All-Norm Approximation for Scheduling on Identical Machines

Yossi Azar [*] Shai Taub [†]

Abstract

We consider the problem of assigning jobs to m identical machines. The load of a machine is the sum of the weights of jobs assigned to it. The goal is to minimize the norm of the resulting load vector. It is known that for any fixed norm there is a PTAS. On the other hand, it is also known that there is no single assignment which is optimal for all norms. We show that there exists one assignment which simultaneously guarantees a 1.388-approximation of the optimal assignments for all norms. This improves the 1.5 approximation given by Chandra and Wong in 1975.

1 Introduction

The problem of machine scheduling is one of the most researched problems in the area of approximation algorithms. The identical machines model is defined by m parallel machines and n independent jobs, where each job j has a non-negative weight w_j. Each job should be assigned to one of the machines, and the load of each machine i, l_i, is defined as the sum of weights of all jobs assigned to it. The goal of the problem is to get the best assignment. For each specific norm ℓ_p $(p > 1)$, this is defined as the assignment that minimizes $\|(l_1, \ldots, l_m)\|_p = (\sum_{i=1}^{m} l_i^p)^{1/p}$. Specifically, for the case of the ℓ_∞ norm (makespan), the goal is to minimize the maximum load of all the machines. In this paper we describe an algorithm that finds an assignment that simultaneously provides a good approximation for all the optimal assignments of all the ℓ_p norms.

It is well known that for any specific norm, one can find for every $\varepsilon > 0$ a polynomial time algorithm for the problem that provides an approximation ratio of $1+\varepsilon$ (PTAS). Hochbaum and Shmoys [11] showed that there is a PTAS of minimizing the makespan (ℓ_∞). Later, it was shown in [1], that there is a PTAS for every ℓ_p norm. However, this does not mean that for every positive ε there exists an assignment that approximates the optimal assignment of various norms simultaneously, up to a factor of $1+\varepsilon$. Actually, in the same paper there is an example that shows that optimal solutions for the ℓ_2 and ℓ_∞ norms are achieved from different assignments. In the Appendix we generalize this example, and prove that for every two different norms we can find an input for which the optimal assignment of the

[*] `azar@tau.ac.il`. School of Computer Science, Tel-Aviv University, Tel-Aviv, 69978, Israel. Research supported in part by the Israeli Ministry of industry and trade and by the Israel Science Foundation.

[†] `shai_ta@netvision.net.il`. School of Computer Science, Tel-Aviv University, Tel-Aviv, 69978, Israel. Research supported in part by the Israeli Ministry of industry and trade.

two norms differ. By that we conclude that in general there is no assignment that $1+\varepsilon$ approximates the optimal assignments of both norms, for small enough ε. Moreover, for any set of different ℓ_p norms there is an example in which there is a different optimal assignment for each norm of the set.

Now that we know that we cannot find an approximation scheme (not necessarily polynomial time) even for two different norms simultaneously, we want to find an algorithm which finds an assignment that approximates the optimal assignments of all ℓ_p norms simultaneously, with a small constant approximation ratio.

Chandra and Wong showed in [4], that the algorithm that sorts the jobs from the biggest weight to the smallest, and then sequentially greedily assigns each job to the least loaded machine, gives an approximation ratio of 1.5 for all norms simultaneously. In particular, this algorithm gives an approximation ratio of $\frac{4}{3}$ for the ℓ_∞ norm (see [8]), and 1.021 for the ℓ_2 norm.

Our results: Our main result is a polynomial time algorithm that provides an assignment that approximates the optimal assignment of all norms simultaneously within a factor of 1.3875. This improves the 1.5 approximation ratio given in [4].

As mentioned above we also show (in the Appendix) that for any two norms ℓ_p and ℓ_q there exists an input that has two different optimal assignments for both norms, and moreover, for any set of norms $\ell_{p_1}, \ell_{p_2}, \dots, \ell_{p_k}$ there is an input for which all the optimal assignments of all these norms differ. This proves that an approximation scheme (not necessarily polynomial time) for the problem of approximating two out of several norms simultaneously, does not exist.

Other related results: Goel *et al.* [7] introduced the definition of globally α-balance. Goel and Meyerson [6] showed that an assignment which is globally α-balanced (defined later) also α-approximates all optimal assignments for all norms (as well as α-approximates the optimal assignments for every convex function on the loads of the machines). They also considered the problem of finding a globally α-balanced assignment for identical machines. They showed how to find a PTAS for an assignment that is globally α-balanced with the best (smallest) value of α, but did not give a bound of this α. In this paper we show that this α is actually bounded by 1.3875.

Other scheduling models have also been studied: related machines, restricted assignment (subset model) and unrelated machines. In the related machines model (i.e. machines have fixed speeds) there is no assignment that approximates all norms simultaneously within a constant factor [3]. This obviously holds for the unrelated machines model as well. The same paper also shows an algorithm that simultaneously 2-approximates the optimal solutions of all norms in the restricted assignment model (i.e. each job arrives with a set of machines that it can be assigned to). We note that more is known for approximating any fixed norm. For the related machines model a PTAS was given by Hochbaum and Shmoys [10] for ℓ_∞ norm and by Epstein and Sgall [5] for any fixed ℓ_p norm. For the restricted assignment model 2-approximation was achieved by Lenstra *et al.* [13] for ℓ_∞ norm and by [3] for any other ℓ_p norm. Moreover, in these papers it was also shown that a PTAS does not exist for ℓ_∞ as well as for any ℓ_p norm ($p > 1$). In the unrelated machines model 2-approximation algorithm was achieved by [13] for ℓ_∞ and a $\theta(p)$ approximation ratio for any other ℓ_p norm [2] (see [12] and [14] for other related results).

Paper structure: In section 2 we repeat the definition of globally α-balance taken from [7] and explain its connection to the approximation of all norms showed in [6]. We also describe a tool used by the algorithm. In section 3 we describe the all-norm approximation algorithm. In subsection 3.1 we show how to easily handle the huge jobs, i.e., jobs which are larger than the average load on the machines. In subsection 3.2 we show how to assign the small jobs (defined later) after assigning the big ones without increasing the imbalance. In subsection 3.3 we show how to find a balanced assignment for the big jobs and by that complete the algorithm and its proof. In the appendix A we show that an approximation scheme for more than one norm does not exist.

2 Definitions and Observations

We use the definition of globally α-balance used in [7], to prove the all-norm approximation. We will briefly repeat some definitions and a theorem that explain the importance of this quantity.
Let $S_k(x)$ denote the sum of the loads of the k most loaded machines in the assignment x, for $1 \leq k \leq m$.

Definition 2.1 *For $\alpha \geq 1$, given two assignments x and y (not necessarily of the same jobs), we say that x is α-submajorized by y, if for every k ($1 \leq k \leq m$), $S_k(x) \leq \alpha S_k(y)$. This will be denoted by $x <_\alpha y$.*

Definition 2.2 *Assignment P is called globally α-balanced if for any other feasible assignment P' of the same jobs, we have $P <_\alpha P'$.*

The next theorem will define our way of proving an all-norm approximation. The proof of the theorem is based on the basic theorem of Hardy *et al.* [9] (see [6]).

Theorem 2.1 *If an assignment is globally α-balanced, then it α-approximates the optimal assignment of all ℓ_p norms ($p \geq 1$).*

The way to prove that an assignment P α-approximates the optimal assignment of each ℓ_p norm, is to pick any other assignment P', and prove that $P <_\alpha P'$. A useful tool used by our algorithm is separating the problem into smaller problems. For that we define the union of assignments and prove lemmas using it.

Definition 2.3 *Given an assignment P_1 of n_1 jobs on m_1 machines, and an assignment P_2 of n_2 jobs on m_2 machines, the (disjoint) union of P_1 and P_2 (denoted by $P_1 \cup P_2$) is the assignment on $m_1 + m_2$ machines, that assigns the n_1 jobs from P_1 on m_1 machines as P_1, and the n_2 jobs from P_2 on the m_2 machines as P_2.*

It is easy to see that given two assignments, P_1 on m_1 machines, and P_2 on m_2 machines, we have

$$S_k(P_1) + S_l(P_2) \leq S_{k+l}(P_1 \cup P_2) \tag{1}$$

for every k, $1 \leq k \leq m_1$ and l, $1 \leq l \leq m_2$.

Lemma 2.1 *Let P_1, Q_1 be two different assignments on m_1 machines (not necessarily consisting of the same jobs), and let P_2, Q_2 be two different assignments on m_2 machines. If $P_1 <_\alpha Q_1$ and $P_2 <_\alpha Q_2$, then the assignment which is the union of P_1 and P_2 on the $m_1 + m_2$ machines is α-submajorized by the assignment which is the union of Q_1 and Q_2.*

Proof: Consider the k most loaded machines in the union of P_1 and P_2. They include the most loaded machines in P_1 and P_2. Assume they include l machines from P_1 and $k - l$ machines from P_2. Then for every k, $1 \leq k \leq m_1 + m_2$ we have

$$\begin{aligned} S_k(P_1 \cup P_2) &= S_l(P_1) + S_{k-l}(P_2) \\ &\leq \alpha S_l(Q_1) + \alpha S_{k-l}(Q_2) \\ &\leq \alpha S_k(Q_1 \cup Q_2) \end{aligned}$$

where the last inequality follows from (1). ■

From the lemma above we can conclude by induction the following lemma:

Lemma 2.2 *Let P_i, Q_i be two different assignments on m_i machines, for $1 \leq i \leq k$ (not necessarily consisting of the same jobs). If $P_i <_\alpha Q_i$ for every i, then $\cup_{i=1}^k P_i <_\alpha \cup_{i=1}^k Q_i$ on the $\sum_{i=1}^k m_i$ machines.*

3 All Norm Approximation

Our algorithm consists of 4 phases. In the first phase, subsection 3.1, we eliminate the huge jobs (jobs of weight larger than the average load of the machines). In the second phase, subsection 3.2, we eliminate the small jobs (jobs of weight smaller than some constant fraction of the average load of the machines). In the third phase we repeat the first phase for the new huge jobs created by eliminating the small jobs in the second phase. Now, we are left only with big jobs, i.e., jobs which are neither huge nor small. In the forth phase, subsection 3.3, we solve the problem for the big jobs. This yields the main result concluded in Theorem 3.4 which states that our algorithm produces a globally 1.3875-balanced assignment.

3.1 Handling Huge Jobs

We apply normalization on the weights by dividing each of the weights by $\frac{\sum_{i=1}^n w_i}{m}$. Then we get $\sum_{i=1}^n w_j = m$, and the average load over all machines is exactly 1.

Definition 3.1 *An assignment P is called "reasonable" if by removing any job from the machine it was assigned to, the load of that machine becomes smaller than 1.*

Lemma 3.1 *If assignment P is not reasonable, then there exists an assignment P', such that $P' <_1 P$.*

Proof: If P is not reasonable, then there exists a machine A and a job j on A, such that if we remove j from A, the total load of A is still not smaller than 1. We build the assignment P' by assigning j to a machine B whose load is less than 1 (such a machine must exist). The other jobs will be assigned in P' as they were assigned in P. Clearly, $P' <_1 P$ since the total load of A and B is the same in P and P', each of the machines A and B in P' have smaller loads than the machine A in P, and all the other machines are unchanged.

If P' is reasonable, we are done. If not, we will continue this process with P' until we get a reasonable assignment. This process must end since we have a finite number of possible assignments, and in every step, the sum of squares of the loads of the machines is reduced. ■

From this lemma it is clear that if we want to prove that an assignment P gives an approximation of α for all norms, it is enough to pick any other **reasonable** assignment P', and prove that $P <_\alpha P'$.

Clearly, in any reasonable assignment, a job whose weight is at least 1 is assigned to a machine by itself. Therefore, our algorithm has the following structure:

all-norm algorithm(preliminary version)

```
1. Normalize weights to get an average load of 1.
2. While there are jobs of weight $\geq 1$ ("huge jobs") do
    (a) Assign each of these jobs individually to a machine and delete these jobs and
        machines.
    (b) Renormalize weights with the remaining jobs and the remaining machines.
3. Handle the remaining jobs and the remaining machines.
4. Insert the jobs and the machines that were deleted in step 2a.
```

Claim 3.1 *If there exists an algorithm for jobs of weight smaller than* 1 *(where* 1 *is the average load of the machines), that provides an assignment which is globally α-balanced, then there is an algorithm that provides an assignment which is globally α-balanced for any input.*

Proof: Use the above algorithm, where you plug into step 3 the algorithm that handles jobs of weight smaller than 1. We will prove that the above algorithm provides an assignment which is globally α-balanced, by comparing for any given input, the assignment provided by this algorithm to any other reasonable assignment.

For a given input, consider the last round that step 2 was applied. Let m' be the number of machines and N' the set of the remaining jobs at the beginning of that last round. Denote by k' be the number of jobs of weight larger than 1 ("huge jobs") at that last round. Let P' be the assignment provided by the above algorithm for the set of jobs N' , let P_1 be the assignment of the k' huge jobs on k' machines in P', and let P_2' be the assignment of the other jobs provided by the algorithm in step 3 on the other $m' - k'$ machines ($P' = P_1' \cup P_2'$). Let Q' be another reasonable assignment for the set of jobs N', let Q_1' be Q' for the k'

huge jobs (which are, of course, assigned to separate k' machines), and let Q'_2 be Q' on the other $m' - k'$ machines ($Q' = Q'_1 \cup Q'_2$). From the assumption of the claim, $P'_2 <_\alpha Q'_2$. Clearly $P'_1 <_1 Q'_1$, and in particular, $P'_1 <_\alpha Q'_1$. Therefore, from Lemma 2.1, $P' <_\alpha Q'$. This complete the proof for the last round of step 2. By repeating this argument for any round of step 2 of the above algorithm (where the average load 1 is different in each such round), we complete the proof. ■

3.2 Handling Small Jobs

We first show that given n jobs, we can separate the jobs into big jobs and small jobs, so that if we could find a good assignment for the big jobs, we could easily add the small jobs without damaging the balance of the assignment.

Theorem 3.1 *Given n jobs, whose average load is normalized to 1, if there exists an assignment P of all jobs whose weights are bigger than β ($0 < \beta < 1$), which is globally $(1+\beta)$-balanced, then adding the small jobs sequentially greedily in any order (each job on the current least loaded machine) also creates a globally $(1+\beta)$-balanced assignment.*

Proof: Let $(P_1, P_2, \ldots, P_m)$ be the load vector of the machines ordered in non-increasing order in the assignment P of the big jobs (the jobs whose weights are bigger than β). Let Q be any other assignment of the big jobs, and $(Q_1, Q_2, \ldots, Q_m)$ be the load vector of the machines in Q, ordered in non-increasing order. Then from the assumption of the theorem,

$$P <_{1+\beta} Q \ . \tag{2}$$

Let P' be the assignment created by adding the small jobs to P sequentially greedily, and let $(P'_1, P'_2, \ldots, P'_m)$ be the load vector of the machines in P', ordered in non-increasing order. Let Q' be the assignment created by adding the small jobs to Q in an arbitrary way, and let $(Q'_1, Q'_2, \ldots, Q'_m)$ be the load vector of the machines in Q', ordered in non-increasing order. Note that Q' stands for an arbitrary assignment. Clearly, for every k, $1 \le k \le m$:

$$Q_k \le Q'_k \ . \tag{3}$$

Let l ($l \le m$) be the largest integer such that $P_k = P'_k$ for every $k \le l$ (if there is no such l, we define $l = 0$). Then for each $k \le l$ we have:

$$S_k(P') = \sum_{i=1}^{k} P'_i = \sum_{i=1}^{k} P_i \le (1+\beta) \sum_{i=1}^{k} Q_i \le (1+\beta) \sum_{i=1}^{k} Q'_i = (1+\beta) S_k(Q') \ . \tag{4}$$

The first inequality follows from (2) and the second inequality follows from (3).

Note that in the $m - l$ least loaded machines in P', the difference between the loads of the most loaded machine and the least loaded machine is at most β (otherwise, the last job that was greedily assigned to the most loaded machine, should not have been assigned to it, since it was not the least loaded machine at that moment). Hence, the difference between each of the loads of these machines and their average load is at most β:

$$P'_j \ \le \ \beta + \frac{\sum_{i=l+1}^{m} P'_i}{m - l}$$

$$
\begin{aligned}
&= \beta + \frac{m - \sum_{i=1}^{l} P_i'}{m-l} && (5)\\
&\le \beta + \frac{m - \sum_{i=1}^{l} P_i}{m-l}
\end{aligned}
$$

for every j, $l+1 \le j \le m$. The equality holds since the sum of all the weights in P' is normalized to be m, and the second inequality follows from the definition of l.

Note also that for any $1 \le k \le m-l$,

$$
\sum_{i=l+1}^{l+k} Q_i' \ge k\left(\frac{\sum_{i=l+1}^{m} Q_i'}{m-l}\right) = k\left(\frac{m - \sum_{i=1}^{l} Q_i'}{m-l}\right) \tag{6}
$$

where the inequality is follow from the fact that the sum of the k biggest values in the vector $(Q_{l+1}', Q_{l+2}', \ldots, Q_m')$ is not smaller than k times the average value in this vector. The equality is true since the sum of all the weights in Q' is normalized to m.

Now for every k $(0 \le k \le m-l)$, we will compare the sums of the $l+k$ most loaded machines in both assignments. The sum of the $l+k$ most loaded machines in our assignment is:

$$
\begin{aligned}
S_{k+l}(P') &= \sum_{i=1}^{k+l} P_i' \\
&= \sum_{i=1}^{l} P_i' + \sum_{i=l+1}^{l+k} P_i' \\
&\le \sum_{i=1}^{l} P_i + k\left(\beta + \frac{m - \sum_{i=1}^{l} P_i}{m-l}\right) && (7)\\
&= \left(1 - \frac{k}{m-l}\right)\sum_{i=1}^{l} P_i + \beta k + \frac{km}{m-l} \\
&\le \left(1 - \frac{k}{m-l}\right)(1+\beta)\sum_{i=1}^{l} Q_i + (1+\beta)\frac{km}{m-l}\ .
\end{aligned}
$$

The first inequality follows from (5) and the second inequality follows from (2).

The sum of the $l+k$ most loaded machines in the other assignment is:

$$
\begin{aligned}
S_{k+l}(Q') &= \sum_{i=1}^{k+l} Q_i' \\
&= \sum_{i=1}^{l} Q_i' + \sum_{i=l+1}^{l+k} Q_i' \\
&\ge \sum_{i=1}^{l} Q_i' + k\left(\frac{m - \sum_{i=1}^{l} Q_i'}{m-l}\right) && (8)\\
&= \left(1 - \frac{k}{m-l}\right)\sum_{i=1}^{l} Q_i' + \frac{km}{m-l}
\end{aligned}
$$

$$\geq \left(1 - \frac{k}{m-l}\right) \sum_{i=1}^{l} Q_i + \frac{km}{m-l} \; .$$

The first inequality follows from (6) and the second inequality follows from (3).

From (4), (7) and (8) we get $S_k(P') \leq (1+\beta)S_k(Q')$ for every k, $1 \leq k \leq m$. Since any assignment can be composed from an assignment Q of the big jobs by adding the small jobs in some way, we conclude that our algorithm provides a globally $(1+\beta)$-balanced assignment. ■

By the above theorem, in order to get an assignment which is $(1+\beta)$-balanced, our algorithm is defined as follows:

all-norm algorithm

1. Normalize weights to get an average load of 1.
2. While there are jobs of weight ≥ 1 ("huge jobs") do:
   (a) Assign each of these jobs individually to a machine and delete these jobs and machines.
   (b) Renormalize weights with the remaining jobs and the remaining machines.
3. Put the jobs of weight smaller than β ("small jobs") aside.
4. Renormalize weights to get an average load of 1.
5. While there are jobs of weight ≥ 1 ("huge jobs") do:
   (a) Assign each of these jobs individually to a machine and delete these jobs and machines.
   (b) Renormalize weights with the remaining jobs and the remaining machines.
6. Handle the jobs of weight between β and 1 ("big jobs"), as will be described later (subsection 3.3).
7. Insert the jobs and the machines that were deleted in step 5a.
8. Add the small jobs greedily sequentially.
9. Insert the jobs and the machines that were deleted in step 2a.

Note that after each of the steps 3 and 5a the average load of the machines becomes smaller, and therefore after renormalization, there will be no jobs of weight smaller than β. Hence, step 3 should only be done once in order to get jobs of weight between β and 1, as required by step 6. Clearly, we have the following theorem:

Theorem 3.2 *If there exists an algorithm that provides an assignment which is globally $(1+\beta)$-balanced for an input which consists of jobs of weight between β and 1 (where 1 is the average load of the machines), then there is an algorithm that provides an assignment which is globally $(1+\beta)$-balanced for any input.*

Proof: Use the above algorithm, where you plug into step 6 the algorithm that handles jobs of weight between β and 1. From Claim 3.1 and Theorem 3.1, this algorithm provides an assignment which is globally $(1+\beta)$-balanced. ■

Remark: Actually step 2a of the above algorithm may be omitted since Theorem 3.1 holds even if there are jobs of weight larger than 1. However, it is more natural to keep step 2a, since the assignments done in step 2a are part of every reasonable assignment.

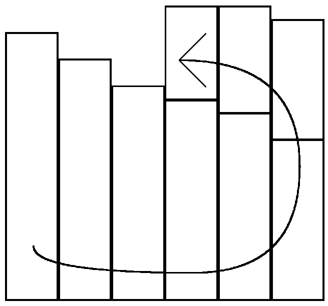
3.3 Handling Big Jobs

In this subsection we will show how to handle jobs of weight between β and 1 (step 6 of the all-norm algorithm).

3.3.1 Treating a small number of big jobs

At first we show a specific case where the number of big jobs is at most $2m$. Since all jobs are of weight at most 1, we have at least m jobs. Let the number of jobs be $2m-k$, for some $0 \le k \le m$.

Definition 3.2 *Given $2m-k$ jobs (for a given k, $0 \le k \le m$) of arbitrary weight (not necessarily at least β), "the snake assignment" assigns each of the k largest jobs to a separate machine, and for every i ($1 \le i \le m-k$) assigns the $i+k$'th largest job and the i'th smallest job to a separate machine.*



The snake assignment sorts the jobs in a non-increasing order of weights, then assigns each job to a machine from the first machine to the last, and then backwards.

Lemma 3.2 *For the case where there are at most $2m$ jobs, "the snake assignment" S, is 1-submajorized by any assignment which does not assign more than two jobs to each machine, and therefore is optimal in all norms.*

Proof: We will first show that it is enough to prove the lemma for the case where the number of jobs is exactly $2m$. In the general case there are $2m-l$ jobs, where $1 \le l \le m$. Any instance of $2m-l$ jobs can be transformed into an instance of $2m$ jobs by adding l zero weighted jobs without changing the loads (the lemma holds for arbitrary weights, including zero). Moreover, this does not affect the snake assignment.

Consider the case where the number of jobs is exactly $2m$. Here all the jobs are divided into pairs, and each pair is assigned to a machine separately. The proof is by induction on m. For $m=1$ the claim is trivial. We will assume that the snake assignment is optimal for $m=k$ and prove it for $m=k+1$. Let R be an arbitrary assignment of these $2(k+1)$ jobs

to $k+1$ machines by pairs. We want to prove that $S <_1 R$. We will build an intermediate assignment I_R and prove $S <_1 I_R <_1 R$.
If in R the biggest job is assigned to the same machine as the smallest job, we will define I_R to be R. Otherwise, we will look at the biggest job, whose weight is denoted by w_1, and assume that in R it is assigned to a machine denoted by A with a job whose weight is denoted by w_2. The smallest job, whose weight is denoted by w_3, is assigned in R to a machine denoted by B with a job whose weight is denoted by w_4. In I_R, the job of weight w_1 will be assigned to A with the job of weight w_3 and the job of weight w_2 will be assigned to B with the job of weight w_4. The assignments to the other machines will be left unchanged. Let P_1 be the assignment I_R on $A \cup B$, and let P_2 be the assignment I_R on the other machines. Let Q_1 be the assignment R on $A \cup B$, and let Q_2 be the assignment R on the other machines. Since $w_1 + w_3 \leq w_1 + w_2$ and $w_2 + w_4 \leq w_1 + w_2$, we have $P_1 <_1 Q_1$. Clearly, $P_2 <_1 Q_2$, and by Lemma 2.1 we conclude that $I_R <_1 R$.
Next we show that $S <_1 I_R$. In I_R the biggest job is assigned to the same machine as the smallest. The assignment to this machine will be denoted by Q_1. The other jobs are assigned to the other machines arbitrarily. The assignment to the other k machines will be denoted by Q_2 . In S, the biggest job is assigned to the same machine as the smallest. The assignment to this machine will be denoted by P_1. The other jobs are assigned to the other machines by the snake assignment. We will denote the assignment of S to the other k machines by P_2. Clearly, $P_1 <_1 Q_1$. From the induction assumption on k machines, $P_2 <_1 Q_2$, and from Lemma 2.1, $S <_1 I_R$, and therefore $S <_1 R$. This concludes the case for exactly $2m$ jobs. ■

Theorem 3.3 *The snake assignment S is globally $\frac{4}{3}$-balanced when there are no more than $2m$ jobs, each of weight between $\frac{1}{3}$ and 1.*

Proof: Let P be another reasonable assignment. We need to prove that $S <_{\frac{4}{3}} P$. We will use an intermediate assignment I_P, and prove that $S <_1 I_P <_{\frac{4}{3}} P$. At first we notice that in P there is no machine with more than three jobs assigned to it (since P is reasonable and the weights of the jobs are at least $\frac{1}{3}$). The algorithm to create the intermediate assignment is defined by:

1. Initialize $i := 1$.
2. While there is a machine with three jobs assigned to it do:
   (a) Find such a machine, denoted by A_i.
   (b) Find another machine, denoted by B_i, with only one job assigned to it (clearly there is such a machine).
   (c) Take the smallest job from A_i, whose weight is denoted by x_i, and assign it to B_i.
   (d) $i := i + 1$.

The output of this algorithm is the intermediate assignment I_P. Note that all the A_i's and B_i's are distinct, and therefore the process stops after at most m steps.

From the previous lemma it is clear that $S <_1 I_P$, since S is optimal among all assignments that do not assign more than two jobs to a machine.

We only have to prove now that $I_P <_{\frac{4}{3}} P$. This will be done by observing step i of the algorithm for creating I_P. Let P_i be the assignment P on $A_i \cup B_i$ and let I_{Pi} be the assignment I_P on $A_i \cup B_i$. We want to prove that $I_{Pi} <_{\frac{4}{3}} P_i$. Of course, once the sum of the loads in both assignments is the same, we only have to compare the machine whose load is the bigger of the two. In P_i the machine with the biggest load is the one with the three jobs (since all the weight are between $\frac{1}{3}$ and 1). If the most loaded machine in I_{Pi} is the one that had the three jobs before the transformation, then $I_{Pi} <_1 P_i$. Otherwise, the most loaded machine in I_{Pi} is the one that had one job assigned to it before the transformation. Consider the job of weight x_i. In P_i it was assigned to a machine whose load was no less than $3x_i$. In I_{Pi} it is assigned to a machine whose load is no more than $1 + x_i$. The ratio between the loads of these machines is $\frac{1+x_i}{3x_i}$ (which is not more than $\frac{4}{3}$, since $x_i \geq \frac{1}{3}$). Then $I_{Pi} <_{\frac{4}{3}} P_i$, and this is true for every A_i,B_i.
Suppose the algorithm stopped after k iterations, then by Lemma 2.2, since all A_i's and B_i's are distinct, $\cup_{i=1}^{k} I_{Pi} <_{\frac{4}{3}} \cup_{i=1}^{k} P_i$. Denote the set of machines not changed by the algorithm by J. Clearly, P and I_P are the same on J. Let P_J be the assignment P on J, and let I_{P_J} be the assignment I_P on J. Then $I_{P_J} <_1 P_J$. From Lemma 2.1:

$$I_P = \left(\cup_{i=1}^{k} I_{Pi}\right) \cup I_{P_J} <_{\frac{4}{3}} \left(\cup_{i=1}^{k} P_i\right) \cup P_J = P \ . \tag{9}$$

Therefore, $S <_{\frac{4}{3}} P$. ■

3.3.2 Treating big jobs - the general case

Recall that all jobs are between β and 1, and assume that $\frac{1}{3} \leq \beta < \frac{1}{2}$. Then no reasonable assignment has more than three jobs on one machine. We may also assume that there are more than $2m$ jobs. Otherwise, we have an assignment which is globally $\frac{4}{3}$-balanced. So there are $2m + k$ jobs to assign, where $1 \leq k \leq m$.

Claim 3.2 *If three jobs are assigned to one machine in a reasonable assignment, then none of them has a weight bigger than* $1 - \beta$.

Proof: If the claim is not true, then one of the jobs has a weight bigger than $1 - \beta$. So there are two jobs with a total weight bigger than 1, which is not a reasonable assignment. ■

For the same reason the next claim is also true:

Claim 3.3 *If three jobs are assigned to one machine in a reasonable assignment, then at most one job has a weight bigger than* 0.5.

Claim 3.4 *There are at least* $3k$ *jobs of weight smaller than* 0.5.

Proof: Suppose there are less than $3k$ jobs of weight smaller than 0.5. Then there are at least $2m-2k+1$ jobs of weight bigger than 0.5. The other jobs have weight of at least β, and the sum of all weights of all jobs $\geq 0.5(2m - 2k + 1) + \beta(3k - 1) \geq m - k + 0.5 + k - \frac{1}{3} > m$. This is a contradiction to the fact that the sum of all weights is m. ■

The "big jobs" algorithm: (step 6 of the all-norm algorithm)

1. Initialize pool of jobs to include all big jobs.
2. Do k times:
   (a) Take the job of biggest weight smaller than $1-\beta$ from the pool of jobs and assign it to a new machine.
   (b) Take the two jobs of biggest weight smaller than 0.5 from the pool of jobs and assign them both to the machine used in the preceding Step (a).
3. Now assign the remaining $2m-2k$ jobs to the remaining $m-k$ machines by the snake assignment.

Lemma 3.3 *The "big jobs" algorithm provides an assignment which is globally* $max(\frac{2-\beta}{3\beta}, \frac{4}{3})$*-balanced, for* $\frac{1}{3} \le \beta < \frac{1}{2}$.

Proof: As can be seen above, the algorithm takes the largest possible triplets in any reasonable assignment, leaving the small jobs to the pairs. Let P be the assignment created by the algorithm and let P_i $(1 \le i \le k)$ be the assignment to one machine created by the iteration i of the loop in the algorithm. Let P_{rem} be the assignment of the algorithm to the remaining machines.
Every reasonable assignment must have at least k triplets of jobs assigned to one machine. Let Q be any reasonable assignment, and let Q_i $(1 \le i \le k)$ be any k different sub-assignments of Q to one machine, each machine having a triplet assigned to it. Let Q_{rem} be the sub-assignment of Q to the remaining machines. So by comparing each P_i to Q_i, we get a ratio of $\frac{0.5+0.5+(1-\beta)}{\beta+\beta+\beta} = \frac{2-\beta}{3\beta}$. By Lemma 2.2, we get $\cup_{i=1}^{k} P_i <_{\frac{2-\beta}{3\beta}} \cup_{i=1}^{k} Q_i$. If we now compare the rest of the $m-k$ machines we will see that in our assignment these machines include the smallest jobs possible (since the largest possible jobs went to the triplets). In any other reasonable assignment these $m-k$ machines will include these jobs or bigger ones (bigger in the sense of comparing the values in the vector of the sorted weights one by one). Even if we assume that the other assignment has the same jobs as ours on these machines (the worst case), then we have the same $2m-2k$ jobs on $m-k$ machines. By Theorem 3.3, the snake assignment is globally $\frac{4}{3}$-balanced, and therefore $P_{rem} <_{\frac{4}{3}} Q_{rem}$.
Now by Lemma 2.1 we get:

$$P = \left(\cup_{i=1}^{k} P_i\right) \cup P_{rem} <_{max(\frac{2-\beta}{3\beta}, \frac{4}{3})} \left(\cup_{i=1}^{k} Q_i\right) \cup Q_{rem} = Q \ . \tag{10}$$

■

Finally , we conclude our main result.

Theorem 3.4 *The all-norm algorithm produces an assignment which is globally* $1+\beta$*-balanced, for* $\beta = \frac{\sqrt{10}-2}{3}$.

Proof: By Lemma 3.3 we have an algorithm that assigns jobs of weight between β and 1 (where 1 is the average load of the machines), and provides an assignment which is globally $max(\frac{2-\beta}{3\beta}, \frac{4}{3})$-balanced, for $\frac{1}{3} \leq \beta < \frac{1}{2}$. We can plug in the "big jobs" algorithm into step 6 of the all-norm algorithm, and by Theorem 3.2 we can return to the original input, and get an assignment which is globally $max(\frac{2-\beta}{3\beta}, \frac{4}{3}, 1+\beta)$-balanced.

If we choose $\frac{2-\beta}{3\beta} = 1+\beta$ ($\beta = \frac{\sqrt{10}-2}{3} \approx 0.3875$), we get an algorithm that provides an assignment which is globally-1.3875 balanced. In particular, this algorithm has an approximation ratio of 1.3875 for all ℓ_p norms simultaneously. ■

References

[1] N. Alon, Y. Azar, G. Woeginger, and T. Yadid. Approximation schemes for scheduling. In *Proc. 8th ACM-SIAM Symp. on Discrete Algorithms*, pages 493–500, 1997.

[2] 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.

[3] Y. Azar, L.Epstein, Y. Richter, and G.J. Woeginger. All-norm approximation algorithms. *Proc. of 8th SWAT*, pages 288–297, 2002.

[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] L. Epstein and J. Sgall. Approximation schemes for scheduling on uniformly related and identical parallel machines. In *Proc. 7th Annual European Symposium on Algorithms*, pages 151–162, 1999.

[6] A. Goel and A. Meyerson. Simultaneous optimization via approximate majorization for concave profits or convex costs. 2003. Manuscript.

[7] A. Goel, A. Meyerson, and S. Plotkin. Approximate majorization and fair online load balancing. In *Proc. 12nd ACM-SIAM Symp. on Discrete Algorithms*, pages 384–390, 2001.

[8] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math*, 17:416–429, 1969.

[9] G.H. Hardy, J.E. Littlewood, and G. Polya. Some simple inequalities satisfied by convex functions. *Messenger Math*, 58:145–152, 1929.

[10] D. Hochbaum and D. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM Journal on Computing*, 17(3):539–551, 1988.

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

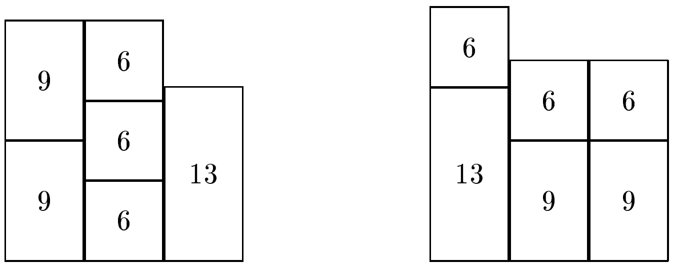
[12] E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling non-identical processors. *Journal of the Association for Computing Machinery*, 23:317–327, 1976.

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

[14] D. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming A*, 62:461–474, 1993. Also in the proceeding of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms, 1993.

A Non-Existence of Approximation Schemes

In this section, we prove that an approximation scheme (not necessarily polynomial time) for finding the optimal solution for two different ℓ_p norms does not exist. In [1] there is an example that shows that optimal solutions for different norms (ℓ_2 and ℓ_∞), might be achieved from different assignments, and therefore not only that we cannot find an assignment that gives an approximation ratio of $1+\varepsilon$, for $\varepsilon > 0$ as small as we want for all norms, but also such an assignment does not exist. We repeat this example in order to prove that there is no such assignment even for any two different norms simultaneously. The example is as follows:



The left assignment is optimal for the ℓ_∞ norm (gives a value of 18, while the right assignment gives a value of 19). The right assignment, on the other hand, is optimal for the ℓ_2 norm (gives a value of $\sqrt{811}$, while the left assignment gives a value of $\sqrt{817}$). So it is clear now that there is no assignment that is $1+\varepsilon$-submajorized by both of these optimal assignments for every ε. But can we find a PTAS for both ℓ_7 and ℓ_3 norms, for example?

Theorem A.1 *For every two norms, ℓ_r and ℓ_q, there exists an input and $\varepsilon > 0$ where there is no assignment that is $1+\varepsilon$-submajorized by both of the optimal assignments for both norms. Hence, there is $\varepsilon > 0$ such that there is no assignment that is globally $(1+\varepsilon)$-balanced.*

Proof: If we look back at the example, we can see that while the left assignment is optimal for some ℓ_{p_1} norm, and the right assignment is optimal for some other ℓ_{p_2} norm, there is no PTAS for both ℓ_{p_1} and ℓ_{p_2} norms. So if $p > 1$ uniquely solves the equation

$$18^p + 18^p + 13^p = 19^p + 15^p + 15^p \ , \tag{11}$$

then there is no approximation scheme for norms from different sides of p ($p = 9.966..$), and an assignment that gives an approximation ratio of $1+\varepsilon$ for any $\varepsilon > 0$ and for two such different norms does not necessarily exist. But this does not mean that we cannot find a PTAS for two different ℓ_p norms, say bigger than 10, or maybe smaller than 9. Consider the equation

$$18^p + 18^p + a^p = (a+6)^p + 15^p + 15^p \tag{12}$$

for $12 \le a \le 15$, which corresponds to the ℓ_p norm of the assignments in the example shown above, when a job of weight a replaces the job of weight 13. We want to prove that for every p this equation has at least one solution a between 12 and 15, and for this a we have two optimal assignments, one for $p' > p$, and one for $p' < p$. This will prove that for every two different norms we can find an input (defined by a), where there are two different optimal assignments for ℓ_r and ℓ_q. For that, we pick p^* between r and q, and solve the value of a from (12) for $p = p^*$.

Solving (12) for a fixed p ($p > 1$) is equivalent to finding the roots of

$$f(a) = 2 * 18^p + a^p - (a+6)^p - 2 * 15^p \ . \tag{13}$$

Note that $f(12) = 18^p + 12^p - 2*15^p > 0$ and $f(15) = 2*18^p - 21^p - 15^p < 0$, and therefore there is a root between 12 and 15 for any $p > 1$.

Claim A.1 *For a fixed a ($12 < a < 15$), the function*

$$g(\boldsymbol{p}) = 2 * 18^p + a^p - (a+6)^p - 2 * 15^p \tag{14}$$

vanishes at most once for $p > 1$. If such a p exists, then it will be denoted by p^*, and $g(p) > 0$ for every $p < p^*$ and $g(p) < 0$ for every $p > p^*$.

Proof: For a fixed a, consider the equation:

$$2 * 18^p + a^p - (a+6)^p - 2 * 15^p = 0 \tag{15}$$

which can be written as:

$$a^p \left(\left(\frac{a+6}{a} \right)^p - 1 \right) = 2 \cdot 15^p \left(\left(\frac{18}{15} \right)^p - 1 \right) \ . \tag{16}$$

By applying ln on both sides of the equation we get

$$p \ln(a) + \ln \left(\left(\frac{a+6}{a} \right)^p - 1 \right) = \ln(2) + p \ln(15) + \ln \left(\left(\frac{18}{15} \right)^p - 1 \right) \tag{17}$$

and then:

$$p(\ln(a) - \ln(15)) - \ln(2) = \ln \left(\left(\frac{18}{15} \right)^p - 1 \right) - \ln \left(\left(\frac{a+6}{a} \right)^p - 1 \right) \ . \tag{18}$$

The left-hand side of the equation is a linear function of p. One can check that the right side of the equation is a concave function for $p \ge 1$. Since $p = 1$ is a solution of the equation, there can be only one more solution bigger than 1. If such a solution p^* exists, then since $g(p) < 0$ when p tends to infinity (since $a > 12$), it is clear (by concavity) that $g(p) > 0$ for every $p < p^*$ and $g(p) < 0$ for every $p > p^*$. ■

So for every p there is an input with three machines and six jobs, where there are two assignments, one of which is optimal for every $\ell_{p'}$ where $p' < p$, and the other, for every $\ell_{p'}$ where $p' > p$. This proves that an assignment which is globally $(1+\varepsilon)$-balanced for two different norms does not necessarily exist for every $\varepsilon > 0$. In particular, there is no approximation scheme for any two different ℓ_p norm problems (even with unlimited computational time). ■

The following theorem generalizes the above theorem for many norms.

Theorem A.2 *Given $k+1$ real numbers, $1 < p_0 < p_1 < p_2 < \ldots < p_k$, there is a set of machines and jobs, such that for every ℓ_{p_i} norm, there is a different optimal assignment.*

Proof: We will pick k numbers q_i, $i = 1 \ldots k$ such that $p_{i-1} < q_i < p_i$. For each of these q_i, we will find an example E_i where there are three machines and six jobs, that has different optimal assignments for ℓ_p norms when $p < q_i$ and for ℓ_p norms when $p > q_i$ (as shown above). If we combine all these examples we get an example of $3k$ machines and $6k$ jobs, which is divided into $3k$ jobs of weight 6, $2k$ jobs of weight 9, and k jobs of different weights a_i, where each a_i is the weight of the job specified for the example E_i, $12 < a_i < 15$. Note that the average load of all the machines is bigger than 15 and smaller than 18. The possible assignments for one machine in any reasonable assignment in this case are given in a decreasing order of loads: (a_i,a_j), $(a_i,9)$, (9,6,6), $(a_i,6)$, (9,9), (6,6,6), (9,6), (a_i), (6,6), (9), (6), (), for some i,j. We will now prove that not all these assignments can be included in some optimal assignment of any norm.
If there is a machine that includes (9,6,6) in some optimal assignment, then there is a machine whose load is smaller than the average. The biggest possible load of such a machine is 15, which includes (9,6). If there is such a machine, then we swap the two jobs between these two machines and create a different assignment, with (9,9) and (6,6,6) assignments on these machines. This assignment is better in all finite norms. For every other load below the average we can find a better assignment in a similar way. This contradicts the optimality of the assignment
If there is a machine that includes (a_i,a_j) in some optimal assignment (for some i,j), then it can be easily shown that there is no machine that does not include some a_k. For example, if (6,6,6) is another machine in this optimal assignment, then switching to $(a_j,6,6)$, $(a_i,6)$ renders a better assignment. If every machine includes some a_i, then there are at least $3k$ jobs whose weight is some a_i. This is a contradiction.
If there is a machine that includes $(a_i,9)$ in some optimal assignment (for some i), then it can be easily shown that there is no machine that includes a job of weight 6, except for machines that include $(a_j,6)$, for some j . For example, $(a_i,9)$ and (6,6,6) cannot be together in an optimal assignment, since $(a_i,6)$ and (9,6,6) render a better assignment in every norm. Therefore, the number of jobs of weight 6, must be at most the number of jobs of weight a_i, for some i . This is a contradiction.
If there is a machine that includes (6,6), then there is no machine that includes $(a_i,6)$, for any i, and there is no machine that includes (9,9) (otherwise the assignment can be improved). Therefore, for every i, a_i is always included by itself on a machine (k machines), and the other $2k$ machines must include a job of weight 9. Therefore, there is no machine left for (6,6). The same proof works for (6).

If there is a machine that includes only (9), then there is no machine that includes (a_i,6), for any i, and there is no machine that includes (6,6,6) (otherwise the assignment can be improved). Therefore, for every i, a_i is always included by itself on a machine (k machines), and the other $2k$ machines must include a job of weight 6. Therefore, there is no machine left for (9).
So now that we know exactly which assignments to a machine are possible for an optimal assignment, we can define for a specific optimal assignment:

$$\begin{aligned}
A &= \text{The number of machines which include } (a_i,6)\text{, for some } i \\
B &= \text{The number of machines which include } (6,6,6) \\
C &= \text{The number of machines which include } (9,9) \\
D &= \text{The number of machines which include } (9,6) \\
E &= \text{The number of machines which include } (a_i)\text{, for some } i\ .
\end{aligned}$$

We know that:

$$\begin{array}{cl}
A+B+C+D+E=3k & \text{(The number of machines)} \\
A+E=k & \text{(The number of } a_i\text{'s, for all } i\text{'s)} \\
2C+D=2k & \text{(The number of 9's)} \\
A+3B+D=3k & \text{(The number of 6's)}\ .
\end{array}$$

From these equations we get $B=C=E$, and $D=2A$. Therefore, an optimal assignment in this case is a union of optimal assignments for each example E_i. The optimal assignment for ℓ_{p_i} takes for every E_j, $j \leq i$, the optimal assignment for $p > q_j$, and for every E_j, $j > i$, the optimal assignment for $p < q_j$. We get $k+1$ different assignments, one for every p_i. ■