Noname manuscript No.
(will be inserted by the editor)

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.) 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], to a smaller constant for any special case with unbounded number of machines and unbounded processing times. We conclude by showing integrality gaps of several relaxations of related problems.

Partially supported by the Center of Excellence – Inst. for Theor. Comp. Sci., Prague (project P202/12/G061 of GA ČR and project 1M0545 of MŠMT ČR), grant IAA100190902 of GA AV ČR, and project GAUK 166610.

Tomáš Ebenlendr
Inst. of Mathematics, AS CR
Žitná 25
CZ-11567 Praha 1
Czech Republic
E-mail: ebik@math.cas.cz.

Marek Krčál
Dept. of Applied Mathematics
Faculty of Mathematics and Physics, Charles University
Malostranské nám. 25
CZ-11800 Praha 1
Czech Republic
E-mail: krcal@kam.mff.cuni.cz,

Jiří Sgall
Computer Science Inst. of Charles University
Faculty of Mathematics and Physics
Malostranské nám. 25
CZ-11800 Praha 1
Czech Republic
E-mail: sgall@iuuk.mff.cuni.cz

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. See Figure 1 for an example.

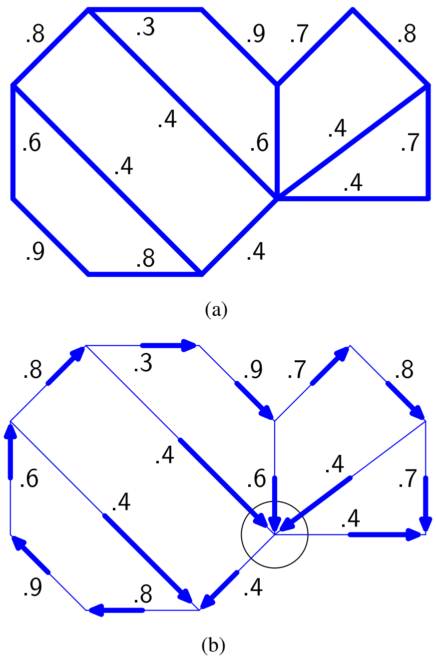

Fig. 1 Figure (a) shows an instance of the problem and Figure (b) its feasible solution with a circle denoting the vertex with the maximal load.

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 this 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 the 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, Shmoys, and Tardos [19] 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 [20], it is also covered in textbooks, e.g. [23].

Our results. Our main result is an 1.75-approximation algorithm for GRAPH BALANCING, see Section 3. Similarly as Lenstra *et al.* [19], 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 [19] is 2, and thus it is not sufficient to use their linear program with perhaps a more careful rounding. We strengthen 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.

The lower bound of 1.5 from Lenstra *et al.* [19] 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. However, it is possible to show that it is *NP*-hard to approximate GRAPH BALANCING with approximation ratio smaller than 1.5. This matches the state-of-the-art lower bound for the general problem of scheduling on unrelated machines. The proof uses a direct reduction from (a variant of) 3-SAT. We have presented the lower bound in the conference abstract of this paper [12]. The reduction was also discovered independently in an earlier published paper by Asahiro *et al.* [4, 5] in a different context and thus we do not include it here.

We conclude in Section 4 by discussing integrality gaps of the linear programs we are using and possible generalizations of our results. We show several examples showing that in a more general situation it may be hard to decrease the gap of the relaxations below 2. We prove that the strongest linear program we use for graph balancing has the integrality gap equal to 2 as soon as we allow a job to be scheduled on 3 machines instead of two, see Section 4.2. We also describe a strong linear program, called the configuration linear program, introduced for a similar problem by Bansal and Sviridenko [9] (see below). We show that it has integrality gap equal to 2 if we generalize graph balancing to allow different speeds for machines, similarly to uniformly related machines scheduling, still with each job restricted to at most two machines, see Section 4.3; this lower bound was independently obtained by Verschae and Wiese [25].

Our results for GRAPH BALANCING are the first improvement of an approximation factor to a constant below 2 for some special case of scheduling on unrelated machines with unbounded number of machines and unbounded processing times. Even though our special case is quite restricted, we find the problem interesting on its own. Also the lower bound shows that the restricted problem is still hard. A preliminary version of our results was published in [12].

Other related results. Subsequent to our work, Svensson [22] proved that the integrality gap of the configuration linear program (see Section 4.3) for the case of scheduling with restricted assignment is at most $33/17+\varepsilon \approx 1.9412$, where $\varepsilon > 0$ is arbitrarily small. This constitutes a major progress and gives an algorithm for estimation of the makespan with this ratio. However, the algorithm does not produce an actual schedule in polynomial time.

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 [15].

For the case of unrelated machines, polynomial time approximation schemes are known for a fixed number of machines [16,17]. For an unbounded number of machines, the 2-approximation algorithm can be slightly improved to $2-1/m$, see [21]. Another line of research is to obtain a 2-approximation without solving a linear program using flow algorithms, see [13] 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 [8,18].

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. The configuration linear program was first introduced in this area and used to obtain a $\log\log(n)/\log\log\log(n)$-approximation algorithm [9] for the restricted assignment version, which was later improved to a 5-estimation algorithm [1] ($(4+\varepsilon)$-estimation in a later unpublished revision); similarly to Svensson's algorithm this result estimates the objective but does not produce the actual solution. Finally, a constant approximation algorithm was presented [14], however, with a large constant factor. For the general version, the currently best algorithm gives an $\sqrt{n}\log^3 n$-approximation [2] and it is also based on the configuration program. For this objective, the restricted case of graph balancing was also studied: A 4-approximation algorithm was given in [10], 2-approximation algorithm based on a linear program in [11] and finally a simpler combinatorial 2-approximation algorithm in [25]; a matching lower bound of 2 was given in [3], even for the special case when the weights are in $\{1,2\}$.

As we mentioned above, the lower bound was found independently by Asahiro *et al.*[4,5]. They originally studied graph balancing and similar problems in a graph-theoretic context. An early result in this direction is an algorithm for the unweighted version [24]. Asahiro et al. [7,6] focus on algorithms for special graph classes and special cases of weights on edges, in some cases improving the ratio of 2; most notably, if the weights are restricted to $\{1,2\}$, they achieve an optimal 1.5-approximation algorithm.

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. A feasible solution 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 $\gamma(e) = v$. The objective is to minimize the maximal load among all vertices.

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. Since we started with a multigraph, a cycle may also include only 2 edges (connecting the same vertices but with opposite orientations).

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 *et al.* [19] 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 $e = \{u, v\} \in E$:

$$x_{eu} + x_{ev} = 1 \qquad \text{(Edge } e\text{)}$$

For each $v \in V$:

$$q_v + \sum_{e:v \in e} x_{ev} p_e \leq 1 \qquad \text{(Load at } v\text{)}$$

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.

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(x, C)
    for all edges e in C, where e is directed from u to v do
        δ_e := x_eu p_e
    δ := min_{e∈C} δ_e
    for all edges e in C, where e is directed from u to v do
        x_eu := x_eu − δ/p_e
        x_ev := x_ev + δ/p_e
    return x
```

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.

We note that in this special case, if a rotation along C is possible then a rotation with opposite orientation of C is also possible, so such a feasible solution is not a basic solution. Thus if a basic solution of the linear program is found, no rotations are needed.

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 4.1. 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: First, every component of G^B with k vertices has to contain at most k edges and thus it has to contain at most a single cycle. Second, if a component contains a cycle, then 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 to orient $\gamma(e)$ away from the cycle. Thus we can change the instance by removing every edge not in the cycle and adding its weight to the dedicated load of its endpoint farther from the cycle, without losing any integral feasible solution. Summarizing, after such preprocessing, in our instance the subgraph G^B is a disjoint union of trees and cycles—and we assume this from now on. We omit the details since later in Section 3.2 we will replace this preprocessing by a different idea.

3 The 1.75-approximation algorithm

To design a stronger linear program, we use 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 $T \subseteq G^B$. Our new linear program is obtained by adding all these constraints:

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

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 $G_{\mathbf{x}}$ 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. Also, it is not necessarily true that in a basic solution of the linear program no rotations are possible: A rotation in one direction may be impossible due to the tree constraint.

If $G_{\mathbf{x}}$ has a leaf v in an edge e, we try to assign e to v and call this step (Leaf Assignment). This is safe to do if the load of v is not increased by more than 0.75, as it will be never increased again. Otherwise, we need to direct e, which is necessarily a big edge, away from v. This in turn forces to direct the whole component of big edges away from v; we do so in step (Tree Assignment). We guarantee that the load of each vertex u increases by at most 0.25 by using the constraint (Tree T) for the path between v and u. Here we need a better bound on the increase of loads since these vertices can be assigned additional load in subsequent steps. If $G_{\mathbf{x}}$ has no leaf, we perform a rotation, carefully chosen so that no constraint (Tree T) is violated. We use the full strength of constraints (Tree T) only here, in the inductive proof that all these constraints are maintained. See Figure 3.1 for an example of the execution of the rounding procedure described below.

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 $G_{\mathbf{x}}^B$ (the graph of not decided 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 $G_{\mathbf{x}}^B$ 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 (except the last one) is visited at most once, so the search 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 running in polynomial time and procedure ROTATE 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.

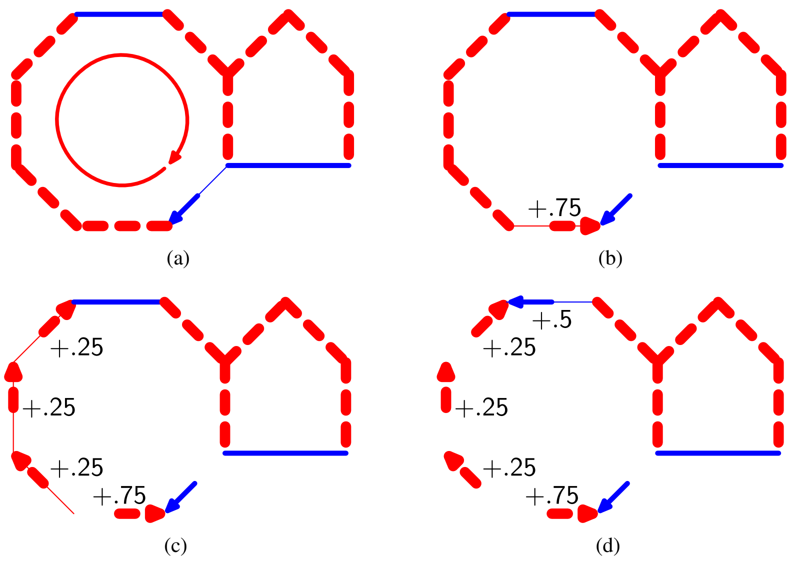

Fig. 2 An example of a possible execution of ROUND. Initially, the edges are assigned fractionally, gradually more of them are assigned as depicted by the arrows. Big edges are drawn as dashed (thick red) segments, other edges are solid lines (blue). In Figure (a), a rotation is performed and one edge becomes assigned. Figure (b) shows an application of (Leaf Assignment) and Figure (c) a subsequent application of (Tree Assignment); the labels give the maximal possible increase of loads. Finally, Figure (d) shows again an application of (Leaf Assignment), showing how the increase of the load of a vertex can be combined in different steps.

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 $e' = \{u', v'\} \in E'$ 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 this cycle C, directed in the direction of the walk
      ROTATE($\mathbf{x}, C$)
  Let $\gamma(e) := v$ for all pairs (e, v) with $x_{ev} = 1$
  return γ

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. For a fractional solution, we extend the definition of the load of v to be $q_v + \sum_{e:v\in e} x_{ev} p_e$.

Theorem 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 $G_{\mathbf{x}}^B$), it has load at most 1.*

(d) *For every tree T that is a subgraph of $G_{\mathbf{x}}^B$, 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 for v, 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 become violated and (d) holds.

(Tree Assignment): Before this step every vertex of the tree T 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 $G_{\mathbf{x}}^B$, 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, T is a maximal tree and it is removed from $G_{\mathbf{x}}$. Thus no constraint (Tree T) can become 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 $T \subseteq G_{\mathbf{x}}^B$. 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 decreases by δ if e is directed from v and it increases 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.

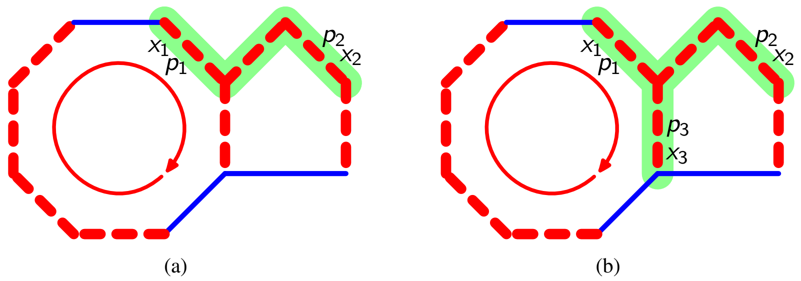

Fig. 3 An illustration of the analysis of the rotation of the left cycle. Figure (a) shows a shaded (light green) example of a tree T, with the labels denoting the loads used in the constraint (Tree T). Figure (b) shows as shaded its extension into the tree T'.

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 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 $e = \{u, v\} \in E$:
$$x_{eu} + x_{ev} = 1 \qquad \text{(Edge } e\text{)}$$
For each $v \in V$:
$$q_v + \sum_{e:v\in e} x_{ev} p_e \leq 1 \qquad \text{(Load at } v\text{)}$$
For each vertex v:
$$\sum_{e\in E^B:v\in e} x_{ev} \leq 1 \qquad \text{(Star } v\text{)}$$

Theorem 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 We first prove the (simpler) second part of the theorem. Suppose for a contradiction that $\mathbf{x}$ is a solution of (LP3) such that the graph $G^B_{\mathbf{x}}$ contains a cycle C of length k and an additional edge $\bar{e}$ incident to a vertex $\bar{v} \in C$ (its other endpoint may be either in C or outside of C). Summing all the constraints (Edge e) over all edges $e \in C$ we have
$$\sum_{e\in C}\sum_{u\in e} x_{eu} = k.$$
Summing all the constraints (Star v) over all vertices $v \in C$ and using the fact that $x_{\bar{e}\bar{v}} > 0$ as $\bar{e} \in G^B_{\mathbf{x}}$ and thus is assigned we obtain
$$k \geq \sum_{u\in C}\sum_{\substack{e\in E^B:\\ u\in e}} x_{eu} \geq x_{\bar{e}\bar{v}} + \sum_{e\in C}\sum_{u\in e} x_{eu} > \sum_{e\in C}\sum_{u\in e} x_{eu} = k\,,$$
a contradiction.

To prove the first part of the theorem, for a given solution $\mathbf{x}$ of (LP3) we need to verify the constraints (Tree T) for any subtree $T = (V', E')$ of G^B with k vertices. Take the sum of $k-1$ constraints (Edge e) over all edges e of T and subtract the sum of $k - |L(T)|$ constraints (Star v) over all non-leaf vertices v of T to get
$$\sum_{(v,e)\in L(T)} x_{ev} = \sum_{e\in E',v\in e} x_{ev} - \sum_{\substack{v\in V':\\ \deg(v)>1}} \sum_{\substack{e\in E':\\ v\in e}} x_{ev} \geq |L(T)| - 1\,.$$

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

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

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 *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 2 and 1 guarantee that the output is an orientation with maximal load at most 1.75. Finally, a 1.75-approximation algorithm for the optimization problem is obtained by a binary search on the optimal makespan and scaling.

4 Relations of linear programs and their extensions

4.1 Integrality gaps

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 $2-2\varepsilon$, with all edges oriented towards one inner vertex of the path.

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

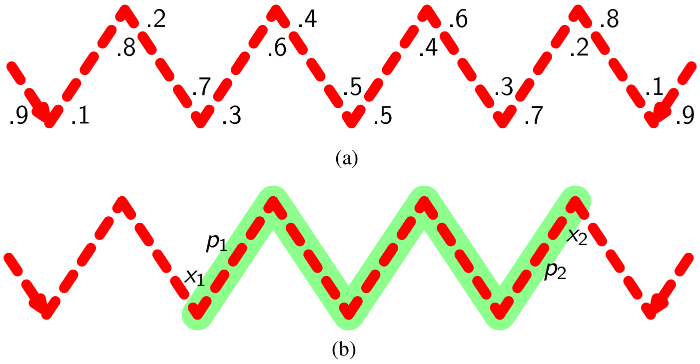

Fig. 4 An example of the integrality gap of 2 for (LP1). Figure (a) shows a feasible fractional solution of an instance with all edges of weight 0.9. Figure (b) shows a violated constraint (Tree T) in (LP2) as a shaded (light green) path.

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. See Figure 5. 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.

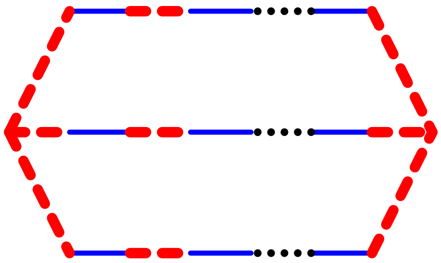

Fig. 5 An example of the integrality gap of 1.75 for (LP2) and (LP3). The edges of weight 1 are drawn as dashed (thick red) segments, the edges of weight $0.5-\varepsilon$ are solid (blue).

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. On the other hand, using the sums of $p_e x_{ev}$ in some local constraints like in (LP3) seems too weak to decrease the integrality gap; in the natural variants similar examples as for (LP1) above show that the gap is 2.

4.2 Modification of (LP3) for k-uniform hypergraphs.

A natural question is whether we can apply the similar approach for the case of restricted assignment where each job is restricted to some set of k machines, where k is fixed. One could hope for an approximation ratio depending on k but better than 2. In this section we show that even when $k = 3$, the straightforward generalization of (LP3), where the (Edge e) constraint is replaced by a constraint of form $x_{eu} + x_{ev} + x_{ew} = 1$, has the integrality gap 2. Consider an instance with a path consisting of vertices $u_1, u_2, \dots, u_n$ and edges $e_1, e_2, \dots, e_{n-1}$. Each edge has weight 1 and vertices u_1 and u_n have a dedicated load of 1. Moreover each edge e_i is incident to a vertex $v_i, i = 1, \dots, n-1$, where every v_i has a dedicated load of $1 - \varepsilon$. See Figure 6.

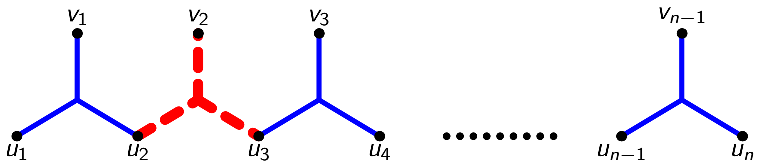

Fig. 6 An example of the integrality gap of 2 for (LP3) modification for 3-uniform hypergraphs. One of the hyperedges is drawn dashed (thick red), the others are solid (blue); all have weight 1.

For this instance, every (Star v) is equivalent to some (Load at v). For a sufficiently large n the linear program is feasible; with all vertices v_i assigned ε of the weight of the hyperedge and the rest of the weight of hyperedges assigned to vertices u_i similarly as in the example of the integrality gap for (LP1) in Section 4.1, see Figure 4.1. On the other hand, every orientation has to either assign an edge to a vertex with the dedicated load or has to assign two edges to the same vertex. In both cases the makespan is at least $2 - \varepsilon$.

4.3 Connections with fair division: Configuration linear program

In this section we consider a strong linear program, called the configuration linear program, introduced by Bansal and Sviridenko [9] for the Santa Claus problem. Let us start by formalizing this linear program for the case of scheduling unrelated parallel machines. Compared to our previous linear programs, instead of considering an indicator variable for each job and machine, we introduce an indicator variable x_{Si} for any subset of jobs S and any machine i. The intended meaning of the linear program is that $x_{Si} = 1$ in an assignment if S is the set of jobs assigned to machine i in the corresponding schedule with makespan at most t (for some $t \in \mathbb{R}$). For the rest of this section let p_{Si} denote $\sum_{j \in S} p_{ji}$. The constraints that we need to have satisfied are contained in the following linear program:

Linear program (ConfLP)
Find values $x_{Si} \geq 0$ for each machine i and every $S \subseteq [n]$, subject to:
Each machine i only accepts assignments of sets with load at most t: $x_{Si} = 0$ whenever $p_{Si} > t$.
Each machine i receives exactly one set: $\sum_S x_{Si} = 1$.
Every job j is assigned to exactly one machine: $\sum_{i,S \ni j} x_{Si} = 1$.

It follows that the integral solutions of the previous linear program are in bijective correspondence with schedules of makespan at most t. If the (ConfLP) is feasible for some t_0, then it is also feasible for all $t \geq t_0$. Let t^* be the minimum of all such values of t (it can be shown that such a minimum exists). The value of t^* and a feasible solution to the (ConfLP) of value t^* can be approximated within any desired degree of accuracy in polynomial time, as shown in [9].

In the case of GRAPH BALANCING it can be shown that (ConfLP) is a further strengthening of our (LP3): For any vertex v, if $x_{Sv} > 0$ then S can contain only one large edge (i.e., an edge with $p_e > 1/2$. The (ConfLP) solution is translated to assignment variables x_{ev} by setting $x_{ev} = \sum_{S:e \in S} x_{Sv}$; the previous observation implies that $\sum_e x_{ev} \leq 1$ for the sum taken over all large edges, i.e., the constraint ((Star v)) of (LP3) holds. Thus the integrality gap of (ConfLP) for GRAPH BALANCING is at most 1.75—but as far as we know it could be as low as 1.5.

By a private communication with Bansal and Sviridenko we know that in the case of unrelated parallel machines the integrality gap of the (ConfLP) is 2. Consider the following instance of the problem:

There are machines $c_1, c_2, \ldots, c_n$ and $c'_1, c'_2, \ldots, c'_n$. For each pair of machines c_i, c'_i there are of n jobs each having processing time $1/n$ on c_i, 1 on c'_i and ∞ otherwise; (let us denote it *ith group*). Besides these jobs, there is one additional job (let us call it a *special job*) that can be processed on any machine $c_1, c_2, \ldots, c_n$ with processing time 1.

It can be easily seen that no schedule attains makespan less than $2 - 1/n$, as either one of machines c'_i receives two jobs, or one of machines c_i receives the special job and $n-1$ jobs from ith group. On the other hand there is a fractional solution of (ConfLP) with $t = 1$ as follows: To every machine c_i we assign a $1/n$ fraction of the special job is assigned (as a singleton subset) as well as $(n-1)/n$ fraction of the ith group (as a set of all jobs in the group). Finally, $1/n$ fraction of every job of every *ith group* is assigned (as a singleton subset) to the machine c'_i.

Can we do any better in some special cases of scheduling unrelated parallel machines? Surprisingly, the same gap appears even in the much simpler setting of *graph balancing with speeds*. In this setting, each machine (vertex) has its speed s_i, each job (edge) has its processing time p_j and to schedule job j on machine i take time $p_{ji} = p_j/s_i$. As in graph balancing, each job is restricted to at most two machines and cannot be scheduled at any other machine, i.e., $p_{ji} = \infty$.

We show that in this setting the configuration linear program has integrality gap equal to 2. This lower bound was independently obtained by Verschae and Wiese [25] (they claimed the bound only for the case of unrelated graph balancing, where the times on the two allowed machines may be independent of other jobs' times, but it works also in our setting with speeds, which gives a stronger integrality gap result).

Theorem 4 *The integrality gap of the configuration linear program* (ConfLP) *is equal to 2, even for the case of graph balancing with speeds.*

Proof The integrality gap is at most two even for the weaker linear program (LP1). We construct an example showing that the integrality gap of (ConfLP) is at least $2-\varepsilon$, for any $\varepsilon > 0$.

The instance again contains the vertices $c_1, \ldots, c_m$ with speed 1 and $c'_1, \ldots, c'_m$ with speed $1/n$ where m is a large integer that we will specify later and n is the parameter determining the gap. For every pair of vertices c_i, c'_i there is again an *ith group* of n edges with weight $1/n$. The special job is replaced by the following tree consisting of levels $0, \ldots, K$ (K will be again specified later), which is essentially an n-ary tree with two roots at level K connected by an edge. The level 0 consists of vertices $c_1, \ldots, c_m$. For every $j = 1, \ldots, K$ the level j consists of vertices with speed $n^j + n^{j-1}$. Each such vertex is connected by n edges of weight n^{j-1} to n distinct vertices in the level $j-1$ and by a single edge of weight n^j to some vertex in level $j+1$. Also each of the vertices in the level 0 (c_1, ..., c_m) is connected by a single edge of weight 1 to some vertex in the level 1. In the level K there are exactly two vertices and instead of the edge connecting them to the higher level there is an edge of weight n^K connecting them together. See an example in Figure 7.

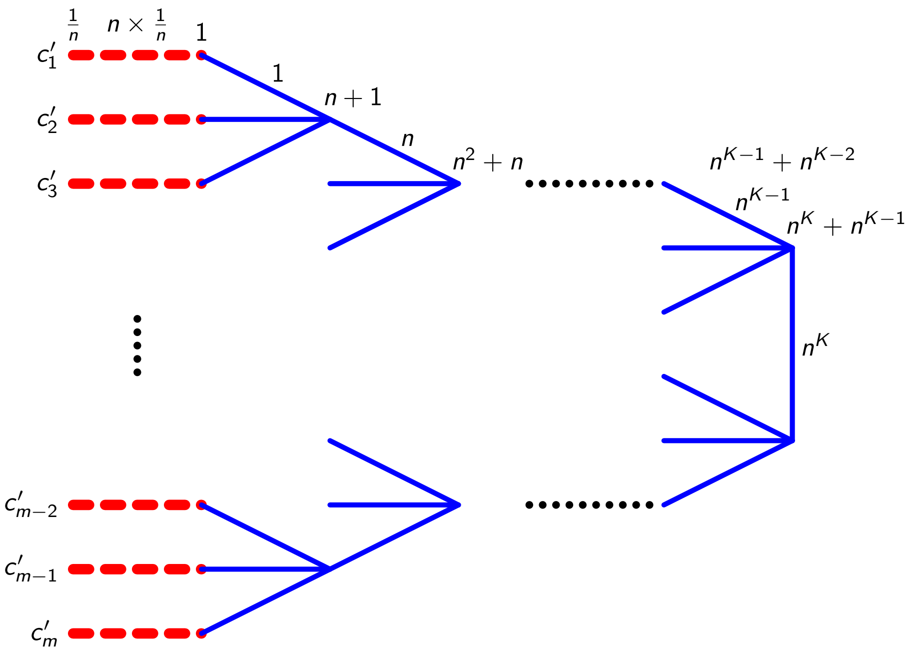

Fig. 7 An example of the integrality gap of 2 for graph balancing with speeds, with the levels arranged in columns. The dashed (thick red) segments denote a group of n edges of weight $1/n$. The solid segments (blue) denote each a single edge of weight given on the top (all the edges in the same column have the same weight). The speed of vertices is given above the top vertex; again the column has vertices of equal speed.

It is not difficult to see that no orientation achieves makespan less than $2-2/(n+1)$: For a smaller load, only one edge can be assigned to each c_i'; working from the lower levels, all the edges have to be assigned to the faster machine on the higher level and the edge connecting the two vertices in the level K cannot be assigned.

On the other hand there exists a solution of (ConfLP) with $t=1$: Every ith group is assigned as in the previous example so that every vertex c_i there is free capacity $1/n$ for the edge of weight 1 which is assigned as a singleton subset. To every vertex of every level j ($1 \le j \le K$) of the tree we assign a $((n-1)^j - n^{j-1})/(n-1)^j$ fraction of a subset of all n edges going down to level $j-1$. The remaining capacity $n^{j-1}/(n-1)^j$ is divided evenly among the n subsets containing two edges: one of the n edges going down to level $j-1$ and either the only edge going up to level $j+1$ or, if $j=K$, the only edge going to the second vertex in the top-most level. What remains to check is that every edge is fully assigned. Every edge between the levels $j-1$ and j ($j=1,\dots,K$) is assigned in the following three ways:

1. Down to the vertex in the level $j-1$ (in many subsets for $j>1$), by a fractional part $n^{j-2}/(n-1)^{j-1}$. Note that this fraction is equal to $1/n$ for $j=1$.
2. Up to the vertex in the level j (in the subset of n edges) by a fractional part $((n-1)^j - n^{j-1})/(n-1)^j$.
3. Up to the vertex in the level j (in the subset with the edge of weight n^j) by a fractional part $n^{j-2}/(n-1)^j$.

It can be verified that the previous fractions sum up to 1. The only edge that is left to be checked is the edge in the level K. But if we choose the number of levels K sufficiently large so that $n^{K-1}/(n-1)^K \ge 1/2$, then also this edge is fully assigned and the solution is correct. The number of levels also determines the number of vertices in each level, in particular we have $m = 2n^K$.

5 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, as the examples in Section 4 indicate, using our linear programs and also the configuration LP is not sufficient. We do not see any promising approach at the moment.

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.

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.

Acknowledgments.

We are grateful to Yossi Azar, Nikhil Bansal and Maxim Sviridenko for discussions concerning this problem. We are grateful to Eiji Miyano for pointing out the work of their group on the same problem. We are also grateful to the anonymous referees for numerous and insightful comments that helped to improve our presentation.

References

1. A. Asadpour, U. Feige, and A. Saberi. Santa Claus meets hypergraph matchings. In *Approximation, Randomization and Combinatorial Optimization: Proc. of the 11th Int. Workshop APPROX 2008 and 12th Int. Workshop RANDOM*, volume 5171 of *Lecture Notes in Comput. Sci.*, pages 10–20. Springer, 2008.
2. A. Asadpour and A. Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. *SIAM J. Comput.*, 39:2970–2989, 2010.
3. Y. Asahiro, J. Jansson, E. Miyano, and H. Ono. Graph orientation to maximize the minimum weighted outdegree. *Int. J. on Found. Comput. Sci.*, 22:583–601, 2011.
4. Y. Asahiro, J. Jansson, E. Miyano, H. Ono, and K. Zenmyo. Approximation algorithms for the graph orientation minimizing the maximum weighted outdegree. In *Proc. 2nd International Conf. on Algorithmic Aspects in Information and Management (AAIM)*, volume 4508 of *Lecture Notes in Comput. Sci.*, pages 167–177. Springer, 2007.
5. Y. Asahiro, J. Jansson, E. Miyano, H. Ono, and K. Zenmyo. Approximation algorithms for the graph orientation minimizing the maximum weighted outdegree. *J. Comb. Optim.*, 22:78–96, 2011.
6. Y. Asahiro, E. Miyano, and H. Ono. Graph classes and the complexity of the graph orientation minimizing the maximum weighted outdegree. *Discrete Appl. Math.*, 159:498–508, 2011.
7. Y. Asahiro, E. Miyano, H. Ono, and K. Zenmyo. Graph orientation algorithms to minimize the maximum outdegree. *Int. J. on Found. Comput. Sci.*, 18:197–215, 2007.
8. 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.
9. N. Bansal and M. Sviridenko. The Santa Claus problem. In *Proc. 38th Symp. Theory of Computing (STOC)*, pages 31–40. ACM, 2006.
10. M. Bateni, M. Charikar, and V. Guruswami. Maxmin allocation via degree lower-bounded arborescences. In *Proc. 41st Symp. Theory of Computing (STOC)*, pages 543–552. ACM, 2009.
11. D. Chakrabarty, J. Chuzhoy, and S. Khanna. On allocating goods to maximize fairness. In *Proc. 50th Symp. Foundations of Computer Science (FOCS)*, pages 107–116. IEEE, 2009.
12. T. Ebenlendr, M. Krčál, and J. Sgall. Graph balancing: a special case of scheduling unrelated parallel machines. In *Proc. 19th Symp. on Discrete Algorithms (SODA)*, pages 483–490. ACM/SIAM, 2008.
13. M. Gairing, B. Monien, and A. Woclaw. A faster combinatorial approximation algorithm for scheduling unrelated parallel machines. *Theoret. Comput. Sci.*, 380:87–99, 2007.
14. B. Haeupler, B. Saha, and A. Srinivasan. New constructive aspects of the Lovasz local lemma. In *Proc. 51st Symp. Foundations of Computer Science (FOCS)*, pages 397–406. IEEE, 2010.
15. D. S. Hochbaum and D. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM J. Comput.*, 17, 1988.
16. E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *J. ACM*, 23:317–327, 1976.
17. K. Jansen and L. Porkolab. Improved approximation schemes for scheduling unrelated parallel machines. *Math. Oper. Res.*, 26:324–338, 2001.
18. V. S. A. Kumar, M. V. Marathe, S. Parthasarathy, and A. Srinivasan. A unified approach to scheduling on unrelated parallel machines. *J. ACM*, 56:Article 28, 2009.
19. J. K. Lenstra, D. B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46:259–271, 1990.
20. P. Schuurman and G. J. Woeginger. Polynomial time approximation algorithms for machine scheduling: Ten open problems. *J. Sched.*, 2:203–213, 1999.
21. E. V. Shchepin and N. Vakhania. An optimal rounding gives a better approximation for scheduling unrelated machines. *Oper. Res. Lett.*, 33:127–133, 2005.
22. O. Svensson. Santa Claus schedules jobs on unrelated machines. In *Proc. 43th Symp. Theory of Computing (STOC)*, pages 617–626. ACM, 2011.
23. V. V. Vazirany. *Approximation algorithms*. Springer, 2001.
24. V. Venkateswaran. Minimizing maximum indegree. *Discrete Appl. Math.*, 143:374–378, 2004.
25. J. Verschae and A. Wiese. On the configuration-LP for scheduling on unrelated machines. In *Proc. 19th European Symp. on Algorithms (ESA)*, volume 6942 of *LNCS*, pages 530–542. Springer, 2011.