Approximation Schemes for Scheduling

Noga Alon* Yossi Azar† Gerhard J. Woeginger‡ Tal Yadid§

Abstract

We consider the classic scheduling/load balancing problems where there are m identical machines and n jobs, and each job should be assigned to some machine. Traditionally, the assignment of jobs to machines is measured by the makespan (maximum load) i.e., the L_∞ norm of the assignment. An ϵ-approximation scheme was given by Hochbaum and Shmoys [10] for minimizing the L_∞ norm.

In several applications, such as in storage allocation, a more appropriate measure is the sum of the squares of the loads (which is equivalent to the L_2 norm). This problem was considered in [4, 5, 13] who showed how to approximate the optimum value by a factor of about 1.04. In fact, a more general measure, which is the L_p norm (for any $p \geq 1$) can also be approximated to some constant (see Chandra and Wong [4]) which may be as large as 3/2. We improve these results by providing an ϵ-approximation scheme for the general L_p norm (and in particular for the L_2 norm). We also consider the case of restricted assignment of unit jobs where we show how to find in polynomial time, a solution which is optimal for all norms.

1 Introduction

We consider the classic scheduling/load balancing problems. These are some of the most well-studied problems in scheduling theory (see e.g., [9]). For these problems, there are m identical machines and n jobs with weights $w_i \geq 0$ where each job should be assigned to some machine. The assignment results in a machines load vector where its j'th coordinate is the sum of weights of jobs assigned to the j'th machine.

In relation to storage allocation problems Chandra and Wong [4] studied the problem of minimizing the sum of the squares of the machines load vector. Cody and Coffman [5], in their study of placing a set of records on a sectored drum to minimize the average latency, were confronted with essentially the same minimization problem.

The above minimization problem is known to be NP-hard [6]. Chandra and Wong [4] considered heuristics to provide approximation algorithms. They showed that sorting the weights (in non-increasing order) and using the LPT rule due to Graham [8] (i.e. assigning a job to the current least loaded machine) results in a worst case performance bound of 25/24 for minimizing the sum of the squares. This result was slightly improved by Leung and Wei [13]. No better approximations have been known prior to our paper.

Note that minimizing the sum of the squares is equivalent to minimizing the L_2 norm of the machines load vector. In fact, Chandra and Wong [4] also considered the the general L_p norm (for any $p \geq 1$). They showed that the LPT rule on the sorted items achieves a performance bound of a constant. The constant depends on p and may be as large as 3/2. The case $p = \infty$ (i.e., L_∞) is the classic ancient problem of minimizing the makespan (or maximum load). Graham [7] showed that the LPT rule achieves a performance bound of $2 - 1/m$ for minimizing the makespan. Later [8] he showed that sorting the weights (in non-increasing order) and using the LPT rule results in a better bound of $4/3 - 1/(3m)$. It took some time until Hochbaum and Shmoys [10] designed an ϵ-approximation scheme for the L_∞ norm. That is for any $\epsilon > 0$ there is a polynomial time algorithm A_ϵ that approximates the optimal solution up to a factor of $1 + \epsilon$. The running time of the algorithm depends polynomially on n but exponentially on $1/\epsilon$. In fact, since the problem is strongly NP hard no fully polynomial approximation scheme exists unless P=NP.

In this paper we resolve the general problem by providing an ϵ-approximation scheme for scheduling jobs with respect to the L_p norm for any $p \geq 1$ (in particular for the L_2 norm). Of course, the running time of the algorithm depends polynomially on n but exponentially on $1/\epsilon$. We also improve on the complexity of [10] by providing an algorithm whose running time depends on $1/\epsilon$ only in the constant (the dependence on n is linear).

In some applications, both the L_∞ and the L_2

*School of Mathematical Sciences, Tel-Aviv University. Supported in part by a USA-Israeli BSF grant. E-Mail: noga@math.tau.ac.il

†Dept. of Computer Science, Tel-Aviv University. Supported by Alon Fellowship and by the Israel Science Foundation, administered by the Israel Academy of Sciences. E-Mail: azar@math.tau.ac.il

‡TU Graz, Institut für Mathematik, Graz, Austria. E-Mail: gwoegi@igi.tu-graz.ac.at

§Dept. of Computer Science, Tel-Aviv University. E-Mail: yadid@math.tau.ac.il

norms are suitable ways to measure how well the jobs are balanced. For example consider a case in which a job describes a process whose weight corresponds to its frequency access to a disk (a machine). Then each access request may see a delay that is proportional to the load on the machine it is assigned to. Thus the *average* delay is proportional to the sum of the squares of the machines loads, where *maximum* delay is proportional to the maximum load.

It is easy to come with an example of two vectors in which one has a better L_2 norm and the other has a better L_∞ norm. Still one might be tempted to think that if the vectors correspond to optimal assignments then there is an assignment that is optimal in all norms. In fact, this is true for the two machines case. Moreover, this is true also for the case of restricted assignment of unit jobs as will be discussed later. In general, however, this is incorrect. In the appendix we give an example in which the optimal assignment in the L_∞ norm is different than the optimal assignment for the L_2 norm. It also follows that an ϵ-approximation scheme for the makespan (L_∞ norm) does not provide such a scheme for the L_2 (or L_p) norm.

Our ϵ-approximation scheme for scheduling jobs with respect to the L_p norm generalizes the algorithm of Hochbaum and Shmoys [10] and builds on some of their ideas. However, there are several differences where a major one is dealing with "small" jobs. The structure of the algorithm in [10] is first to remove the small jobs, then solve the remaining problem, and at last add greedily the small jobs. One may suggest that a similar approach may be used to approximate the L_p norm. It turns out that if we remove the small jobs and then get the exact (not just an approximate) solution, it may be the case that there is no way to add the small jobs and get even close to the required approximate factor. More specifically, in the appendix we show that this method cannot approximate the optimal solution better than some fixed ϵ. Therefore a new way is required to deal with the small jobs.

We also consider the case of unit weights jobs, where each is associated with a subset of the machines and should be assigned to one of them [3]. This case is called restricted assignment of unit jobs. Recall that, in general, an optimal assignment with respect to one norm may be non-optimal with respect to other norms. Surprisingly, we show that for restricted assignment of unit jobs there exists an assignment that is optimal in all norms. We call it a strongly-optimal assignment since it has the nice advantage of being optimal in all respects. Moreover, we show that such an assignment can be found in polynomial time. Lenstra-Shmoys-Tardos [12] considered the restricted assignment of unit jobs only for the L_∞ norm. They obtained an optimal solution by using network flow algorithms. We find an assignment that is optimal for all norms using "special" augmenting paths.

On-line scheduling/load balancing with respect to the L_2 (or L_p) norm has been considered in [1, 2]. Here the jobs arrive one by one and should be assigned to a machine based only on the previous jobs without any knowledge on the future jobs. It is not hard to to see that the LPT rule achieves an approximation factor (competitive ratio) of 2 for any L_p norm. For the L_2 norm the competitive ratio of the LPT rule is in fact $\sqrt{(4/3)}$ [2]. For the restricted assignment in the L_p norm a $\Theta(p)$ competitive ratio and a matching lower bound are shown in [1].

The paper is structured as follows. In section 2 we give definitions, notations and some easy observations for the classic scheduling problem. In section 3 we show how to get an optimal solution for the case where the numbers of possible job weights is constant. In section 4 we deal with the ϵ-approximation scheme for the general case. In section 5 we consider the case of unit jobs on restricted assignment. In the appendix we show examples that were discussed earlier.

2 Definitions and Notations

In the identical machines scheduling and load balancing problems, we are given m identical machines (servers) and a set of n jobs with non negative weights $w_1, \ldots, w_n$. Each job should be assigned to a machine. For a given assignment, the load l_i on a machine i is the sum of the weights of the jobs assigned to it. We denote by $\vec{l} = (l_1, \ldots, l_m)$ the machines load vector. The quality of the assignment is measured by the L_p norm, i.e. $|l|_p = (\sum_{i=1}^m l_i^p)^{\frac{1}{p}}$. We Denote the average load on the machines by $L = \frac{1}{m}\sum_{j=1}^n w_j$. We start with few simple observations, that allow us to assign very large jobs appropriately.

Claim 2.1. *If for a given assignment $l_{i_1} - w_j > l_{i_2}$ where job j is assigned to machine i_1 then assigning j to i_2 reduces the L_p norm of the load vector.*

Proof. Consider the load vector $\vec{x}$ generated by assigning j to i_2. Obviously, $x_i = l_i$ for each machine except for i_1 and i_2. Clearly $l_{i_2} < x_{i_2} < l_{i_1}$, $l_{i_2} < x_{i_1} < l_{i_1}$ and $x_{i_1} + x_{i_2} = l_{i_1} + l_{i_2}$. It follows from the convexity of $f(t) = t^p$ that $x_{i_1}^p + x_{i_2}^p < l_{i_1}^p + l_{i_2}^p$. Hence $|\vec{x}|_p < |\vec{l}|_p$.

The following claim implies that large jobs should be assigned to separate machines.

Claim 2.2. *If a job j_1 has weight $w_{j_1} > L$, then in any optimal solution no other job is assigned to the same machine that j_1 is assigned to.*

Proof. Consider an optimal assignment with load vector $\vec{l}$. Suppose that job j_1 is assigned to machine i_1 and another job, $j_2 \neq j_1$, is assigned to i_1 as well. Let i_2 be a machine such that $l_{i_2} < L$. Such a machine must exist since L is the average load on the machines. Observe that $l_{i_1} - l_{i_2} > w_{j_1} + w_{j_2} - L > w_{j_2}$. Claim 2.1 implies that assigning j_2 to machine i_2 yields a better assignment, which is a contradiction.

For assigning the large jobs, we use the above claim iteratively. While there is a job whose weight is larger than the current average load, we assign the job to a machine and remove both from the setting.

Note that both the optimal algorithm and the iterative process described above assign the large jobs to separate machines. Thus, we may assume that no job has weight larger than the average. More specifically, given an ϵ-approximation algorithm A_ϵ for the case where none of the jobs has weight larger than the average, we can generate an approximation algorithm for the general case, by assigning the large jobs to separate machines, and applying A_ϵ on the remaining jobs and machines. Obviously the resulting algorithm assures ϵ-approximation for the general case. Since no job has weight larger than the average load, we can bound the maximum load on the machines in an optimal solution as follows:

CLAIM 2.3. *In an optimal solution no machine has load greater than $2L$.*

Proof. Assume that for the optimal solution there is a machine i_1 with load $l_{i_1} > 2L$. There must be a machine i_2 whose load is smaller than the average load L. Let j be a job assigned to machine i_1. It follows that $l_{i_1} - l_{i_2} > L \geq w_j$. By Claim 2.1, $\vec{l}$ is not optimal, which is a contradiction.

3 Scheduling with a constant number of weights

We start by considering the case where all the weights belong to some constant size set $C = \{y_1, y_2, \ldots, y_r\}$ such that $L \geq y_1 > y_2 > \ldots > y_r \geq \alpha L$. We use the following notation to represent the input and the assignments. The input jobs are represented as a vector $\vec{n} = (n_1, \ldots, n_r)$, where n_k denotes the number of jobs whose weight is y_k. Notice that $n = \sum_{k=1}^{r} n_k$. An assignment to a machine is a vector $\vec{u} = (u_1, \ldots, u_r)$, where u_k is the number of jobs of weight y_k assigned to that machine. The load of assignment $\vec{u}$, denoted $W(\vec{u})$ is $\sum_{k=1}^{r} u_k \cdot y_k$. Denote by E the set of all possible assignments $\vec{u}$. It follows from Claim 2.3 that we need to consider only vectors $\vec{e} \leq \vec{n}$ (i.e $\leq$ in each coordinate) with $W(\vec{e}) \leq 2L$. Hence, each $\vec{e} \in E$ consists of at most $2/\alpha$ jobs. Therefore, for $r > 1$, $|E| < r^{\frac{2}{\alpha}+1}$ and can be found is time $O(|E|)$.

We show two alternative methods to find an optimal solution in polynomial time. The first is by procedure $DYNPROG$ which is based on a simple dynamic programming. The second is more efficient but more complex algorithm ILP which is based on integer linear programming in a fixed dimension [11].

First we describe $DYNPROG$, the algorithm for solving the scheduling problem given the vector of jobs $\vec{n}$ and m machines using dynamic programming. Let V be the set of vectors corresponding to subsets of the input, i.e. $V = \{\vec{v} : \vec{0} \leq \vec{v} \leq \vec{n}\}$. We build a layered-graph G, with $m+1$ layers. Layer 0 has a single node, $(0, \vec{0})$. The next $m-1$ layers consist of the nodes $\{(i, \vec{v}), 1 \leq i \leq m-1, \vec{v} \in V\}$, where i indicates the layer, and $\vec{v}$ is a vector of jobs. Layer m has, again, a single node, $(m, \vec{n})$. We associate a value with each node, denoted $Val(i, \vec{v})$, which corresponds to the p power of the load of a best assignment of $\vec{v}$ to the first i machines. More specifically, if $(l_1, \ldots, l_i)$ is a load vector that minimizes the L_p norm for assigning $\vec{v}$ on i machines, then $Val(i, \vec{v}) = \sum_{k=1}^{i} (l_k)^p$.

It is easy to see that the optimal assignment of v to i machines, satisfies the following recurrence:

$$\begin{aligned} Val(0, \vec{0}) &= 0 \\ Val(1, \vec{v}) &= W(\vec{v})^p \qquad \forall \vec{v} \in V \end{aligned}$$

and for $i \geq 2$

$$Val(i, \vec{v}) = \min_{\vec{v} \geq \vec{e} \in E} \{W(\vec{e})^p + Val(i-1, \vec{v} - \vec{e})\}$$

Notice that we can find the best assignment, not only its value, by keeping for each node $(i, \vec{v})$ an incoming edge $e \in E$ such that $Val(i, \vec{v}) = W(\vec{e})^p + Val(i-1, \vec{v} - \vec{e})$.

For computing the value of a node, we look at no more than $|E|$ possible assignments. For each assignment we do a constant amount of work. Thus, the total running time of the dynamic programming is $O(m \cdot |V||E|)$.

In order to upper bound $|V|$ we note that $|V| = \prod_{k=1}^{r} (n_k + 1)$. Under the constraint $\sum_{k=1}^{r} n_k = n$, this product is upper bounded by $(\frac{n}{r} + 1)^r$ by the Arithmetic Geometric Means inequality. Hence, $|V| = O((\frac{2n}{r})^r)$. Recall that $|E| \leq r^{\frac{2}{\alpha}+1}$. Therefore the running time of the algorithm is $O(m \cdot (\frac{2n}{r})^r \cdot r^{\frac{2}{\alpha}+1})$.

Thus we proved the following theorem:

THEOREM 3.1. *Given m machines and n jobs with weights taken from the set $\{y_1, \ldots, y_r\}$, and $\min_i\{y_i\} \geq \alpha L$, where α and r are constants independent of the* *input, DYNPROG finds a best assignment in terms of the L_p norm, in time $O(m \cdot (\frac{2n}{r})^r \cdot r^{\frac{2}{\alpha}+1})$.*

The second method to solve the constant number of weight scheduling problem is by using integer linear programming (ILP) with a fixed number of variables [11]. In this case, we require that the weights of the jobs, $y_1, \ldots, y_r$ are all rational numbers represented by $O(\log n)$ bits. As before, E is the set of possible assignments to a machine. For each vector $\vec{e} \in E$ denote by $x_{\vec{e}}$ the numbers of machines that were assigned $\vec{e}$. In these terms, A feasible solution to the problem is a non-negative integer vector $\vec{X}$ such that $\sum_{\vec{e} \in E} x_{\vec{e}} = m$, i.e., m machines are used, and $\sum_{\vec{e} \in E} x_{\vec{e}} \cdot \vec{e} = \vec{n}$, i.e., all jobs were assigned. The p power of the L_p norm of a solution $\vec{X}$ is $\sum_{\vec{e} \in E} x_{\vec{e}} \cdot W(\vec{e})^p$. Clearly the optimal solution is given by the following integer system:

$$\min \sum_{\vec{e} \in E} x_{\vec{e}} \cdot W(\vec{e})^p$$

subject to

$$\begin{aligned} &\sum_{\vec{e} \in E} x_{\vec{e}} = m \\ &\sum_{\vec{e} \in E} x_{\vec{e}} \cdot \vec{e} = \vec{n} \\ &x_{\vec{e}} \geq 0 \qquad\qquad \forall \vec{e} \in E \end{aligned}$$

Notice that the dimension and the number of equations are fixed. We apply Lenstra [11] method to find the optimal solution of the ILP. The time complexity of Lenstra's algorithm is exponential in the dimension of the program but polynomial in the logarithms of the coefficients and therefore is $O(f(r, \frac{1}{\alpha}) \log^{O(1)} n)$ where f is a function of the two constants r and $1/\alpha$.

Therefore, we have shown:

THEOREM 3.2. *Given m machines and n jobs with rational weights in the set $\{y_1, \ldots, y_r\}$ and $\min_i\{y_i\} \geq \alpha L$, where α and r are constants independent of the input, ILP finds the best assignment in terms of the L_p norm in time $O(n + f(r, \frac{1}{\alpha}) \log^{O(1)} n)$.*

4 The general case

We now describe the polynomial approximation scheme for the general case. We are given a set of n jobs with weights $w_1, \ldots, w_n$ and m machines. For any positive (constant) $\epsilon \leq 1$ we describe an algorithm $Approx_\epsilon$ that finds an assignment of the jobs, with load vector $\vec{l}$ such that $|\vec{l}|_p \leq (1+4\epsilon)|\vec{l}^{opt}|_p$ where $\vec{l}^{opt}$ is an optimal load vector in the L_p norm.

We define 3 sets of inputs : I, I_2, I_3.

- Denote by I the original input to the problem. Denote by L the average load on the machines for this input.
- Define a set of small jobs $J = \{j : w_j < \epsilon L\}$. Denote by $W(J)$ the total weight of the jobs in J, that is $W(J) = \sum_{j \in J} w_j$. I_3 is generated from I by replacing the jobs in J with $R = \lfloor W(J)/\epsilon L \rfloor$ jobs each of weight ϵL. Notice that $R \leq |J|$, therefore the number of jobs needed to be assigned in I_3 is at most n.
- For a non-negative integer k, define $c_k = \epsilon L + k \cdot \epsilon^2 L$. We generate I_2 from I_3 simply by replacing the weight w_j by $\hat{w}_j = \max\{c_k : c_k \leq w_j\}$ for each j. Since $w_j \leq L$ for all j, then

  $$\hat{w}_j \in C = \{\epsilon L + k \cdot \epsilon^2 L : 0 \leq k \leq \frac{1-\epsilon}{\epsilon^2}\} \ .$$

  Note that the number of jobs is the same as in I_3, hence, at most n.

We are ready to describe algorithm $Approx_\epsilon$:

1. Generate the 3 inputs I, I_3, I_2
2. Solve I_2 using an algorithm for constant number of jobs ($DYNPROG$ or ILP).
3. Apply the solution of I_2 to I_3, simply by changing the weight of each job.
4. Convert the solution of I_3 to a solution for I by replacing R jobs of weight ϵL with the jobs from J in a greedy procedure described below.

Next we describe how to replace the R jobs of weight ϵL with jobs from J. Let r_i be the number of such jobs assigned to machine i. For each machine i, we assign jobs arbitrarily from J of total weight of at most $(r_i + 1) \cdot \epsilon L$. We claim that all the jobs in J are assigned in this process. Assume that some jobs are left unassigned. Since each job is of size of at most ϵL then for each i machine i is at least $r_i \cdot \epsilon L$ loaded with small jobs. If we force the job to be assigned to the first machine, the total load of small jobs assigned to it exceeds $(r_1 + 1)\epsilon L$. Hence

$$W(J) > ((\sum_{i=1}^{m} r_i) + 1)\epsilon L \geq (R+1) \cdot \epsilon L$$

which is a contradiction.

The *running time* of step $1, 3$ and 4 in $Approx_\epsilon$ is $\Theta(n)$. In step 2, we use either $DYNPROG$ or ILP. Note that $\alpha = \epsilon$ and $r \leq 1/\epsilon^2$. Using theorem 3.2, the running time of step 2 and the total running time is

$O(n + g(\frac{1}{\epsilon}) \log^{O(1)} n)$ where g grows (somewhat worse than) exponentially in $1/\epsilon$.

We are now ready to show that the solution to I, which is a feasible assignment to the original problem's input, is a $(1+4\epsilon)$ approximation to that problem. We do this in steps according to the algorithm, showing that in each step we do not get far from the optimal solution.

Denote by $\vec{l^{opt}}$ the load vector on the machines of an optimal assignment, and by $\vec{l^{(2)}}$ the load vector on the machines of the optimal solution to I_2 generated in step 2 of the algorithm. The following claim will show that $|\vec{l^{(2)}}|_p$ is not much larger than $|\vec{l^{opt}}|_p$.

Claim 4.1. $|\vec{l^{(2)}}|_p \leq |\vec{l^{opt}}|_p + \epsilon L \cdot m^{\frac{1}{p}}$.

Proof. Consider an optimal assignment for input I. Let Z_i be the total weight of jobs in J assigned to machine i. For each machine i, replace the jobs in J, by at most $\lceil Z_i/(\epsilon L) \rceil$ jobs of weight ϵL. This is always possible since $\sum_{i=1}^{m} \lceil Z_i/(\epsilon L) \rceil \geq R$. This defines an assignment to the input I_3. Next, for each j replace the weight w_j by $\hat{w}_j$. This defines a feasible solution for input I_2. Denote by $\vec{x}$ the load vector on the machines generated by this process. Since $\vec{l^{(2)}}$ is optimal, $|\vec{l^{(2)}}|_p \leq |\vec{x}|_p$. The load on each machine in this process was increased by less than ϵL. Hence, $l_i^{opt} + \epsilon L > x_i$ for each i. Therefore, by the triangle inequality

$$|\vec{l^{(2)}}|_p \leq |\vec{x}|_p \leq |\vec{l^{opt}} + \epsilon L \vec{1}|_p \leq |\vec{l^{opt}}|_p + \epsilon L \cdot m^{\frac{1}{p}} \ .$$

We move now to step 3 in the algorithm. Denote by $\vec{l^{(3)}}$ the load vector on the machines in the solution of I_3. Next we claim that $|\vec{l^{(3)}}|_p$ is not much larger than $|\vec{l^{(2)}}|_p$.

Claim 4.2. $|\vec{l^{(3)}}|_p \leq (1+\epsilon)|\vec{l^{(2)}}|_p$.

Proof. For each job j

$$\frac{w_j}{\hat{w}_j} = 1 + \frac{w_j - \hat{w}_j}{\hat{w}_j} \leq 1 + \frac{\epsilon^2 L}{\hat{w}_j} \leq 1 + \frac{\epsilon^2 L}{\epsilon L} = 1 + \epsilon \ .$$

Hence, $l_i^{(3)}/l_i^{(2)} \leq 1 + \epsilon$ for each machine i. Thus

$$|\vec{l^{(3)}}|_p \leq |(1+\epsilon)\vec{l^{(2)}}|_p = (1+\epsilon)|\vec{l^{(2)}}|_p \ .$$

In step 4 we use a greedy procedure in order to replace R jobs of weight ϵL with the small jobs in J. We now show that the greedy procedure performs well. Denote by $\vec{l}$ the load vector on machines generated by our algorithm for the solution to I.

Claim 4.3. $|\vec{l}|_p \leq |\vec{l^3}|_p + \epsilon L \cdot m^{\frac{1}{p}}$.

Proof. Recall that the greedy process simply adds to each machine $1 \leq i \leq m$, jobs from J with total load of at least $r_i \cdot \epsilon L$, but less than $(r_i + 1) \cdot \epsilon L$. Thus $l_i \leq l_i^{(3)} + \epsilon L$, for $1 \leq i \leq n$. Hence, by the triangle inequality

$$|\vec{l}|_p \leq |\vec{l^{(3)}} + \epsilon L \vec{1}|_p \leq |\vec{l^{(3)}}|_p + \epsilon L \cdot m^{\frac{1}{p}} \ .$$

The next simple lemma lower bounds the performance of the optimal solution:

Lemma 4.1. *Any assignment of n jobs on m machines, with average load L, has a total load of at least $L \cdot m^{\frac{1}{p}}$.*

Proof. This follows immediately from the convexity of the function x^p.

Putting all this together, we have

$$\begin{aligned}
|\vec{l}|_p &\leq |\vec{l^{(3)}}|_p + \epsilon L \cdot m^{\frac{1}{p}} \\
&\leq (1+\epsilon)|\vec{l^{(2)}}|_p + \epsilon L \cdot m^{\frac{1}{p}} \\
&\leq (1+\epsilon)(|\vec{l^{opt}}|_p + \epsilon L \cdot m^{\frac{1}{p}}) + \epsilon L \cdot m^{\frac{1}{p}} \\
&\leq (1+\epsilon)(|\vec{l^{opt}}|_p + \epsilon |\vec{l^{opt}}|_p) + \epsilon |\vec{l^{opt}}|_p \\
&\leq (1+4\epsilon)|\vec{l^{opt}}|_p \ .
\end{aligned}$$

By replacing ϵ by $\epsilon/4$ we conclude:

Theorem 4.1. *Given m machines and n jobs, and any $\epsilon > 0$, algorithm $Approx_\epsilon$ finds in linear time in n (the constants depend on $1/\epsilon$) an assignment of the jobs on the machines, whose total load (in the L_p norm) is at most $(1+\epsilon)$ of the optimal.*

5 Unit jobs restricted assignment

In this section we consider the case of unit jobs, each of which should be assigned to one out of subset of machines that is associated with it. We define it formally as follows. Let $G = (U, V, E)$ be a bipartite graph having $|U| = n$ vertices on one side (jobs or tasks) and $|V| = m$ vertices on the other side (servers or machines) connected by the set of edges E where the degree of each vertex $u \in U$ is at least 1. We define an *assignment* H as an assignment of each job $u \in U$ to a machine $v \in V$ such that $(u, v) \in E$. In other words $H \subseteq G$ such that $deg(u) = 1$ for all $u \in U$. Note that $d_v = deg(v)$ for $v \in V$ might be more than 1. The number of jobs assigned to a machine $v \in V$ is referred to as the *load* of vertex v or its *degree* in the assignment. The goal is to come up with an assignment that minimizes the L_p norm of the load vector on V.

Denote by d_H the load vector of some assignment H. For $T \subseteq V$ define

$$S_T = \sum_{i \in T} d_i$$

and

$$M_T = \min_{i \in T} d_i \ .$$

For $1 \leq k \leq m$ let S_k be the value of S_T for the set T of the k most loaded machines of U. An assignment is called strongly-optimal if for any assignment H' and for all $1 \leq k \leq m$

$$S_k \leq S_{k'}$$

Our main result in this section is the following

THEOREM 5.1. *For any G there exists a strongly-optimal assignment. Moreover, it can be found in polynomial time.*

The proof follows from Theorem 5.3 described below.

Strongly-optimal assignments are of a special interest since they are optimal in any norm. More specifically:

THEOREM 5.2. *Let H be a strongly-optimal assignment. For any $p \geq 1$ and any assignment H'*

$$|d_H|_p \leq |d_{H'}|_p.$$

Proof. Let h (resp. h') be the vector d_H (resp. $d_{H'}$) with the coordinates sorted in non-increasing order. Clearly $|h|_p = |d_H|_p$ and $|h'|_p = |d_{H'}|_p$. Here for $1 \leq i \leq k$

$$\sum_{i=1}^{k} h_i \leq \sum_{i=1}^{k} h'_i \ .$$

It is not difficult to show that one can generate a sequence of steps that starts with the vector h and ends with h' where in each step one unit moves from some coordinate to a coordinate with larger or equal value. Each such step can only increase the norm and thus $|h|_p \leq |h'|_p$.

For an assignment H we define an alternating path

$$P = v_0 u_0 v_1 \ldots v_{k-1} u_{k-1} v_k$$

for $k \geq 0$ as a simple path such that $(v_i, u_i) \in H$ and $(u_i, v_{i+1}) \in G$ for $0 \leq i \leq k-1$. If in addition $d_{v_k} \leq d_{v_0} - 2$ the path is called an augmenting path.

Given an assignment H and an augmenting path P we can generate a new assignment by replacing the odd edges of the path by the even ones. The new assignment has the following properties: d_{v_0} is reduced by 1 while d_{v_k} is increased by 1, and the degrees of all other vertices do not change.

We make the following easy observations:

1. For any assignment $0 < \sum_{i=1}^{m} d_i^2 \leq n^2$.
2. An augmenting path reduces $\sum_{i=1}^{m} d_i^2$ by at least 2.
3. An augmenting path from a vertex can be found (if one exists) in linear time (e.g. by Breadth First Search).

We are ready to describe the algorithm that generates a strongly-optimal assignment.

ALG-Augment: Start with an arbitrary assignment. Improve the assignment repetitively as long as an augmenting path exists.

THEOREM 5.3. **ALG-Augment** *terminates in polynomial time and generates a strongly-optimal assignment.*

Proof. By the previous observations the number of times we may augment an assignment is at most n^2. Finding an augmenting path and improving the assignment can be done in $O(n|E|)$ (we make no attempt to achieve the best complexity here). Thus the algorithm terminates in polynomial time.

It is left to prove that an assignment H that cannot be improved by an augmenting path is strongly-optimal. Assume by contradiction that H is not strongly-optimal. Then there exists an assignment H' and some k such that

$$S_k \geq S'_k + 1 \ .$$

Let k_0 be the minimal k with the above property. Let T_0 (resp. T'_0) be a set of the k_0 most loaded machines of the assignment H (resp. H') . Note that $S_k = S_{k-1} + M_{T_0}$ and $S'_k = S'_{k-1} + M'_{T'_0}$. The minimality of k_0 implies that

$$M_{T_0} \geq M'_{T'_0} + 1 \ .$$

For the assignment H let $T_1 = \{i | d_i \geq M_{T_0}\}$. Clearly $T_0 \subseteq T_1$. Consider all alternating paths that start at T_1. Let T_2 be the set of ending points of these paths and put $k_2 = |T_2|$. Clearly $T_1 \subseteq T_2$. It is important to note that T_2 is closed under alternating paths. More specifically, there are no alternating paths from machines of T_2 to machines outside T_2, otherwise, there were such paths from a machine in T_1 to a machine outside T_2.

If there exists $i \in T_2$ such that $d_i \leq M_{T_0} - 2$ then it would yield an augmenting path which contradicts the fact that H cannot be augmented. Thus for each $j \in T_2$

$$d_j \geq M_{T_0} - 1 \geq M'_{T'_0} \ .$$

Thus

$$\begin{aligned} S_{T_2} &\geq S_{T_0} + M_{T_2}|T_2 - T_0| \\ &\geq S_{k_0} + (M_{T_0} - 1)(k_2 - k_0) \\ &\geq S'_{k_0} + 1 + M'_{T'_0}(k_2 - k_0) \\ &\geq S'_{k_2} + 1 \\ &\geq S'_{T_2} + 1 \end{aligned}$$

where the last inequality follows from the fact that S'_{k_2} is the sum of the k_2 most loaded machines by H' and S'_{T_2} is the sum of the loads on a specific set T_2 of size k_2. The inequality before that follows from the fact that $M'_{T'_0}$ is the k_0'th largest load induced by H'.

Hence among all the jobs assigned by H to machines of the set T_2 at least one is assigned by H' to $j \notin T_2$. This defines an alternating path out of T_2 which contradicts the fact that T_2 is closed under alternating paths.

References

[1] B. Awerbuch, Y. Azar, E. Grove, M. Kao, P. Krishnan, and J. Vitter. Load balancing in the l_p norm. In *Proc. 36th IEEE Symp. on Found. of Comp. Science*, pages 383–391, 1995.

[2] Y. Azar and J. Sgall. unpublished notes.

[3] Yossi Azar, Joseph Naor, and Raphael Rom. The competitiveness of on-line assignment. In *Proc. 3rd ACM-SIAM Symp. on Discrete Algorithms*, pages 203–210, 1992.

[4] A.K. Chandra and C.K. Wong. Worst-case analysis of a placement algorithm related to storage allocation. *SIAM Journal on Computing*, 4(3):249–263, 1975.

[5] R.A. Cody and E.G. Coffman, Jr. Record allocation for minimizing expected retrieval costs on crum-like storage devices. *J. Assoc. Comput. Mach.*, 23(1):103–115, January 1976.

[6] M.R. Garey and D.S. Johnson. *Computers and Intractability*. W.H. Freeman and Company, San Francisco, 1979.

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

[8] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math*, 17:263–269, 1969.

[9] 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.

[10] Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *J. of the ACM*, 34(1):144–162, January 1987.

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

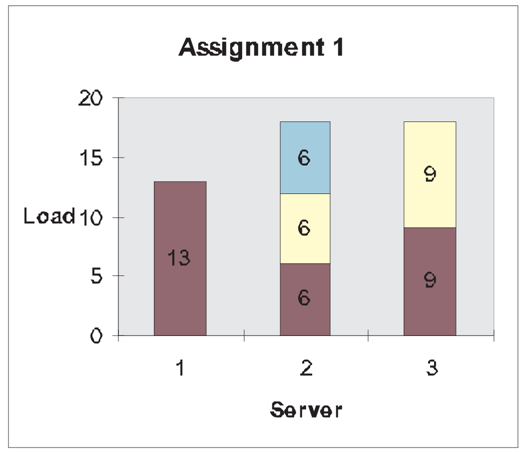
[12] J.K. Lenstra, D.B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Prog.*, 46:259–271, 1990.

[13] J.Y.T. Leung and W.D. Wei. Tighter bounds on a heuristic for a partition problem. *Information Processing Letters*, 56:51–57, 1995.

6 Appendix

We first show an example where an optimal assignment in the L_∞ norm is not optimal in the L_2 norm.

The example consists of 3 machines and 6 jobs with weights $13, 9, 9, 6, 6, 6$. Figure 1 shows two feasible assignments for the problem:



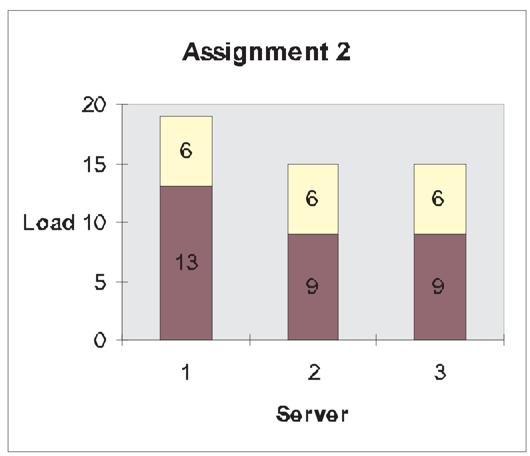


Figure 1: The best assignment in L_∞ and L_2 norm

Assignment 1 is optimal in terms of the L_∞ norm. Its value in the L_∞ norm is 18, and its value in the L_2 norm is $\sqrt{817}$. Assignment 2 is optimal in terms of the L_2 norm. Its value in the L_∞ norm is 19, and its value in the L_2 norm is $\sqrt{811}$.

Next we show that the following process cannot yield an ϵ-approximation scheme. First remove the small jobs, then get the exact (not just an approximate) solution, and at last add the small jobs optimally.

We can use the same example as before with a set of very small jobs of total weight 5. Obviously, the optimal solution has a load vector $(18, 18, 18)$ which is assigning the small jobs on machine 1 in Assignment 1.

However, if we remove the small jobs then assignment 2 is the optimal assignment for the remaining jobs and any procedure that assigns the small jobs cannot produce a load vector better than $(19, 17.5, 17.5)$. The latter assignment approximate the optimal solution by a relative error of $7 \cdot 10^{-4}$. For ϵ which is much smaller than that and small enough jobs the scheme does not provide an approximation which is even close to ϵ.