All-norm Approximation Algorithms

Yossi Azar * Leah Epstein † Yossi Richter ‡
Gerhard J. Woeginger §

Abstract

A major drawback in optimization problems and in particular in scheduling problems is that for every measure there may be a different optimal solution. In many cases the various measures are different ℓ_p norms. We address this problem by introducing the concept of an *All-norm ρ-approximation algorithm*, which supplies one solution that guarantees ρ-approximation to all ℓ_p norms simultaneously. Specifically, we consider the problem of scheduling in the restricted assignment model, where there are m machines and n jobs, each is associated with a subset of the machines and should be assigned to one of them. Previous work considered approximation algorithms for each norm separately. Lenstra *et al.* [11] showed a 2-approximation algorithm for the problem with respect to the ℓ_∞ norm. For any fixed ℓ_p norm the previously known approximation algorithm has a performance of $\theta(p)$. We provide an all-norm 2-approximation polynomial algorithm for the restricted assignment problem. On the other hand, we show that for any given ℓ_p norm ($p > 1$) there is no PTAS unless P=NP by showing an APX-hardness result. We also show for any given ℓ_p norm a FPTAS for any fixed number of machines.

1 Introduction

1.1 Problem definition

A major drawback in optimization problems and in particular in scheduling problems is that for every measure there may be a different optimal solution. Usually, different algorithms

* `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.

† `lea@idc.ac.il`. School of Computer and Media Sciences, The Interdisciplinary Center, P.O.B 167, 46150 Herzliya, Israel. Research supported in part by the Israel Science Foundation (grant no. 250/01).

‡ `yo@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.

§ `g.j.woeginger@math.utwente.nl`. Department of Mathematics, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands, and Institut für Mathematik, Technische Universität Graz, Steyrergasse 30, A-8010 Graz, Austria. Supported by the START program Y43-MAT of the Austrian Ministry of Science.

are used for diverse measures, each supplying its own solution. Therefore, one may ask what is the "correct" solution for a given scheduling problem. In many cases there is no right answer to this question. We show that in some cases one can provide an appropriate answer, especially when the measures are different ℓ_p norms. Specifically, we address the optimization problem of scheduling in the restricted assignment model. We have m parallel machines and n independent jobs, where job j is associated with a weight w_j and a subset $M(j) \subseteq \{1, \ldots, m\}$ of the m parallel machines and should be assigned to one of them. For a given assignment, the load l_i on a machine i is the sum of weights of the jobs assigned to it. We denote by $\vec{l} = (l_1, \ldots, l_m)$ the machines load vector corresponding to an assignment, and further denote by $\vec{h}$ the vector $\vec{l}$ sorted in non-increasing order. We may use the ℓ_p norm ($p \geq 1$) to measure the quality of an assignment, namely the cost of an assignment is the ℓ_p norm of its corresponding load vector. The ℓ_p norm of a vector $\vec{l}$, denoted $\|\vec{l}\|_p$, is defined by: $\|\vec{l}\|_p = (\sum_{i=1}^{m} l_i^p)^{1/p}$.

Most research done so far in the various scheduling models considered the makespan (ℓ_∞) measure. In some applications other norms may be suitable such as the ℓ_2 norm. Consider for example a case where the weight of a job corresponds to its machine disk access frequency. Then each job 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 squares of the machines loads (namely the ℓ_2 norm of the corresponding machine load vector) whereas the *maximum* delay is proportional to the maximum load.

Simple examples illustrate that for the general restricted assignment problem, an optimal solution for one norm is not necessarily optimal in another norm (and in fact may be very far from being optimal). Given that, one may ask what is the "correct" solution to a scheduling problem. When a solution optimal in all norms exists we would naturally define it as the correct solution and try to obtain it. For the special case of restricted assignment with unit jobs only, Alon *et al.* [1] showed that a *strongly-optimal* assignment that is optimal in all norms exists, and can be found in polynomial time. However, this is not the case in general.

1.2 Our results

All-norm approximation: In light of the above discussion, we introduce the concept of an *All-norm ρ-approximation algorithm*, which supplies one solution guaranteeing simultaneously ρ-approximation with respect to the optimal solutions for all norms. Note that an approximated solution with respect to one norm may not guarantee any constant approximation ratio for any other norm. This does not contradict the fact that there may be a different solution approximating the two norms simultaneously. Simple examples illustrate that we can not hope for an all-norm $(1+\varepsilon)$-approximation for arbitrary ε for this problem (the example in [1] illustrates that ε must be larger than 0.003 even for two norms), hence the best we can hope for (independent of the computational power) is an all-norm ρ-approximation, when ρ is constant. Moreover, from the computational point of view, we can not expect to achieve an all-norm approximation polynomial algorithm with ratio better than 3/2 since Lenstra *et al.* [11] proved a 3/2 lower bound on the approximation ratio of any polynomial algorithm for the makespan alone (assuming $P \neq NP$). Lenstra *et al.* [11] and Shmoys and Tardos [15] presented a 2-approximation algorithm for the makespan, however their algorithm does not guarantee any constant approximation ratio to optimal solutions for any other norms (it is easy to come up with a concrete example to support that). Our main result is an all-norm 2-approximation polynomial algorithm for the restricted assignment model. Our algorithm returns a feasible solution which is at most 2 times the optimal solution for all ℓ_p norms ($p \geq 1$) simultaneously. In contrast, note that for the related machines model and hence for the more general model of unrelated machines, in general there is no assignment obtaining constant approximation ratio for all norms simultaneously (this can be shown by a simple example even when considering only the ℓ_1 and ℓ_∞ norms).

Kleinberg *et al.* [10] and Goel *et al.* [5] considered the problem of fairest bandwidth allocation, where the goal is to maximize the bandwidth allocated to users, in contrast to minimizing the machines loads. In [5] α-balanced assignments are defined, which are similar to our concept of all-norm approximation. However, the algorithm suggested there works only for unit jobs and is $O(\log m)$-competitive. In contrast, our algorithm works for arbitrary size jobs and guarantees constant approximation. We note that the idea of approximating more than one measure appears in [16, 2] where bicriteria approximation for the makespan and the average completion time is provided.

Approximation for any given norm: Recall that for the ℓ_∞ case Lenstra *et al.* [11] presented a 2-approximation algorithm (presented for the more general model of unrelated machines, where each job has an associated m-vector specifying its weight on each machine). For any given ℓ_p norm the only previous approximation algorithm for restricted assignment, presented by Awerbuch *et al.* [3], has a performance of $\theta(p)$ (this algorithm was presented as an on-line algorithm for the unrelated machines model). Note that not only our all-norm 2-approximation algorithm provides 2-approximation to all norms simultaneously, it also improves the previous best approximation algorithm for each fixed ℓ_p norm separately.

Non-approximability for any given norm: Clearly, one may hope to get for any given ℓ_p norm a better approximation ratio (smaller than 2), or even a Polynomial Time Approximation Scheme (PTAS). However, we show that for any given ℓ_p norm ($p > 1$) the problem of scheduling in the restricted assignment model is APX-hard, thus there is no PTAS for the problem unless $P = NP$. Note that for $p = 1$ any assignment is optimal.

Approximation scheme: For any given ℓ_p norm it is impossible to get a PTAS for an arbitrary number of machines. Therefore, the only possible approximation scheme for a given norm is for a fixed number of machines. We present for any given norm a Fully Polynomial Time Approximation Scheme (FPTAS) for any fixed number of machines. Note that for minimizing the makespan Horowitz and Sahni [8] presented a FPTAS for any fixed number of machines. Lenstra *et al.* [11] suggested a PTAS for the same problem (i.e. minimizing the makespan) with better space complexity.

1.3 Techniques and related results

Other related results: Other scheduling models have also been studied. For the identical machines model, where each job has an associated weight and can be assigned to any machine, Hochbaum and Shmoys [7] presented a PTAS for the case of minimizing the makespan. Later, Alon *et al.* [1] showed a PTAS for any ℓ_p norm in the identical machines model. For the related machines model, in which each machine has a speed and the machine load equals the sum of jobs weights assigned to it divided by its speed, Hochbaum and Shmoys [6] presented a PTAS for the case of minimizing the makespan. Epstein and Sgall [4] showed a PTAS for any ℓ_p norm in the same model.

Note that, previous work discussed above showed that PTAS can be achieved for the identical and related machines models when considering the makespan for cost. In contrast, only constant approximation is possible for the restricted assignment and unrelated machines models (see [11]). Our work establishes the same phenomenon for the ℓ_p norm, by proving that only constant approximation can exist for restricted assignment.

Techniques: Our main result, the all-norm 2-approximation algorithm, consists of two phases - finding a strongly-optimal fractional assignment and rounding in to an integral assignment which guarantees 2-approximation to the optimal assignments in all norm simultaneously. The first phase depends on constructing linear programs with exponential number of constraints solved using the ellipsoid algorithm with a supplied oracle. Our algorithm works for the more general model of unrelated machines and finds the lexicographically best (smallest) assignment. Hence, in this sense, it generalizes the algorithm suggested by Megiddo [12, 13], which can be used for the restricted assignment model only. Although the second phase can employ the rounding scheme of [15], our rounding technique, based on eliminating cycles in a bipartite graph, is considerably simpler and more suitable for our needs. Our hardness of approximation result is reduced (by a L-Reduction) from a result by Petrank [14] concerning a variant of 3-Dimensional matching.

Paper structure: In Section 2 we present our approximation algorithm. In section 3 we show the hardness of approximation result for the problem. In section 4 we show for any given ℓ_p norm a FPTAS for any fixed number of machines.

2 All-norm approximation algorithm

We use the notion of a *strongly-optimal assignment* defined in [1] throughout this paper. We repeat the definition in short :

Definition 2.1 Given an assignment H denote by S_k the total load on the k most loaded machines. We say that an assignment is *strongly-optimal* if for any other assignment H' and for all $1 \leq k \leq m$ we have $S_k \leq S'_k$.

A strongly-optimal assignment is optimal in any norm. In the case of unit jobs a strongly-optimal integral assignment exists (and can be found in polynomial time), however this is not the case in general (see [1]). It turns out there always exists a strongly-optimal *fractional* assignment in the general case. Our algorithm works in two stages: in the first stage we find a strongly-optimal fractional assignment and in the second stage we round this fractional assignment to an integral assignment which guarantees 2-approximation with respect to the optimal solutions for all ℓ_p norms.

2.1 Finding a strongly-optimal fractional assignment

The following lemma can be deduced indirectly from general results in [17]. We provide a simple direct proof for it.

Lemma 2.1 For every instance in the restricted assignment model there exists a fractional assignment that is strongly-optimal. In particular, every fractional assignment which induces the lexicographically smallest load vector is a strongly-optimal fractional assignment.

Proof: We restrict ourselves only to rational weights. The lexicographically smallest load vector induced by a fractional assignment (when considering the machines load vector sorted in non-increasing order) is uniquely defined and consists of rational weights (since it is a solution of a set of rational linear equations). Denote such an assignment by H. Assume by contradiction that H is not strongly-optimal, thus there exist a fractional assignment H' and an integer k, $1 \leq k \leq m$, such that $S_k > S'_k$ (we may assume that H' also consists of rational weights by means of limit). We may scale all the weights such that each assigned fraction in H and H' is integral. We may then translate the scaled instance to a new instance with unit jobs only, by viewing a job with associated weight w_j as w_j unit jobs. Clearly, the lexicographically smallest assignment for the new instance is the scaled H and it is also the strongly-optimal assignment (see [1]). However, the scaled H' contradicts this fact. ■

Note that although [1] provides an algorithm to find the strongly-optimal assignment in the unit jobs case which is polynomial in the number of jobs, we can not use it since it is not clear how to choose the units appropriately. Even if such units could be found, translating our original jobs to unit jobs would not necessarily result in a polynomial number of jobs and therefore the algorithm would not be polynomial.

The first stage of our algorithm consists of finding this strongly-optimal assignment. We present a more general algorithm. Our algorithm works for the more general model of unrelated machines and finds the lexicographically smallest fractional assignment (when considering the machines load vector $\vec{h}$ sorted in non-increasing order). In particular, according to lemma 2.1, for the restricted assignment model the lexicographically smallest fractional assignment is the strongly-optimal fractional assignment. In this sense, our algorithm generalizes the algorithm suggested by Megiddo [12, 13], which can be used only for the restricted assignment model.

Theorem 2.1 In the unrelated machines model, the lexicographically smallest fractional assignment can be found in polynomial time.

Proof: We define the following decision problem in the unrelated machines model : given n jobs, where job j is associated with a weight vector $\vec{w}_j$, and $k \leq m$ limits: $S_1 \leq S_2 \leq \ldots \leq S_k$ is there an assignment H such that $\sum_{i=1}^{r} l_i \leq S_r$ $(r = 1, \ldots, k)$ where $\vec{l}$ is the vector of machine loads introduced by H sorted in non-increasing order. We note that the lexicographically smallest prefix vector $\vec{S} = (S_1, \ldots, S_m)$ induces the lexicographically smallest assignment $\vec{h}$ by defining $h_i = S_i - S_{i-1}$ $(S_0 = 0)$. Denote by $M(j)$ $(j = 1, \ldots, n)$ the set of machines to which job j can be assigned, i.e $\forall i \in M(j)$ $w_{ij} < \infty$. For the case of $k = 1$ (i.e. deciding the makespan) the decision problem can be translated to the following linear program:

$$\begin{array}{lll}
\sum_{i=1}^{m} x_{ij} = 1 & \text{for} & j = 1, \ldots, n \\
\sum_{j=1}^{n} x_{ij} w_{ij} \leq S_1 & \text{for} & i = 1, \ldots, m \\
x_{ij} \geq 0 & \text{for} & j = 1, \ldots, n \ , \ i = 1, \ldots, m \\
x_{ij} = 0 & \text{for} & j = 1, \ldots, n \ , \ i \notin M(j) \ ,
\end{array}$$

where x_{ij} denotes the relative fraction of job j placed on machine i. Since we can not identify the machines according to their loads order, the general case is represented by a linear program with number of constraints exponential in m, as follows:

$$\begin{array}{lll}
\sum_{i=1}^{m} x_{ij} = 1 & \text{for} & j = 1, \ldots, n \\
\sum_{j=1}^{n} x_{i_1 j} w_{i_1 j} + \ldots + \sum_{j=1}^{n} x_{i_t j} w_{i_t j} \leq S_t & \forall 1 \leq t \leq k & \forall 1 \leq i_1 < \ldots < i_t \leq m \\
x_{ij} \geq 0 & \text{for} & j = 1, \ldots, n \ , \ i = 1, \ldots, m \\
x_{ij} = 0 & \text{for} & j = 1, \ldots, n \ , \ i \notin M(j) \ .
\end{array}$$

We employ the ellipsoid algorithm to solve this linear program in polynomial time (see [9] for details). In order to use the ellipsoid algorithm we should supply a separation oracle running in polynomial time. We next describe the algorithm we use as the oracle for the general linear program:

1. Given the assignment we construct the corresponding machines load vector.
2. We sort the load vector. Denote by $\vec{h}$ the sorted vector.
3. If there exists r, $1 \leq r \leq k$ such that $\sum_{i=1}^{r} h_i > S_r$ then the algorithm returns 'not feasible' together with the unsatisfied constraint - the one involving the r most loaded machines (whose indices we have).
4. Otherwise the algorithm returns 'feasible'.

Since the sorting operation (step 2) dominates the time complexity of the algorithm, its running time is clearly polynomial. We prove its correctness:

Claim 2.1 The algorithm returns 'feasible' $\Leftrightarrow$ the given assignment is feasible.

Proof: $\Rightarrow$ Suppose on the contrary that the given assignment is not feasible. Then there is an unsatisfied constraint involving $r \leq k$ machines such that their total load is greater than S_r. In particular the constraint involving the r most loaded machines introduced by the given schedule is not satisfied. Since our algorithm checks all the constraints involving the $1 \leq r \leq k$ most loaded machines, it will return 'not feasible'.

$\Leftarrow$ Suppose on the contrary that the algorithm returned 'not feasible'. Thus for some $1 \leq r \leq k$ the total load on the r most loaded machines exceeds S_r, and there is an unsatisfied constraint. Hence the assignment is not feasible. ∎

We use an incremental process to find the lexicographically smallest assignment. Our algorithm has m steps where in step i we determine the total load on the i most loaded machines in the assignment, given the total loads on the k most loaded machines ($1 \leq k \leq i-1$). Each step is done by performing a binary search on the decision problems. Consider the first step for example: we want to establish the load on the most loaded machine. Denote for job j ($j = 1, \ldots, n$) its smallest possible weight by $w_j^{min} = \min_i w_{ij}$. Let $t = \sum_{j=1}^{n} w_j^{min}$. Clearly t is an upper bound on the load of the most loaded machine, and t/m a lower bound. We can perform a binary search on the load of the most loaded machine while starting with $u = t$ (initial upper bound) and $l = t/m$ (initial lower bound). Testing a bound S on the most loaded machine is done by considering the decision problem with the n jobs and limit $S_1 = S$. We can stop the binary search when $u - l < \varepsilon$ and set the load on the most loaded machine to the load obtained from the feasible solution to the linear program. Later we show how to choose ε such that the value produced by the feasible solution is the exact one since there is at most one possible load value in the range $[l, u]$. Given this ε, the number of iterations needed for the binary search to complete is $O(\log(t/\varepsilon))$. In the ith step ($i = 1, \ldots, m$) we perform the binary search on the total load of the i most loaded machines given the total loads on the k most loaded machines ($k = 1, \ldots, i-1$). Denote by $L_1, \ldots, L_{i-1}$ the prefix loads we found. We perform the binary search on the total load of the i most loaded machines starting with $u = L_{i-1} + t,\ l = L_{i-1}$. Testing a bound S is done by considering the decision problem with the n jobs and limits $S_1 = L_1, \ldots, S_{i-1} = L_{i-1}, S_i = S$. Again we stop the binary search when $u - l < \varepsilon$ and set L_i to the total load on the i most loaded machines produced by the feasible assignment we found for the linear program.

We now determine the value of ε. Each feasible solution to the linear problem $\{x_{ij}\}$ can be written as $\left\{\frac{d_{ij}}{d}\right\}$ where d and $\{d_{ij}\}$ are integers smaller than $2^{P(I)}$ for some polynomial P in the size of the input (see [9] for example). If we choose $\varepsilon = 2^{-2P(I)}$ then we are guaranteed that there is only one possible load value in the range $[l, u]$ when $u - l < \varepsilon$ (see [9]). Thus in each step $i = 1, \ldots, m$ the binary search involves $O(P(I) + log \sum_{j=1}^{n} w_j^{min})$ iterations, polynomial in the size of the input. Hence in polynomial time we find the desired lexicographically smallest assignment. ∎

2.2 Rounding the strongly-optimal fractional assignment

We now return to the restricted assignment model. As mentioned above, the algorithm presented in theorem 2.1 finds the strongly-optimal fractional assignment in polynomial time. The second stage of our algorithm consists of rounding the fractional assignment $\{x_{ij}\}$ to an integral assignment for the problem obtaining 2-approximation for every ℓ_p norm measure. We note that although the rounding scheme presented in [15] can be used for this purpose, our rounding technique is considerably simpler and more suitable for our needs.

Theorem 2.2 A strongly-optimal fractional assignment can be rounded in polynomial time to an integral assignment which is at most 2 times the optimal solution for all ℓ_p norms at the same time.

Proof: Given the fractional assignment $\{x_{ij}\}$ we will show how to construct the desired integral assignment $\{\hat{x}_{ij}\}$ in polynomial time. We construct the bipartite graph $G = (U, V, E)$ having $|U| = n$ vertices on one side (representing the jobs) and $|V| = m$ vertices on the other (representing the machines) while $E = \{(i, j)|x_{ij} > 0\}$. At first we would like to eliminate all cycles in G while preserving the same load on all machines. We eliminate the cycles in G in polynomial time by performing the following steps:

1. We define a weight function $W : E \to R^+$ on the edges of G such that $W(i, j) = x_{ij}w_j$, i.e. the actual load of job j that is assigned to machine i.
2. As long as there are cycles in G, find a cycle, and determine the edge with the smallest weight on the cycle (denote this edge by e and its weight by t).
3. Starting from e subtract t and add t from the weights on alternating edges on the cycle, and remove from G the edges with weight 0. See Figure 1 for an example.

It is clear that this method eliminates the cycles one by one (by discarding the edge with the smallest weight on each cycle) while preserving the original load on all machines. Denote by G the new graph obtained after eliminating the cycles and by $\{x_{ij}\}$ the new strongly-optimal fractional assignment represented by G (which is a forest). In the first rounding phase consider each integral assignment $x_{ij} = 1$, set $\hat{x}_{ij} = 1$ and discard the corresponding edge from the graph. Denote again by G the resulting graph.

In the second rounding phase we assign all the remaining fractional jobs. For this end we construct a matching in G that covers all job nodes by using the same method presented in [11]. We consider each connected component in G, which is a tree, and root that tree in one of the job nodes. Match each job node with any one of its children. Since every node in the tree has at most one father we get a matching and since each job node is not a leaf (each job node has a degree at least 2) the resulting matching covers all job nodes. For each edge (i, j) in the matching set $\hat{x}_{ij} = 1$.

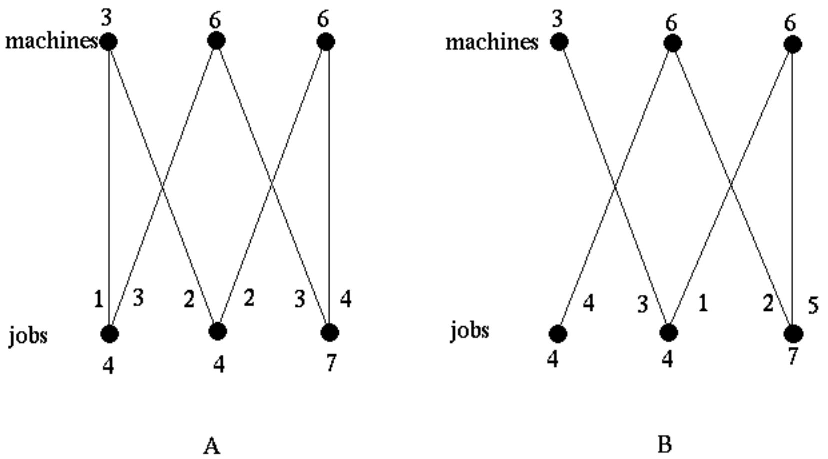

Figure 1: Eliminating the cycle. Edge and job weights and machine loads are listed. Figure A - before eliminating the cycle. Figure B - after eliminating the cycle.

We now prove that the schedule obtained from the assignment $\{\hat{x}_{ij}\}$ guarantees a 2-approximation to the optimal solutions for all ℓ_p norms (for $p \geq 1$). Fix p and denote by OPT the optimal solution for the problem using ℓ_p for cost. Denote by H^{opt} the strongly-optimal fractional schedule obtained after eliminating the cycles and denote by H the schedule returned by the algorithm. Further denote by H_1 the schedule consisting of the jobs assigned in the first rounding phase (right after eliminating the cycles) and by H_2 the schedule consisting of the jobs assigned in the second rounding phase (those assigned by the matching process). We have :

$$\|H_1\|_p \leq \|H^{opt}\|_p \leq \|OPT\|_p \ ,$$

where the first inequality follows from the fact that H_1 is a sub-schedule of H^{opt} and the second inequality results from H^{opt} being a strongly-optimal fractional schedule thus optimal in any ℓ_p norm compared with any other fractional schedule, and certainly optimal compared with OPT which is an integral schedule. We also know that:

$$\|H_2\|_p \leq \|OPT\|_p \ ,$$

where the inequality results from the fact that H_2 schedules only *one job per machine* thus optimal integral assignment in any ℓ_p norm for the subset of jobs it assigns and certainly has cost smaller than any integral assignment for the whole set of jobs. We can now show :

$$\|H\|_p = \|H_1 + H_2\|_p \leq \|H_1\|_p + \|H_2\|_p \leq \|OPT\|_p + \|OPT\|_p = 2\|OPT\|_p \ ,$$

which concludes the proof that the schedule H we constructed guarantees a 2-approximation to optimal solutions for all ℓ_p norms and can be found in polynomial time. ∎

3 APX-hardness for an arbitrary number of machines

In this section we describe an L-reduction from the APX-hard Maximum Bounded 3-Dimensional Matching problem (Max-3DM) to the minimization of sum of squared machine loads for the restricted assignment problem. This clearly implies APX-hardness of ℓ_2-norm minimization for restricted assignment (since a PTAS for approximating $\sqrt{x}$ yields a PTAS for approximating x). The proof can be easily modified and extended to the other ℓ_p-norms with $p > 1$. Our construction draws some ideas from Lenstra, Shmoys & Tardos [11]. The problem Max-3DM is defined as follows.

> **Instance:** Three sets $A = \{a_1, \ldots, a_q\}$, $B = \{b_1, \ldots, b_q\}$, and $C = \{c_1, \ldots, c_q\}$, together with a subset T of $A \times B \times C$. Any element in A, B, C occurs in one, two, or three triples in T; note that this implies $q \leq |T| \leq 3q$.
>
> **Goal:** Find a subset T' of T of maximum cardinality such that no two triples of T' agree in any coordinate.
>
> **Measure:** The measure of a feasible solution T' is the cardinality of T'.

Petrank [14] has shown that Max-3DM is APX-hard even if one only allows instances where the optimal solution consists of $q = |A| = |B| = |C|$ triples; in the following we will only consider this additionally restricted version of Max-3DM.

For the L-reduction we specify a function R that maps instances I of Max-3DM into scheduling instances $R(I)$, and a function S that maps feasible solutions of $R(I)$ back into feasible solutions of I. Given any instance I of Max-3DM, the instance $R(I)$ contains $3q$ machines.

- For every triple T_i in T, there is a corresponding triple machine $M(T_i)$.
- Moreover, there are $3q - |T|$ so-called dummy machines.

The instance $R(I)$ contains $5q$ jobs.

- For every a_j, b_j, and c_j ($j = 1, \ldots, q$) there are corresponding element jobs $J(a_j)$, $J(b_j)$, and $J(c_j)$. An element job cannot be assigned to dummy machines; an element job can only be assigned to a triple machine $M(T_i)$ if its underlying element is contained in the triple T_i. Every element job has processing time 1.
- Moreover there are $2q$ so-called dummy jobs. Dummy jobs have processing time 3 on all machines.

This completes the description of the scheduling instance $R(I)$. Since we only consider instances of Max-3DM where the optimal solution consists of q triples, we have $\text{Opt}(I) = q$. Now consider the following schedule for instance $R(I)$: For each triple $T_i = (a_j, b_k, c_l)$ in the optimal solution to I, we schedule the three element jobs $J(a_j)$, $J(b_k)$, and $J(c_l)$ on machine $M(T_i)$. The $2q$ dummy jobs are assigned to the remaining $2q$ empty machines so that each machine receives exactly one dummy job. In the resulting schedule every machine has load 3, and hence the objective value of this schedule is $27q$. Therefore, $\textsc{Opt}(R(I)) \le 27q = 27\,\textsc{Opt}(I)$ and the first condition on L-reductions is satisfied with $\alpha = 27$.

Next we specify the function S. Let s be a feasible schedule for a scheduling instance $R(I)$. A machine $M(T_i)$ in the schedule s is called *good*, if it processes three jobs of length 1. Note that these three jobs can only be the jobs $J(a_j)$, $J(b_k)$, and $J(c_l)$ with $T_i = (a_j, b_k, c_l)$. We define the feasible solution $S(s)$ for the instance I of Max-3DM to consist of all triples T_i for which the machine $M(T_i)$ is good.

Consider a feasible schedule s for an instance $R(I)$ of the scheduling problem. For $k = 0, 1, 2, 3$ let m_k denote the number of machines in schedule s that process exactly k jobs of length 1. Then the total number of machines equals

$$m_0 + m_1 + m_2 + m_3 \;=\; 3q, \qquad (1)$$

and the total number of processed element jobs of length 1 equals

$$m_1 + 2m_2 + 3m_3 \;=\; 3q. \qquad (2)$$

Note that by our definition of the function S, the objective value $c(S(s))$ of the feasible solution $S(s)$ equals m_3. In Lemma 3.1 we will prove that $c(s) \ge 29q - 2m_3$ holds. Altogether, this then yields that

$$|c(S(s)) - \textsc{Opt}(I)| \;=\; q - m_3 \;=\; \frac{1}{2}(29q - 2m_3 - 27q) \;\le\; \frac{1}{2}|c(s) - \textsc{Opt}(R(I))|,$$

and that the second condition on L-reductions is satisfied with $\beta = 1/2$. Since the functions R and S are computable in polynomial time, we have established all necessary properties of an L-reduction. Hence, minimizing the sum of squared machine loads for the restricted assignment problem indeed is an APX-hard problem.

Lemma 3.1 The objective value $c(s)$ of the feasible solution s of the scheduling instance $R(I)$ satisfies $c(s) \ge 29q - 2m_3$.

Proof: Let us remove all dummy jobs from schedule s and then add them again in the cheapest possible way, such that the resulting new schedule s' has the smallest possible objective value that can be reached by this procedure. Since $c(s) \ge c(s')$, it will be sufficient to establish the inequality $c(s') \ge 29q - 2m_3$. What is the cheapest way of adding the $2q$ dummy jobs of length 3 to m_0 empty machines, to m_1 machines with load 1, to m_2 machines with load 2, and to m_3 machines with load 3? Each machine should receive at most one dummy job, and the dummy jobs should be added to the machines with the smallest loads. The inequality (2) implies $m_3 \leq q$, and then (1) yields $m_0 + m_1 + m_2 \geq 2q$. Hence, the m_3 machines of load 3 will not receive any dummy job. The inequality (2) implies $m_1 + m_2 + m_3 \geq q$, and then (1) yields $m_0 \leq 2q$. Hence, the m_0 empty machines all will receive a dummy job. For the rest of the argument we will distinguish two cases.

In the first case we assume that $m_0 + m_1 \geq 2q$. In this case there is sufficient space to accommodate all dummy jobs on the machines with load at most 1. Then schedule s' will have $m_0 + m_3$ machines of load 3, m_2 machines of load 2, $m_0 + m_1 - 2q$ machines of load 1, and $2q - m_0$ machines of load 4. From (1) and (2) we get that $m_0 = m_2 + 2m_3$ and that $m_1 = 3q - 2m_2 - 3m_3$. Moreover, our assumption $m_0 + m_1 \geq 2q$ is equivalent to $m_2 + m_3 - q \leq 0$. We conclude that

$$\begin{aligned} c(s') &\geq 9(m_2 + 3m_3) + 4m_2 + (q - m_2 - m_3) + 16(2q - m_2 - 2m_3) \\ &= 33q - 4m_2 - 6m_3 \;\geq\; 33q - 4m_2 - 6m_3 + 4(m_2 + m_3 - q) \;=\; 29q - 2m_3. \end{aligned}$$

In the second case we assume that $m_0 + m_1 < 2q$. In this case there is not sufficient space to accommodate all dummy jobs on the machines with load at most 1, and some machines with load 2 must be used. Then schedule s' will have $m_0 + m_3$ machines of load 3, m_1 machines of load 4, $2q - m_0 - m_1$ machines of load 5, and $m_0 + m_1 + m_2 - 2q$ machines of load 2. As in the first case we use $m_0 = m_2 + 2m_3$ and $m_1 = 3q - 2m_2 - 3m_3$. Our assumption $m_0 + m_1 < 2q$ is equivalent to $q - m_2 - m_3 < 0$. We conclude that

$$\begin{aligned} c(s') &\geq 9(m_2 + 3m_3) + 16(3q - 2m_2 - 3m_3) + 25(m_2 + m_3 - q) + 4(q - m_3) \\ &= 27q + 2m_2 \;>\; 27q + 2m_2 + 2(q - m_2 - m_3) \;=\; 29q - 2m_3. \end{aligned}$$

This completes the proof of the lemma. ■

4 FPTAS for any fixed number of machines and a given ℓ_p norm

For a given ℓ_p norm and any fixed number of machines we describe a FPTAS for the restricted assignment problem, i.e. a $(1+\varepsilon)$-approximation algorithm for any $\varepsilon > 0$ running in time polynomial in n and $1/\varepsilon$. Recall that there is no approximation scheme supplying the same solution for all ℓ_p norms since the optimal solutions for different norms can vary significantly. By the hardness of approximation result we showed, there is no approximation scheme (PTAS or FPTAS) for a given norm and any number of machines unless P=NP. Hence the only possible approximation scheme is for a given norm and any fixed number of machines. Our FPTAS is a modification of the method presented initially by Horowitz and Sahni in [8]. Our algorithm works for all scheduling models: identical, related, restricted assignment and unrelated machines, and is therefore presented in the most general model, i.e unrelated machines. For any ε our algorithm A_ε consists of the following steps:

1. Given the jobs weights $\{w_{ij}\}$, we denote for each job its smallest possible weight by $\bar{w}_j = \min_i w_{ij}$. Given that there is a feasible assignment placing each job on the machine where its weight is minimal, we know that in any optimal assignment the load on each machine is at most $\sum_{j=1}^{n} \bar{w}_j$. For this reason we can replace all weights $w_{ij} > \sum_{j=1}^{n} \bar{w}_j$ by ∞, since no optimal assignment will ever use them. Denote by l^{opt} the machines load vector corresponding to the optimal assignment. By the convexity of the norm function we get that: $\|l^{opt}\|_p \geq (\sum_{j=1}^{n} \bar{w}_j)/m \cdot m^{1/p}$. Assume for simplicity of notation that: $(\sum_{j=1}^{n} \bar{w}_j)/m = 1$, hence $\|l^{opt}\|_p \geq m^{1/p}$ and the maximum load on any machine in any optimal assignment is at most m. We divide the interval $[1, \ldots, m]$ into m/δ equal parts of size δ each (where δ is a function of ε chosen later) and round each weight w_{ij} to $w'_{ij} = k\delta$ for the maximal $k \geq 0$ such that $w'_{ij} \leq w_{ij}$.

2. Using dynamic programming we would like to find all possible load vectors corresponding to legal assignments. We define the following states for the jth layer ($j = 1, \ldots, n$):

$$T_j(l_1, \ldots, l_m) \quad l_i = k \cdot \delta \, , \, k = 0, \ldots, m/\delta \, ,$$

where $T_j(l_1, \ldots, l_m) = 1$ if and only if the load vector $(l_1, \ldots, l_m)$ corresponds to any legal assignment of the first j jobs ($T_j(l_1, \ldots, l_m) = 0$ otherwise). The dynamic program computes each value in the following way:

$$T_j(l_1, \ldots, l_m) = \bigvee_{i=1}^{m} T_{j-1}(l_1, \ldots, l_i - w'_{ij}, \ldots, l_m) \, .$$

For each $T_j(l_{i_1}, \ldots, l_{i_m}) = 1$ we can store the assignment of the jth job, thus for any legal load vector we can trace back the corresponding assignment (one of the possible corresponding assignments, to be accurate).

3. After the completion of the dynamic program we choose among all possible load vectors (all load vectors $(l_{i_1}, \ldots, l_{i_m})$ for which $T_n(l_{i_1}, \ldots, l_{i_m}) = 1$) the one obtaining the minimal value for the given norm. We return the assignment corresponding to this load vector. The real cost corresponding to the returned assignment is obtained by considering the ℓ_p norm of the load vector when substituting the rounded weights with the original ones.

Denote by l^A the load vector corresponding to the assignment returned by the algorithm with the original job weights and by l'^A the load vector corresponding to the assignment with the rounded weights. Analogously denote by l^{opt} and l'^{opt} the optimal assignment with the original and rounded weights, respectively. We first prove that the suggested algorithm returns an assignment which guarantees $(1+\varepsilon)$-approximation to the optimal solution.

Lemma 4.1 For any $\varepsilon > 0$ choosing $\delta = \varepsilon/n$ for the algorithm yields: $\frac{\|l^A\|_p - \|l^{opt}\|_p}{\|l^{opt}\|_p} \leq \varepsilon$

Proof:

$$\|l^A\|_p \quad \leq \quad \|l'^A + \delta n \cdot \vec{1}\|_p$$

$$\begin{aligned}
&\leq \|l'^A\|_p + \delta n \cdot m^{1/p} \\
&\leq \|l'^{opt}\|_p + \delta n \cdot m^{1/p} \\
&\leq \|l^{opt}\|_p + \delta n \cdot m^{1/p} \ .
\end{aligned}$$

The first inequality follows from the fact that the rounding procedure decreases each job weight by at most δ thus $l_i^A \leq l_i'^A + \delta n$ $(i = 1, \ldots, m)$. The third inequality results from l'^A being optimal for the rounded weights. Recall that $\|l^{opt}\|_p \geq m^{1/p}$, thus:

$$\frac{\|l^A\|_p - \|l^{opt}\|_p}{\|l^{opt}\|_p} \leq \frac{\delta n \cdot m^{1/p}}{m^{1/p}} \ .$$

By the choice $\delta = \varepsilon/n$ we get: $\frac{\|l^A\|_p - \|l^{opt}\|_p}{\|l^{opt}\|_p} \leq \varepsilon$, as required. ■

We now analyze the algorithm time complexity. There are n layers (n jobs) in the dynamic program and the number of states in each layer is $(m/\delta)^m$ since there are m machines and each machine load has m/δ possibilities. Calculating the value for a certain state requires looking at the values of at most m other states. Hence the algorithm time complexity is: $O\left(mn\left(\frac{m}{\delta}\right)^m\right)$. By substituting δ with its chosen value the complexity is: $O(mn(\frac{mn}{\varepsilon})^m)$, which is polynomial in n and $1/\varepsilon$. Hence the family of algorithms A_ε is a FPTAS.

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] J. Aslam, A. Rasala, C. Stein, and N. Young. Improved bicriteria existence theorems for scheduling. In *Proc. 10th ACM-SIAM Symp. on Discrete Algorithms*, pages 846–847, 1999.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] H. Karloff. *Linear Programming.* Birkhäuser, Boston, 1991.

[10] J. Kleinberg, Y. Rabani, and E. Tardos. Fairness in routing and load balancing. *J. Comput. System Sci.*, 63(1):2–20, 2001.

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

[12] N. Megiddo. Optinal flows in networks with multiple sources and sinks. *Mathematical Programming*, 7:97–107, 1974.

[13] N. Megiddo. A good algorithm for lexicographically optimal flows in multi-terminal networks. *Bulletin of the American Mathematical Society*, 83(3):407–409, 1977.

[14] E. Petrank. The hardness of approximation: Gap location. In *Computational Complexity 4*, pages 133–157, 1994.

[15] 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.

[16] C. Stein and J. Wein. On the existence of schedules that are near-optimal for both makespan and total weighted completion time. *Operations Research Letters*, 21, 1997.

[17] A. Tamir. Least majorized elements and generalized polymatroids. *Mathematics of Operations Research*, 20(3):583–589, 1995.