Preprint, Institute of Mathematics, AS CR, Prague. 2007-12-10

Graph Balancing: A Special Case of Scheduling Unrelated Parallel Machines

Tomáš Ebenlendr* Marek Krčál* Jiří Sgall*

Abstract

We design a 1.75-approximation algorithm for a special case of scheduling parallel machines to minimize the makespan, namely the case where each job can be assigned to at most two machines, with the same processing time on either machine. (This is a special case of so-called restricted assignment, where the set of eligible machines can be arbitrary for each job.) We also show that even for this special case it is NP-hard to compute better than 1.5 approximation.

This is the first improvement of the approximation ratio 2 of Lenstra, Shmoys, and Tardos [Approximation algorithms for scheduling unrelated parallel machines, *Math. Program.*, 46:259–271, 1990], for any special case with unbounded number of machines. Our lower bound yields the same ratio as their lower bound which works for restricted assignment, and which is still the state-of-the-art lower bound even for the most general case.

1 Introduction

Graph balancing. Suppose we are given an undirected multigraph (i.e., there may be multiple edges connecting any two vertices and also loops) with weights on the edges. We are asked to orient the edges so that the load of each vertex is small, where the load is the sum of the weights of the incoming edges. More exactly, our objective is to minimize the maximum of the loads of all vertices. We call this problem Graph Balancing.

It is obvious that Graph Balancing is NP-hard: Already if the graph contains only two vertices and parallel edges, an exact solution would solve Subset Sum, one of the basic NP-complete problems.

Thus the main question, and the topic of the paper is: How well is it possible to approximate Graph Balancing?

Previous work and motivation. As a motivating example, consider an airline company that runs many connections between many airports. The company management wants to (re)assign a home airport for every connection (naturally there are always two possibilities). Each connection creates certain load on the airport, which may be different for different planes (flights). The goal is to balance the load among the airports, more exactly to minimize the maximal load among the airports.

Our main motivation is a classical problem of *scheduling unrelated parallel machines* to minimize makespan. In this problem, we are given an $n \times m$ matrix of non-negative numbers. Its entry p_{ji} denotes the amount of time which machine i needs to process job j. We also allow $p_{ji} = \infty$ as a shortcut denoting that job j cannot be processed by machine i. (∞ may be replaced by a sufficiently large number, of course.) The output is a schedule which is described by an assignment of the n jobs to the m machines. The objective is to minimize the makespan, i.e., the length of the schedule, defined as the maximum among all machines i of the sum of p_{ji} over all jobs j assigned to i.

A special case considered in literature is that of *restricted assignment*. Here we require that in each row j all entries are either ∞ or equal to the same value p_j. Thus each job has some fixed processing time, but it is also restricted to be scheduled on some machine from a given subset. In this context, the vertices in an instance of Graph Balancing correspond to machines and the edges correspond to jobs that can be assigned to one of their endpoints. Thus Graph Balancing corresponds to the special case of scheduling with restricted assignment where each job can be scheduled on one of at most two machines.

Lenstra *et al.* [9] gave a beautiful 2-approximation algorithm based on linear programming for the general problem and proved that approximating it with ratio better than 1.5 is NP-hard, even for restricted assignment. The problem of finding a better than 2-approximation algorithm (or improve the lower bound) is one of the most prominent open problems in the area of approximation algorithms for scheduling [11], it is also covered in textbooks, e.g. [13]. Despite of that and all the research related to this problem, there have been

*Inst. of Mathematics, AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic, `ebik@math.cas.cz`, `marek.krcal@seznam.cz`, `sgall@math.cas.cz`. Partially supported by Institutional Research Plan No. AV0Z10190503, by Inst. for Theor. Comp. Sci., Prague (project 1M0545 of MŠMT ČR), grant IAA1019401 of GA AV ČR, and grant 201/05/0124 of GA ČR.

no improvement of the bounds for an unbounded number of machines, not even in any special case like restricted assignment.

Our results. Our main result is an 1.75-approximation algorithm for Graph Balancing, see Section 3. Similarly as Lenstra *et al.* [9], we use an integer programming formulation of the problem and its linear relaxation as a lower bound on the optimum. However, even in our special case, the integrality gap of the formulation from [9] is 2, and thus it is not sufficient to use their linear program with perhaps a more careful rounding. We enhance the integer program by introducing new constraints. This in turn forces us to design the rounding procedure much more carefully. In addition, in our relaxation, there are possibly exponentially many new constraints, and we need to do some additional work to solve this linear program.

We also show that it is NP-hard to approximate Graph Balancing with approximation ratio smaller than 1.5, see Section 4. This matches the state-of-the-art lower bound for the general problem of scheduling on unrelated machines. The lower bound of 1.5 from Lenstra *et al.* [9] for scheduling of unrelated machines does not apply to our problem, as it uses jobs that can be assigned to many machines, not only two. Our proof uses a direct reduction from (a variant of) 3-SAT. In our opinion, it is actually even simpler than the original proof which reduces from 3-dimensional matching [9].

Our results for Graph Balancing are the first non-trivial improvement of an approximation factor for any special case of scheduling on unrelated machines after almost 20 years since the work of Lenstra *et al.* [9]. Even though our special case is quite restricted, we find the problem interesting on its own. Also our lower bound shows that the restricted problem is still hard.

Other related results. Another prominent special case of the scheduling problem is that of uniformly related machines. Here each machine has a speed s_i and p_{ji} is given as p_j/s_i (in other words, the input matrix has rank 1). In this case the problem becomes significantly easier and a polynomial time approximation scheme is known [5].

For the case of unrelated machines, polynomial time approximation schemes are known for a fixed number of machines [6, 7]. For an unbounded number of machines, the 2-approximation algorithm can be actually slightly improved to $2-1/m$, see [12]. Another line of research is to obtain a 2-approximation without solving a linear program using flow algorithms, see [4] and references therein. There is also a large amount of work on heuristics with no guarantee on the worst case approximation ratio.

One approach, recently popular, to attack the bound of 2-approximability of scheduling on unrelated machines, is to study the l_p norm of the vector of machine loads. In this context, makespan corresponds to the l_∞ norm. For any $p < \infty$ it is possible to achieve a better than 2 approximation, and in fact it is possible to achieve this simultaneously for all norms, see [2, 8]. Another objective studied for scheduling unrelated machines is that of maximizing the minimum machine completion time, also known as fair allocation or Santa Claus problem, see [3, 1] for recent interesting approximation algorithms.

2 Preliminaries

2.1 Problem formulation

From now on, we restrict ourselves to the special case of restricted assignment where each job is allowed to be assigned to one of at most two machines. If a job can be assigned to one machine only, its assignment is determined. We can omit all such jobs and replace them by one quantity for each machine, which we call a *dedicated load* of that machine.

We use a weighted multigraph $G = (V, E, \mathbf{p}, \mathbf{q})$ to capture instances of our problem. Vertices V correspond to the machines, edges E to the jobs with a choice of two machines, the weights $p_e \geq 0$, $e \in E$, are their processing times, and finally the weights $q_v \geq 0$, $v \in V$ are the dedicated loads. The desired output is an orientation of edges, which is defined as a mapping $\gamma : E \to V$ such that $\gamma(e)$ is incident with e for each $e \in E$. The load of v is q_v plus the sum of all p_e of edges e oriented towards v, i.e., with $\gamma(e) = v$. We minimize the maximal load.

We consider the set of edges E to be an abstract set, to distinguish the parallel edges. However, we abuse notation, and still use the notation $v \in e$ for "the vertex v is incident to the edge e". Similarly, we slightly abuse "subgraph" and use $H \subseteq G$ to mean H is a multigraph whose edges are some subset of edges of G, with or without weights, sometimes even directed as is clear from context. In particular, if we talk about a directed cycle in G, it has to be a subgraph, oriented by one of two cyclic orientations; this includes also the case of a 2-cycle.

Now the problem is formally stated as follows:

Problem Graph Balancing
Input: A weighted multigraph $G = (V, E, \mathbf{p}, \mathbf{q})$ with edge weights $p_e \geq 0$ for each $e \in E$ and vertex weights $q_v \geq 0$ for each $v \in V$.
Output: An orientation of edges $\gamma : E \to V$, where $\gamma(e) \in e$ for each $e \in E$.
Objective: Minimize $\max_{v\in V} \left(q_v + \sum_{e:\gamma(e)=v} p_e\right)$

Using binary search and scaling in a standard way, we can reduce the optimization problem to its decision version, where we test if there exists an orientation with the maximal load at most 1. Similarly, finding a 1.75-approximation can be reduced to the following relaxed version of the decision problem:

Problem Graph Balancing Approximation **(GBapx)**
Input: A weighted multigraph $G = (V, E, \mathbf{p}, \mathbf{q})$
Output:
either output FAIL, if for every orientation γ there is $v \in V$ with $q_v + \sum_{e:\gamma(e)=v} p_e > 1$,
or output an orientation γ such that for each $v \in V$: $q_v + \sum_{e:\gamma(e)=v} p_e \leq 1.75$.

Note that if the optimum is in $(1, 1.75]$, both answers are allowed. If there is an edge e with $p_e > 1$, we can immediately answer FAIL to (GBapx). Hence from now, w.l.o.g., we focus on the problem (GBapx) on instances with $p_e \leq 1$ for all edges $e \in E$.

2.2 2-approximation and rotations

We now describe the 2-approximation algorithm of Lenstra, Shmoys, and Tardos [9] in our restricted case. Our improved algorithm follows the same scheme and uses some of the same building blocks.

Again, it is sufficient to describe a relaxed decision procedure. Consider the following linear program, also called the assignment linear program:

Linear program (LP1)
Find values $x_{ev} \geq 0$ for each $e \in E$ and $v \in e$, subject to:
For each $e \in E, u, v \in e$:
$x_{eu} + x_{ev} = 1$ (Edge e)
For each $v \in V$:
$q_v + \sum_{e:v \in e} x_{ev} p_e \leq 1$ (Load at v)

The feasible integral solutions of (LP1) are in one-to-one correspondence with orientations γ with maximal load at most 1. The algorithm starts by finding a feasible fractional solution $\mathbf{x}$ of (LP1) if one exists; otherwise it outputs FAIL. Now it repeatedly takes an arbitrary cycle C in G with all edges with non-integral values x_{eu}, orients it in an arbitrary direction and applies procedure Rotate$(C, \mathbf{x})$ described formally later. This procedure modifies the solution $\mathbf{x}$ so that it is still feasible and the number of integral values in $\mathbf{x}$ increases. Eventually we arrive at a solution where edges with non-integral values form a forest. We take any one-to-one orientation of these edges, i.e., we root each tree component at an arbitrary vertex and orient its edges away from the root. We define the orientation of the remaining edges to agree with $\mathbf{x}$, i.e., $\gamma(e) = v$ if $x_{ev} = 1$. The output orientation has maximal load at most 2, as each vertex has load at most 1 from the single edge assigned to it by the orientation of the forest and at most 1 from the edges determined by the integral part of the linear program solution. The procedure Rotate is described in Algorithm 1 box (on the next page).

Obviously, Rotate is efficient; it needs only a linear number of arithmetic operations. All the sums $x_{eu} + x_{ev}$ and $\sum_{e:v \in e} x_{ev} p_e$ stay unchanged. Furthermore, all values x_{eu} stay non-negative and at least one x_{eu} becomes 0, while once a variable is equal to 0 it does not change. Together this implies that the whole algorithm is sound and terminates after at most a linear number of applications of Rotate. The computationally hardest part is to find a feasible solution of the linear program; this is also done in polynomial time using standard ellipsoid or interior point methods.

The rounding process has freedom in its choice of cycles that are to be rotated. This might seem to be a weak place of the algorithm. However, even in our restricted case the integrality gap of our linear program is 2, see Section 3.3. This means that there are instances where (LP1) is feasible, but any integral solution has maximal load arbitrarily close to 2. Thus, a more careful rounding cannot help on its own. A more careful choice of cycles to be rotated is a part of our approximation algorithm. However, as a main new part, we need to introduce a stronger linear program with the same integral solutions, and use a more complicated rounding process.

2.3 Notations and preprocessing

Given a weighted multigraph $G = (V, E, \mathbf{p}, \mathbf{q})$, let $E^B = \{e \in E \mid p_e > 0.5\}$ be the set of *big edges* and $G^B = (V, E^B)$ the subgraph with only the big edges. Given also a fractional assignment $\mathbf{x}$, let $E_{\mathbf{x}} = \{e \in E \mid 0 < x_{eu} < 1 \text{ for some } u \in e\}$ be the set of fractionally assigned edges and $G_{\mathbf{x}} = (V, E_{\mathbf{x}})$ the corresponding subgraph. Composing the notations, $G^B_{\mathbf{x}} = (G_{\mathbf{x}})^B = (V, E^B \cap E_{\mathbf{x}})$ is the subgraph of fractionally assigned big edges. Given a tree $T = (V, E)$, we call a *leaf pair* a pair of a leaf (vertex of degree 1) and the incident edge. Finally, $L(T) = \{(v, e) \in V \times E \mid \deg(v) = 1, v \in e\}$ is the set of all the leaf pairs of T.

In order for the maximal load to be at most one, the orientation γ restricted to $G^B = (V, E^B)$ has to be one-to-one. This gives us interesting consequences: Consider a component of G^B that has a cycle. Every one-to-one orientation of the edges on the cycle has in its range every vertex of the cycle. Hence for any edge e of that component not in the cycle, the only possibility is

Algorithm 1 Procedure Rotate

Procedure Rotate takes as input a feasible solution $\mathbf{x}$ of (LP1) and a directed cycle C in G and returns a modified solution $\mathbf{x}$

```
procedure Rotate($\mathbf{x}, C$)
    for all edges $e$ in $C$, where $e$ is directed from $u$ to $v$ do
        $\delta_e := x_{eu} p_e$
    $\delta := \min_{e \in C} \delta_e$
    for all edges $e$ in $C$, where $e$ is directed from $u$ to $v$ do
        $x_{eu} := x_{eu} - \delta / p_e$
        $x_{ev} := x_{ev} + \delta / p_e$
    return $\mathbf{x}$
```

to orient $\gamma(e)$ away from the cycle. Thus we can remove that edge and add its weight to the dedicated load of that vertex. This also implies that if there are two cycles in the component, no orientation γ is feasible. Hence we can preprocess our instance so that the subgraph G^B is a disjoint union of trees and cycles. We omit the details since later we will replace this preprocessing by a different trick.

3 The 1.75-approximation algorithm

What are other consequences of the fact that γ has to be one-to-one on E^B? Let us consider an arbitrary subtree T of the graph G^B of big edges. Since every tree has one more vertex than is the number of edges, the image of a one-to-one orientation γ can miss at most one of the vertices. In particular, if we consider the leaf pairs, all but one edges have to be oriented towards the leaves. Thus any integral solution of (LP1) has to satisfy

$$\sum_{(v,e) \in L(T)} x_{ev} p_e \geq \left(\sum_{(v,e) \in L(T)} p_e \right) - 1$$

for all trees $T \subseteq G^B$. We add this constraint to our linear program:

Linear program (LP2)
Find values $x_{ev} \geq 0$ for each $e \in E$ and $v \in e$, subject to:
For each $e \in E, u, v \in e$:
$x_{eu} + x_{ev} = 1$ (Edge e)
For each $v \in V$:
$q_v + \sum_{e: v \in e} x_{ev} p_e \leq 1$ (Load at v)
For each tree $T \subseteq G^B$:
$\sum_{(v,e) \in L(T)} x_{ev} p_e \geq \sum_{(v,e) \in L(T)} p_e - 1$ (Tree T)

Note that we are adding constraint (Tree T) for any tree T, which does not need to be the whole component. Here we might have justifiable misgivings about polynomiality of number of constraints (Tree T). However, for a given vector $\mathbf{x}$, in polynomial time we can verify all the constraints or find a constraint which is violated: This is done using the tree structure of the constraints and dynamic programming. Using this as a separation oracle, we can still solve the linear program in polynomial time using the ellipsoid method. We omit the details at this point, since in the end we find an initial feasible solution of (LP2) by solving a stronger linear program (LP3) with polynomial size and, moreover, at the same time (LP3) guarantees the proper structure of G^B and thus it also replaces the preprocessing described earlier.

We postpone the description of (LP3) to Section 3.2. Now in Section 3.1 we describe the rounding procedure.

3.1 The rounding procedure

In the 2-approximation algorithm, we first did all the rotations, obtained a solution with $G^B_{\mathbf{x}}$ being a forest and then defined the orientation on all the edges. In our case, the procedure is more complicated and alternates the rotations steps with the steps where we decide orientation of an edge or even some tree-like part of the graph.

Summarizing the assumptions, we have now a weighted multigraph $G = (V, E, \mathbf{p}, \mathbf{q})$ such that $p_e \leq 1$ for all $e \in E$. We also have a feasible solution $\mathbf{x}$ of (LP2) such that each component of $G^B_{\mathbf{x}}$ (the graph of not decided and big edges) is a cycle or a tree. Our goal is to find an orientation γ with maximal load at most 1.75. This is the specification of the procedure Round described in Algorithm 2 box. Note that during the procedure, as the solution $\mathbf{x}$ is changed, the graphs $G_{\mathbf{x}}$ and $G^B_{\mathbf{x}}$ change as well.

The step (Rotation) is well defined: Since there is no leaf in $G_{\mathbf{x}}$ in this case, the walk of the (FindCycle) can always continue. Every vertex is visited at most once, so it stops after a linear number of iterations. Each (Main While) iteration removes at least one edge from $G_{\mathbf{x}}$ and never adds an edge to it. Since we use only easy graph-algorithmic subroutines and procedure Rotate

Algorithm 2 Procedure Round

```
procedure Round($G = (V, E, \mathbf{p}, \mathbf{q}), \mathbf{x}$)
    while $G_{\mathbf{x}}$ has an edge do                                    ▷ (Main While)
        if $G_{\mathbf{x}}$ has a leaf pair $(v, e)$ then
            Let $u$ be the vertex $u \in e$, $u \neq v$.
            if $x_{eu}p_e \leq 0.75$ then                                   ▷ (Leaf Assignment)
                $(x_{ev}, x_{eu}) := (1, 0)$
            else                                                             ▷ (Tree Assignment)
                Let $T = (V', E')$ be the component of $G^B_{\mathbf{x}}$ containing $e$
                [$e \in G^B_{\mathbf{x}}$ in this case, and $T$ has to be a tree since $v$ is a leaf]
                for all $e' \in E'$, $v', u' \in e'$ such that $v'$ is on the path from $v$ to $u'$ in $T$ do
                    $(x_{e'v'}, x_{e'u'}) := (0, 1)$
        else                                                                 ▷ (Rotation)
            Find a directed cycle $C$ in the following way:
            Start a walk in an arbitrary vertex and repeat                   ▷ (FindCycle)
                Append a new edge to the end of the walk; if possible, choose a big edge
            until the walk contains a cycle, denote it $C$.
            Rotate($\mathbf{x}, C$)
    Let $\gamma(e) := v$ for all pairs $(e, v)$ with $x_{ev} = 1$
    return $\gamma$
```

that was already shown to be polynomial, the whole procedure Round runs in polynomial time. Finally, since the (Main While) cycle ends only when $G_{\mathbf{x}} = \emptyset$, the final assignment $\mathbf{x}$ is integral and the output γ indeed is an orientation.

It remains to show that the final load of each vertex is at most 1.75. We prove the following stronger theorem, which describes the invariants maintained during procedure Round.

Theorem 3.1. *Before and after each iteration of (Main While) during procedure* Round *for every vertex $v \in V$:*

(a) *The load of v is at most 1.75.*

(b) *If v is incident to any edge in $G_{\mathbf{x}}$, it has load at most 1.25.*

(c) *If v is incident to a big edge in $G_{\mathbf{x}}$ (i.e., any edge in $G^B_{\mathbf{x}}$), it has load at most 1.*

(d) *For every tree T that is a subgraph of $G^B_{\mathbf{x}}$, the constraint* (Tree T) *is not violated.*

Proof. At the beginning, all the conditions are true, since $\mathbf{x}$ is a feasible solution of (LP2). We show that all the conditions are preserved during each iteration of (Main While) by case analysis. In each case, we verify the conditions (a)–(c) for any vertex whose load changed and (d).

(Leaf Assignment): If $p_e \leq 0.5$ then the load of v goes up from at most 1.25 guaranteed by (b) before this iteration to at most 1.75. Otherwise before this step the vertex v is incident to a big edge, thus by (c) it has load at most 1 before this iteration and due to the case condition the load increases to at most 1.75. Thus in both cases after this step, v satisfies (a); (b) and (c) are void, as v is now isolated in $G_{\mathbf{x}}$. The loads of other vertices do not increase, which guarantees (a) to (c). Edge e is no longer in $G_{\mathbf{x}}$, thus no constraint (Tree T) can be violated and (d) holds.

(Tree Assignment): Before this step every vertex of the maximal tree T containing e is incident to a big edge of $G_{\mathbf{x}}$, hence by (c) it has load at most 1. Let us consider any vertex u' of T after the step. If $u' = v$ its load has decreased. If $u' \neq v$, there is a path $P \subseteq T$ from u' to v, starting by an edge e'. Since P is also a subtree of $G^B_{\mathbf{x}}$, we can use the constraint (Tree P):

$$x_{ev}p_e + x_{e'u'}p_{e'} \geq p_e + p_{e'} - 1$$

Using this in the first inequality below and the case condition $x_{eu}p_e > 0.75$ in the last inequality we obtain

$$\begin{aligned} x_{e'u'}p_{e'} &\geq p_e - x_{ev}p_e + p_{e'} - 1 \\ &= x_{eu}p_e + p_{e'} - 1 \\ &> 0.75 + p_{e'} - 1 \ = \ p_{e'} - 0.25 \end{aligned}$$

It follows that the load of u' increases by $p_{e'} - x_{e'u'}p_{e'} < 0.25$. Thus (a) and (b) holds for u'. Because of maximality of T, u' is not incident to a big edge after the step and (c) is void.

No vertices outside T increase their loads. Also, since all the maximal tree T is removed from $G_{\mathbf{x}}$, no constraint (Tree T) is violated and (d) holds.

(Rotation): By the analysis of ROTATE we know that every vertex keeps its load, thus (a) to (c) are preserved.

Take an arbitrary tree $T \subseteq G^B_{\mathbf{x}}$. To prove (d), we need to prove that (Tree T) is preserved. If (v,e) is a leaf pair in T, then x_{ev} is changed in ROTATE iff $e \in C$. More precisely, it increases by δ if e is directed from v and it decreases by δ if e is directed towards v. Thus (Tree T) seems to be problematic if there are more leaf pairs oriented from the leaf. We extend T to a certain tree T' and show that both (Tree T') and (Tree T) are preserved. A typical case is when $T \cap C$ is a directed path which starts by a leaf pair but then ends at some vertex t which is not a leaf of T. Then we add to T' the edge by which C continues from t; due to the choice of the cycle in (FindCycle) this edge is big. In general, $T \cap C$ may have several disjoint paths and we may need to add one edge for each path.

Formally, for any vertex t in C, let e_t be the edge of C starting at t. Let W be set of all vertices $t \in T \cap C$ such that e_t is not in T and t is not a leaf vertex of T. Finally, let $T' = T \cup \{e_t \mid t \in W\}$. We claim that any edge $e_t \in T' \setminus T$ is big. Since t is not a leaf in T, there is an edge in $G^B_{\mathbf{x}}$ which could be chosen by (FindCycle) at t, as there are at least two big edges incident to t and at most one of them is already on the path. Therefore the chosen edge e_t is big. Also, if the component of $G^B_{\mathbf{x}}$ containing T is a cycle, then $W = \emptyset$, as (FindCycle) follows the whole cycle once it reaches any of its vertices. We now see that T' is a tree, as whenever $T' \neq T$, it is a connected subgraph of a tree component of $G^B_{\mathbf{x}}$. It also follows that all the new edges e_t are leaf edges of T' oriented towards their leaf vertices in the orientation of C and that all the leaves in T are also leaves in T'. By the construction of T', the number of leaf pairs in T' oriented to the leaf vertex in the orientation of C is at least the number of leaf pairs of T' oriented away from the leaf. By the previous discussion, this guarantees that (Tree T') is maintained during ROTATE, and after this step we have

$$\sum_{(v,e)\in L(T')} x_{ev} p_e \geq \sum_{(v,e)\in L(T')} p_e - 1.$$

The inequalities $x_{ev} p_e \leq p_e$ are always true; we subtract them for all $(v,e) \in L(T') \setminus L(T)$ and obtain

$$\sum_{(v,e)\in L(T)} x_{ev} p_e \geq \sum_{(v,e)\in L(T)} p_e - 1.$$

This means that (Tree T) also holds and the proof of the case and also of the theorem is complete.

3.2 Solving the linear program

We can replace constraints (Tree T) by a polynomial set of new constraints. Consider an arbitrary vertex v and the set of big edges incident to it. In an integral solution, at most one of them can be assigned to v, hence the sum of x_{ev} over the star of big edges can be at most one. Thus the following linear program has the same set of integral solutions as (LP1) and (LP2) and thus is a valid LP-relaxation of GRAPH BALANCING.

Linear program (LP3)
Find values $x_{ev} \geq 0$ for each $e \in E$ and $v \in e$, subject to:
For each $e \in E, u, v \in e$:
$x_{eu} + x_{ev} = 1$ (Edge e)
For each $v \in V$:
$q_v + \sum_{e: v \in e} x_{ev} p_e \leq 1$ (Load at v)
For each vertex v:
$\sum_{e \in E^B: v \in e} x_{ev} \leq 1$ (Star v)

THEOREM 3.2. *Any solution* $\mathbf{x}$ *of* (LP3) *is also a solution of* (LP2). *Moreover, graph* $G^B_{\mathbf{x}}$ *is a disjoint union of cycles and trees.*

Proof. For a given solution of $\mathbf{x}$ of (LP3) we need to verify the constraints (Tree T) for any subtree $T = (V', E')$ of G^B. Take the sum of constraints (Edge e) over all edges e of T and subtract the sum of constraints (Star v) over all non-leaf vertices v of T to get

$$\begin{aligned}\sum_{(v,e)\in L(T)} x_{ev} &= \sum_{e\in E', v\in e} x_{ev} - \sum_{\substack{v\in V':\\ \deg(v)>1}} \sum_{\substack{e\in E':\\ v\in e}} x_{ev}\\ &\geq |L(T)| - 1\end{aligned}$$

Now we can use this inequality and $x_{ev} \leq 1$ to verify the constraint (Tree T):

$$\begin{aligned}\sum_{(v,e)\in L(T)} x_{ev} p_e &= \sum_{(v,e)\in L(T)} x_{ev} - \sum_{(v,e)\in L(T)} x_{ev}(1-p_e)\\ &\geq |L(T)| - 1 - \sum_{(v,e)\in L(T)} (1-p_e)\\ &= \sum_{(v,e)\in L(T)} p_e - 1\end{aligned}$$

To prove the second part of the theorem, consider any subgraph H of G^B consisting of a cycle with an edge e incident to one vertex on the cycle. Take again a sum of constraints (Edge e) over all edges e of H and subtract a sum of constraints (Star v) over all non-leaf vertices v of H. We obtain $x_{ev} \geq 1$ where (v,e) is the leaf pair of H. Hence the edge e cannot be in $G^B_{\mathbf{x}}$ as it is integral, and the proof is complete.

Let us formulate the algorithm for (GBapx).

Algorithm 3 LP-Balance

Algorithm LP-Balance has a weighted multigraph G on input and either returns FAIL or outputs orientation γ with maximal load at most 1.75.

```
Find a feasible solution $\mathbf{x}$ of (LP3) using some polynomial algorithm (ellipsoid or interior point method).
**if** no feasible solution exists **then return** FAIL
**return** Round$(G, \mathbf{x})$
```

Theorem 3.3. *Algorithm* LP-Balance *solves the relaxed decision problem (GBapx). Thus there exists a polynomial 1.75-approximation algorithm for the optimization problem* Graph Balancing.

Proof. The set of integral feasible solutions for (LP3) is in one-to-one correspondence with solutions of (GBapx) with maximal load at most 1. Thus if there is no feasible solution, the answer FAIL is correct. If there is a feasible solution, Theorems 3.2 and 3.1 guarantee that the output is an orientation with maximal load 1.75. Finally, a 1.75-approximation algorithm for the optimization problem is obtained by a binary search on the optimal makespan and scaling.

3.3 Integrality gaps and relations of linear programs

We have used three linear programs, (LP1), (LP2), and (LP3). All of them have the same integral solutions, but the set of fractional feasible solutions is decreasing.

The original linear program (LP1) has integrality gap 2 even in our special case. An example is a long path of edges of weight $1-\varepsilon$ with each endpoint having a dedicated load of 1. If the length of the path is more than $1/\varepsilon$, it is easy to see that (LP1) is feasible, while the best orientation has maximal load $2-2\varepsilon$.

The constraints (Tree T) are designed to avoid this example, and the stronger constraints (Star v) avoid it as well. Nevertheless, the integrality gap of both (LP2) and (LP3) is 1.75. Consider a graph with three long disjoint path with odd number of edges, all connecting the same two vertices u and v. The weights of edges on the paths alternate between 1 and $0.5-\varepsilon$, so that the edges incident to u and v have weight 1. In addition, there is a dedicated load of 0.25 on any vertex. Again, for sufficiently long paths, both (LP2) and (LP3) are feasible, as the new constraints are non-trivial only in the neighborhood of u and v. Any orientation has to assign at least two edges to the same vertex, which creates a load of at least 1.75. Thus to improve the approximation ratio, it is not sufficient to use our linear programs.

To compare our linear programs, (LP3) is the strongest one and has polynomial size, so it is more suitable for finding the initial solution. However it cannot be used directly in our rounding procedure. The reason is that the sum $\sum_{e:v\in e} x_{ev}$ used in (LP3) is not preserved by Rotate, but the sums of $p_e x_{ev}$ used in (LP2) are preserved.

4 The lower bound

Theorem 4.1. *Solving* Graph Balancing *with approximation ratio better than* 1.5 *is* NP*-hard.*

Proof. We prove NP-hardness by a reduction from from the following variant of 3-SAT: We are given a formula in CNF (conjunctive normal form), with clauses of size at most 3, with at most 3 occurrences of each variable, and with at most 2 occurrences of each literal (a variable or its negation). The NP-completeness of this problem goes back (at least) to [10]. In fact the reduction is easy: Given a general 3-CNF, we replace each occurrence of each variable x_i by a new variable $x_{i1}, \dots, x_{ik}$ for some k, and add new clauses for a chain of implications $x_{i1} \Rightarrow x_{i2}$, $x_{i2} \Rightarrow x_{i3}$, …, $x_{i(k-1)} \Rightarrow x_{ik}$, $x_{ik} \Rightarrow x_{i1}$ (written as disjunctions $\neg x_{i1} \vee x_{i2}, \dots$). Obviously, the new formula is satisfiable if and only if ϕ is. Every new variable occurs only three times and every literal occurs only twice.

The graphs in the reduction will use half-integral weights and dedicated loads, i.e., for all e, $p_e \in \{\frac{1}{2}, 1\}$ and for all v, $q_v \in \{0, \frac{1}{2}, 1\}$. (In fact the reduction can be easily modified to not use the dedicated loads.) Thus the load of each v is also half-integral. The reduction is constructed so that it is NP-hard to decide if there exists orientation such that the maximal load is 1. If not, then half-integrality of p_e and q_v implies that the maximal load is at least $\frac{3}{2}$. This gives the desired inapproximability bound.

In the rest of the proof, we denote by d_v the weighted degree of vertex, i.e., $d_v = q_v + \sum_{e:v\in e} p_e$.

The reduction: Given a 3-CNF(3) formula φ, we construct a graph $G(\varphi)$ as follows. There are vertices for each literal, i.e., each variable is represented by two vertices v_x and $v_{\neg x}$. There are also vertices v_α for each clause α in φ. The two vertices corresponding to each variable are connected with edge $e_x = \{v_x, v_{\neg x}\}$ of weight $p_{e_x} = 1$. (The intended meaning is that the orientation of e_x represents the false value. I.e., if $\gamma(e_x) = v_{\neg x}$, then x is true and vice versa.) Each clause vertex v_α is connected to vertices v_l corresponding to all of its literals l. The weight of each such edge $e_{\alpha l} = \{v_\alpha, v_l\}$ is $p_{e_{\alpha l}} = \frac{1}{2}$. The dedicated load q_{v_α} is set so that $d_{v_\alpha} = \frac{3}{2}$, i.e., we put $q_{v_\alpha} = 0$ for clauses with three literals, $q_{v_\alpha} = \frac{1}{2}$ for clauses with two literals, and $q_{v_\alpha} = 1$ for clauses with a single literal.

We claim that φ is satisfiable if and only if $G(\varphi)$ has an orientation γ with maximal load 1.

Satisfiability: Suppose we have an orientation γ with maximal load 1. We set variables in φ so that x is true if $\gamma(e_x) = v_{\neg x}$ and x is false otherwise. Consider an arbitrary clause α. Since $d_{v_\alpha} = \frac{3}{2}$, there is at least one literal l in α such that $\gamma(e_{\alpha l}) = v_l$. Let x be the variable in this literal l. Then $\gamma(e_x) \neq v_l$, because $p_{e_{\alpha l}} + p_{e_x} > 1$. So our edge $e_{\alpha l}$ demonstrates that l and α are both satisfied by the assignment x as defined above.

Orientability: Now we are given a satisfying assignment of our formula. We orient every edge e_x so that $\gamma(e_x) = v_{\neg x}$ if x is true and $\gamma(e_x) = v_x$ otherwise. We orient edges $e_{\alpha l}$ so that $\gamma(e_{\alpha l}) = v_l$ if l is true, and $\gamma(e_{\alpha l}) = v_\alpha$ otherwise. Each vertex v_l has load at most 1, as its load consists either of the single edge of weight 1 if l is false or of at most two edges of weight 1/2 if l is true. Each clause α is satisfied by at least one literal, so there is an edge $e_{\alpha l}$ with weight 1/2 satisfying $\gamma(e_{\alpha l}) \neq v_\alpha$. Thus the load of v_α is at most $d_{v_\alpha} - 1/2 = 1$.

Open problems

The main open problem is to improve the 2-approximation algorithm for unrelated machines. This seems to be a very hard problem, and we do not see how to extend our methods. Obviously, one approach would be to extend our result to some larger class of the restricted assignment problem, corresponding for example to k-uniform hypergraphs in place of graph.

Another interesting way to extend our result would be the case of unrelated graph balancing, by which we mean the case when the load (weight) of an edge is not the same for both endpoints. Even simpler but nontrivial generalization of Graph Balancing is the case with uniform speeds: every vertex has its speed and the load of an edge with some weight is given by the ratio of the edge weight and the vertex speed, for all allowed assignments.

Even for the case of graphs there is the remaining gap between 1.5 and 1.75. It would be nice to have a tight(er) bound.

Acknowledgments. We are grateful to Yossi Azar for discussions concerning this problem.

References

[1] A. Asadpour and A. Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. In *Proc. 39th Symp. Theory of Computing (STOC)*, pages 114–121. ACM, 2007.

[2] Y. Azar and A. Epstein. Convex programming for scheduling unrelated parallel machines. In *Proc. 37th Symp. Theory of Computing (STOC)*, pages 331–337. ACM, 2005.

[3] N. Bansal and M. Sviridenko. The Santa Claus problem. In *Proc. 38th Symp. Theory of Computing (STOC)*, pages 31–40. ACM, 2006.

[4] M. Gairing, B. Monien, and A. Woclaw. A faster combinatorial approximation algorithm for scheduling unrelated parallel machines. *Theor. Comput. Sci.*, 380:87–99, 2007.

[5] D. Hochbaum and D. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM J. Comput.*, 17, 1988.

[6] E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *J. ACM*, 23:317–327, 1976.

[7] K. Jansen and L. Porkolab. Improved approximation schemes for scheduling unrelated parallel machines. In *Proc. 31st Symp. Theory of Computing (STOC)*, pages 408–417. ACM, 1999.

[8] V. S. A. Kumar, M. V. Marathe, S. Parthasarathy, and A. Srinivasan. Approximation algorithms for scheduling on multiple machines. In *Proc. 45th Symp. Foundations of Computer Science (FOCS)*, pages 254–263. IEEE, 2005.

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

[10] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoret. Comput. Sci.*, 84:127–150, 1991.

[11] P. Schuurman and G. J. Woeginger. Polynomial time approximation algorithms for machine scheduling: Ten open problems. *J. of Scheduling*, 2:203–213, 1999.

[12] E. V. Shchepin and N. Vakhania. An optimal rounding gives a better approximation for scheduling unrelated machines. *Oper. Res. Lett.*, 33:127–133, 2005.

[13] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.