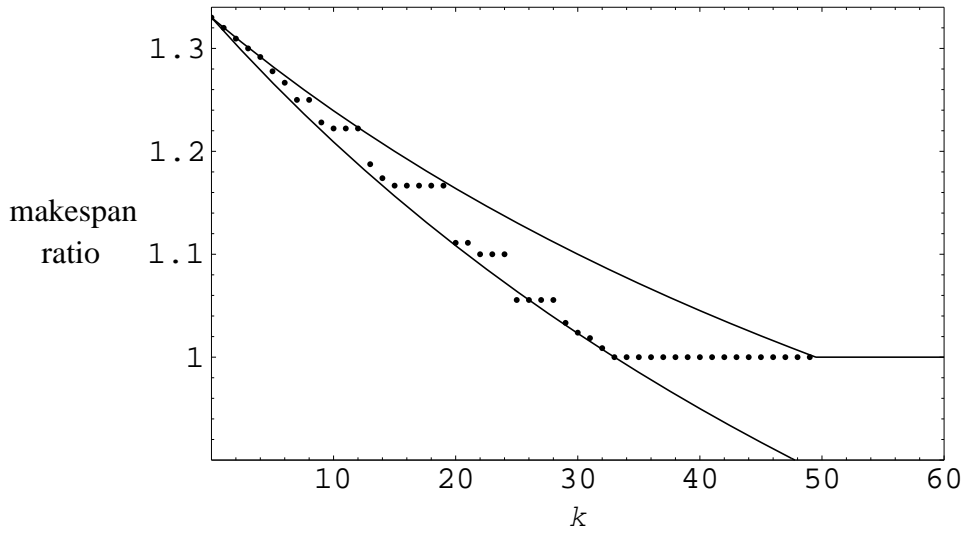


Figure 7: This figure summarizes upper and lower bounds for \mathcal{LPT} when $m = 100$. Points in the figure are the lower bounds computed by our linear program. The upper curve is the performance guarantee of \mathcal{LPT} from Theorem 3.1. The upper curve is described by $y = \max\{\frac{4m+k-1}{3m+3k}, 1\}$. The lower curve is the performance guarantee of \mathcal{LPT} when it is given m machines with speed $\frac{m+k}{m}$ instead of $m+k$ machines with unit speed. The lower curve is described by $y = \frac{4m-1}{3m} \frac{m}{m+k} = \frac{4m-1}{3m+3k}$.

References

- [1] S. Albers. Better bounds for online scheduling. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 130–139, 1997.
- [2] S. Albers, S. Arora, and S. Khanna. Page replacement for generalized caching problems. In *Proceedings of the 10th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 31–40, 1999.
- [3] Y. Azar and O. Regev. On-line bin-stretching. In *Proceedings of the 2nd RANDOM*, pages 71–81, 1998.
- [4] Yossi Azar, Leah Epstein, and Rob van Stee. Resource augmentation in load balancing, 1999. manuscript, 14 pages.
- [5] Piotr Berman and Chris Coulston. Speed is more powerful than clairvoyance. *Nordic Journal of Computing*, pages 181–193, 1999.
- [6] Allan Borodin, Sandy Irani, Prabhakar Raghavan, and Baruch Schieber. Competitive paging with locality of reference. *Journal of Computer and System Sciences*, 50(2):244–258, 1995.
- [7] Joan Boyar and Kim S. Larsen. The seat reservation problem. Technical report, University of Southern Denmark, 1996. PP-1996-24, September 1996, 12 pages.
- [8] Joan Boyar, Kim S. Larsen, and Morten N. Nielsen. The accommodating function – a generalization of the competitive ratio. Technical report, University of Southern Denmark, 1998. PP-1998-24, December 22, 1998, 30 pages.
- [9] E. Coffman, M. Garey, and D. Johnson. Approximation algorithms for bin packing: A survey. In D. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, chapter 2, pages 46–93. PWS Publishing Company, 1997.
- [10] Paolo Dell’Olmo, Hans Kellerer, Maria Grazia Speranza, and Zsolt Tuza. A 13/12 approximation algorithm for bin packing with extendable bins. *Information Processing Letters*, 65:229–233, 1998.
- [11] Jeff Edmonds. Scheduling in the dark. In *Proceedings of the 31st ACM Symposium on Theory of Computing*, pages 179–188, 1999.
- [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] T. Gormley, N. Reingold, E. Torng, and J. Westbrook. Generating adversaries for request-answer games. In *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms*, pages 564–565, 2000.
- [14] R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.
- [15] R.L. Graham. Bounds on multiprocessing anomalies. *SIAM Journal on Applied Mathematics*, 17:263–269, 1969.
- [16] D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems: practical and theoretical results. *Journal of the ACM*, 34(1):144–162, 1987.

- [17] D.S. Johnson, A. Demers, J.D. Ullman, M.R. Garey, and R.L. Graham. Worst case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3:299–325, 1974.
- [18] B. Kalyanasundaram and K. Pruhs. The online transportation problem. In *Lecture Notes in Computer Science 979*, pages 484–493, 1995. European Symposium on Algorithms (ESA). Also to appear in *SIAM Journal on Discrete Mathematics*.
- [19] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. In *Proceedings of the 36th Annual IEEE Foundations of Computer Science*, pages 214–221, 1995. Also to appear in *Journal of the ACM*.
- [20] Bala Kalyanasundaram and Kirk Pruhs. Maximizing job completions online. In *Lecture Notes in Computer Science 1461*, pages 235–246, 1998. European Symposium on Algorithms (ESA).
- [21] R.M. Karp and N. Karmarkar. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proc. of the 23th IEEE Symp. Foundations of Computer Science*, pages 312–320, 1982.
- [22] Madhukar R. Korupolu, C. Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, pages 1–10, 1998.
- [23] E. Koutsoupias and C. Papadimitriou. Beyond competitive analysis. In *Proceedings of the 35th Annual IEEE Foundations of Computer Science*, pages 394–400, 1994.
- [24] Tak Wah Lam and Kar Keung To. Trade-offs between speed and processor in hard-deadline scheduling. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 623–632, 1999.
- [25] Cynthia A. Phillips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 140–149, 1997.
- [26] M.B. Richey. Improved bounds for harmonic-based bin packing algorithms. *Discrete Applied Mathematics*, 34:203–227, 1991.
- [27] M.G. Speranza and Z. Tuza. On-line approximation algorithms for scheduling tasks on identical machines with extendable working time. *Annals of Operations Research*, 86:491–506, 1999.