A Combinatorial Approximation Algorithm for Graph Balancing with Light Hyper Edges

Chien-Chung Huang[1] and Sebastian Ott[2]

1 **Chalmers University of Technology, Göteborg, Sweden**
villars@gmail.com

2 **Max-Planck-Institut für Informatik, Saarbrücken, Germany**
ott@mpi-inf.mpg.de

Abstract

Makespan minimization in restricted assignment $(R|p_{ij} \in \{p_j, \infty\}|C_{\max})$ is a classical problem in the field of machine scheduling. In a landmark paper in 1990 [9], Lenstra, Shmoys, and Tardos gave a 2-approximation algorithm and proved that the problem cannot be approximated within 1.5 unless P=NP. The upper and lower bounds of the problem have been essentially unimproved in the intervening 25 years, despite several remarkable successful attempts in some special cases of the problem [2, 3, 13] recently.

In this paper, we consider a special case called *graph-balancing with light hyper edges*, where heavy jobs can be assigned to at most two machines while light jobs can be assigned to any number of machines. For this case, we present algorithms with approximation ratios strictly better than 2. Specifically,

- **Two job sizes**: Suppose that light jobs have weight w and heavy jobs have weight W, and $w < W$. We give a 1.5-approximation algorithm (note that the current 1.5 lower bound is established in an even more restrictive setting [1, 4]). Indeed, depending on the specific values of w and W, sometimes our algorithm guarantees sub-1.5 approximation ratios.
- **Arbitrary job sizes**: Suppose that W is the largest given weight, heavy jobs have weights in the range of $(\beta W, W]$, where $4/7 \leq \beta < 1$, and light jobs have weights in the range of $(0, \beta W]$. We present a $(5/3 + \beta/3)$-approximation algorithm.

Our algorithms are purely combinatorial, without the need of solving a linear program as required in most other known approaches.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Approximation Algorithms, Machine Scheduling, Graph Balancing, Combinatorial Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.49

1 Introduction

Let $\mathcal{J}$ be a set of n jobs and $\mathcal{M}$ a set of m machines. Each job $j \in \mathcal{J}$ has a *weight* w_j and can be assigned to a specific subset of the machines. An assignment $\sigma : \mathcal{J} \to \mathcal{M}$ is a mapping where each job is mapped to a machine to which it can be assigned. The objective is to minimize the *makespan*, defined as $\max_{i \in \mathcal{M}} \sum_{j:\sigma(j)=i} w_j$. This is the classical MAKESPAN MINIMIZATION IN RESTRICTED ASSIGNMENT $(R|p_{ij} \in \{p_j, \infty\}|C_{\max})$, itself a special case of the MAKESPAN MINIMIZATION IN UNRELATED MACHINES $(R||C_{\max})$, where a job j has possibly different weight w_{ij} on different machines $i \in \mathcal{M}$. In the following, we just call them RESTRICTED ASSIGNMENT and UNRELATED MACHINE PROBLEM for short.

The first constant approximation algorithm for both problems is given by Lenstra, Shmoys, and Tardos [9] in 1990, where the ratio is 2. They also show that RESTRICTED ASSIGNMENT (hence also the UNRELATED MACHINE PROBLEM) cannot be approximated within 1.5 unless P=NP, even if there are only two job weights. The upper bound of 2 and the lower bound of 1.5 have been essentially unimproved in the intervening 25 years. How to close the gap continues to be one of the central topics in approximation algorithms. The recent book of Williamson and Shmoys [15] lists this as one of the ten open problems.

Our Result

We consider a special case of RESTRICTED ASSIGNMENT, called GRAPH BALANCING WITH LIGHT HYPER EDGES, which is a generalization of the GRAPH BALANCING problem introduced by Ebenlendr, Krčál and Sgall [4]. There the restriction is that every job can be assigned to only two machines, and hence the problem can be interpreted in a graph-theoretic way: each machine is represented by a node, and each job is represented by an edge. The goal is to find an orientation of the edges so that the maximum weight sum of the edges oriented towards a node is minimized. In our problem, jobs are partitioned into *heavy* and *light*, and we assume that heavy jobs can go to only two machines while light jobs can go to any number of machines[1]. In the graph-theoretic interpretation, light jobs are represented by hyper edges, while heavy jobs are represented by regular edges.

We present approximation algorithms with performance guarantee *strictly better than* 2 in the following settings. For simplicity of presentation, we assume that all job weights w_j are integral (this assumption is just for ease of exposition and can be easily removed).

Two job sizes: Suppose that heavy jobs are of weight W and light jobs are of weight w, and $w < W$. We give a 1.5-approximation algorithm, matching the general lower bound of RESTRICTED ASSIGNMENT (it should be noted that this lower bound is established in an even more restrictive setting [1, 4], where all jobs can only go to two machines and job weights are 1 and 2). This is the first time the lower bound is matched in a nontrivial case of RESTRICTED ASSIGNMENT (without specific restrictions on the job weight values). In fact, sometimes our algorithm achieves an approximation ratio strictly better than 1.5. Supposing that $w \leq \frac{W}{2}$, the ratio we get is $1 + \frac{\lfloor W/2 \rfloor}{W}$.

Arbitrary job sizes: Suppose that $\beta \in [4/7, 1)$ and W is the largest given weight. A heavy job has weight in $(\beta W, W]$ while a light job has weight in $(0, \beta W]$. We give a $(5/3 + \beta/3)$-approximation algorithm.

Both algorithms have the running time of $\mathcal{O}\big(n^2 m^3 \log\big(\sum_{j \in \mathcal{J}} w_j\big)\big)$.[2]

The general message of our result is clear: as long as the heaviest jobs have only two choices, it is relatively easy to break the barrier of 2 in the upper bound of RESTRICTED ASSIGNMENT. This should coincide with our intuition. The heavy jobs are in a sense the "trouble-makers". A mistake on them causes bigger damage than a mistake on lighter jobs. Restricting the choices of the heavy jobs thus simplifies the task.

[1] If some jobs can be assigned to just one machine, then it is the same as saying a machine has some *dedicated load*. All our algorithms can handle arbitrary dedicated loads on the machines.

[2] For simplicity, here we upper bound $\sum_{j \in \mathcal{J}} a_j$, where a_j is the number of the machines j can be assigned to, by nm.

The original GRAPH BALANCING problem assumes that all jobs can be assigned to only two machines and the algorithm of Ebenlendr et al. [4] gives a 1.75-approximation. According to [11], their algorithm can be extended to our setting: given any $\beta \in [0.5, 1)$, they can obtain a $(3/2 + \beta/2)$-approximation. Although this ratio is superior to ours, let us emphasize two interesting aspects of our approach.

(1) The algorithm of Ebenlendr et al. requires solving a linear program (in fact, almost all known algorithms for the problem are LP-based), while our algorithms are purely combinatorial. In addition to the advantage of faster running time, our approach introduces new proof techniques (which do not involve linear programming duality).

(2) In GRAPH BALANCING, Ebenlendr et al. showed that with only two job weights and dedicated loads on the machines, their strongest LP has the integrality gap of 1.75, while we can break the gap. Our approach thus offers a possible angle to circumvent the barrier posed by the integrality gap, and has the potential of seeing further improvement.

Before explaining our technique in more detail, we should point out another interesting connection with a result of Svensson [13] for general RESTRICTED ASSIGNMENT. He gave two local search algorithms, which terminate (but it is unknown whether in polynomial time) and (1) with two job weights $\{\epsilon, 1\}$, $0 < \epsilon < 1$, the returned solution has an approximation ratio of $5/3 + \epsilon$, and (2) with arbitrary job weights, the returned solution has an approximation ratio of ≈ 1.94. It is worth noting that his analysis is done via the primal-duality of the configuration-LP (thus integrality gaps smaller than two for the configuration-LP are implied). With two job weights, our algorithm has some striking similarity to his algorithm – but it should be emphasized that the two algorithms behave differently. We are able to prove our algorithm terminates in polynomial time – but our setting is more restrictive. A very interesting direction for future work is to investigate how the ideas in the two algorithms can be related and combined.

Our Technique

Our approach is inspired by that of Gairing et al. [5] for general RESTRICTED ASSIGNMENT. So let us first review their ideas. Suppose that a certain optimal makespan t is guessed. Their core algorithm either (1) correctly reports that t is an underestimate of OPT, or (2) returns an assignment with makespan at most $t + W - 1$. By a binary search on the smallest t for which an assignment with makespan $t + W - 1$ is returned, and the simple fact that $\text{OPT} \geq W$, they guarantee the approximation ratio of $\frac{t+W-1}{\text{OPT}} \leq 1 + \frac{W-1}{\text{OPT}} \leq 2 - \frac{1}{W}$ (the first inequality holds because t is the smallest number an assignment is returned by the core algorithm). Their core algorithm is a preflow-push algorithm. Initially all jobs are arbitrarily assigned. Their algorithm tries to redistribute the jobs from overloaded machines, i.e., those with load more than $t + W - 1$, to those that are not. The redistribution is done by pushing the jobs around while updating the height labels (as commonly done in preflow-push algorithms). The critical thing is that after a polynomial number of steps, if there are still some overloaded machines, they use the height labels to argue that t is a wrong guess, i.e., $\text{OPT} \geq t + 1$. Our contribution is a refined core algorithm in the same framework. With a guess t of the optimal makespan, our core algorithm either (1) correctly reports that $\text{OPT} \geq t + 1$, or (2) returns an assignment with makespan at most $(5/3 + \beta/3)t$.

We divide all jobs into two categories, the *rock jobs* $\mathbb{R}$, and the *pebble jobs* $\mathbb{P}$ (not to be confused with heavy and light jobs). The former consists of those with weights in $(\beta t, t]$ while the latter includes all the rest. We use the rock jobs to form a graph $G_{\mathbb{R}} = (V, \mathbb{R})$, and assign the pebbles arbitrarily to the nodes. Our core algorithm will push around the pebbles so as to redistribute them. Observe that as $t \geq W$, all rocks are heavy jobs. So the formed graph $G_{\mathbb{R}}$ has only simple edges (no hyper edges). As $\beta \geq 4/7$, if $\text{OPT} \leq t$, then every node can receive at most one rock job in the optimal solution. In fact, it is easy to see that we can simply assume that the formed graph $G_{\mathbb{R}}$ is a disjoint set of trees and cycles. Our entire task boils down to the following:

> Redistribute the pebbles so that there exists an orientation of the edges in $G_{\mathbb{R}}$ in which each node has total load (from both rocks and pebbles) at most $(5/3 + \beta/3)t$; and if not possible, gather evidence that t is an underestimate.

Intuitively speaking, our algorithm maintains a certain *activated set* $\mathbb{A}$ of nodes. Initially, this set includes those nodes whose total loads of pebbles cause conflicts in the orientation of the edges in $G_{\mathbb{R}}$. A node "reachable" from a node in the activated set is also included into the set. (Node u is reachable from node v if a pebble in v can be assigned to u.) Our goal is to push the pebbles among nodes in $\mathbb{A}$, so as to remove all conflicts in the edge orientation. Either we are successful in doing so, or we argue that the total load of all pebbles currently owned by the activated set, together with the total load of the rock jobs assigned to $\mathbb{A}$ in any *feasible orientation* of the edges in $G_{\mathbb{R}}$ (an orientation in $G_{\mathbb{R}}$ is *feasible* if every node receives at most one rock), is strictly larger than $t \cdot |\mathbb{A}|$. The progress of our algorithm (hence its running time) is monitored by a potential function, which we show to be monotonically decreasing.

The most sophisticated part of our algorithm is the "activation strategy". We initially add nodes into $\mathbb{A}$ if they cause conflicts in the orientation or can be (transitively) reached from such. However, sometimes we also include nodes that do not fall into the two categories. This is purposely done for two reasons: pushing pebbles from these nodes may help alleviate the conflict in edge orientation indirectly; and their presence in $\mathbb{A}$ strengthens the contradiction proof.

Due to the intricacy of our main algorithm, we first present the algorithm for the two job weights case in Section 3 and then present the main algorithm for the arbitrary weights in Section 4. The former algorithm is significantly simpler (with a straightforward activation strategy) and contains many ingredients of the ideas behind the main algorithm.

Due to the space limit, some of the proofs are omitted. Please refer to the full version [7] for details.

Related Work

For RESTRICTED ASSIGNMENT, besides the several recent advances mentioned earlier, see the survey of Leung and Li for other special cases [10]. For two job weights, Chakrabarti, Khanna and Li [2] showed that using the configuration-LP, they can obtain a $(2 - \delta)$-approximation for a fixed $\delta > 0$ (and note that there is no restriction on the number of machines a job can go to). Kolliopoulos and Moysoglou [8] also considered the two job weights case. In the GRAPH BALANCING setting (with two job weights), they gave a 1.652-approximation algorithm using a flow technique (thus they also break the integrality gap in [3]). They also show that the optimal makespan for RESTRICTED ASSIGNMENT with two job weights can be estimated in polynomial time within a factor of at most 1.883 (and this is further improved to 1.833 in [2]).

For UNRELATED MACHINES, Shchepin and Vakhania [12] improved the approximation ratio to $2 - 1/m$. A combinatorial 2-approximation algorithm was given by Gairing, Monien, and Woclaw [6]. Verschae and Wiese [14] showed that the configuration-LP has integrality gap of 2, even if every job can be assigned to only two machines. They also showed that it is possible to achieve approximation ratios strictly better than 2 if the job weights w_{ij} respect some constraints.

2 Preliminary

Let t be a guess of OPT. Given t, our two core algorithms either report that $\mathrm{OPT} \geq t+1$, or return an assignment with makespan at most $1.5t$ or $(5/3+\beta/3)t$, respectively. We conduct a binary search on the smallest $t \in [W, \sum_{j \in \mathcal{J}} w_j]$ for which an assignment is returned by the core algorithms. This particular assignment is then the desired solution.

We now explain the initial setup of the core algorithms. In our discussion, we will not distinguish a machine and a node. Let $dl(v)$ be the dedicated load of v, i.e., the sum of the weights of jobs that can only be assigned to v. We can assume that $dl(v) \leq t$ for all nodes v. Let $\mathcal{J}' \subseteq \mathcal{J}$ be the jobs that can be assigned to at least two machines. We divide $\mathcal{J}'$ into rocks $\mathbb{R}$ and pebbles $\mathbb{P}$. A job $j \in \mathcal{J}'$ is a rock,

- in the 2 job weights case (Section 3), if $w_j > t/2$ and $w_j = W$;
- in the general job weights case (Section 4), if $w_j > \beta t$.

A job $j \in \mathcal{J}'$ that is not a rock is a pebble. Define the graph $G_{\mathbb{R}} = (V, \mathbb{R})$ as a graph with machines $\mathcal{M}$ as node set and rocks $\mathbb{R}$ as edge set. By our definition, a rock can be assigned to exactly two machines. So $G_{\mathbb{R}}$ has only simple edges (no hyper edges). For the sake of convenience, we call the rocks just "edges", avoiding ambiguity by exclusively using the term "pebble" for the pebbles.

Suppose that $\mathrm{OPT} \leq t$. Then a machine can receive at most one rock in the optimal solution. If any connected component in $G_{\mathbb{R}}$ has more than one cycle, we can immediately declare that $\mathrm{OPT} \geq t+1$. If a connected component in $G_{\mathbb{R}}$ has exactly one cycle, we can direct all edges away from the cycle and remove these edges, i.e., assign the rock to the node v to which it is directed. W.L.O.G, we can assume that this rock is part of v's dedicated load. (Also observe that then node v must become an isolated node). Finally, we can eliminate cycles of length 2 in $G_{\mathbb{R}}$ with the following simple reduction. If a pair of nodes u and v is connected by two distinct rocks $r1$ and $r2$, remove the two rocks, add $\min(w_{r1}, w_{r2})$ to both u's and v's dedicated load, and introduce a new pebble of weight $|w_{r1} - w_{r2}|$ between u and v. Let Ψ denote the set of orientations in $G_{\mathbb{R}}$ where each node has at most one incoming edge. We use a proposition to summarize the above discussion.

▶ **Proposition 1.** *We can assume that*

- *the rocks in $\mathbb{R}$ correspond to the edge set of the graph $G_{\mathbb{R}}$, and all pebbles can be assigned to at least two machines;*
- *the graph $G_{\mathbb{R}}$ consists of disjoint trees, cycles (of length more than 2), and isolated nodes;*
- *for each node $v \in V$, $dl(v) \leq t$;*
- *if $\mathrm{OPT} \leq t$, then the orientation of the edges in $G_{\mathbb{R}}$ in the optimal assignment must be one of those in Ψ.*

3 The 2-Valued Case

In this section, we describe the core algorithm for the two job weights case, with the guessed makespan $t \geq W$. Observe that when $t \in [W, 2w)$, if $\mathrm{OPT} \leq t$, then every node can receive at most one job (pebble or rock) in the optimal assignment. Hence, we can solve the problem exactly using the standard max-flow technique. So in the following, assume that $t \geq 2w$.

```
EXPLORE1
Initialize $\mathbb{A} := \{v | v$ is hypercritical, or $v$ is critical in a bad system$\}$.
    Set LEVEL$(v) := 0$ for all nodes in $\mathbb{A}$; $i := 0$.
While $\exists v \notin \mathbb{A}$ reachable from $\mathbb{A}$ do:
    $i := i + 1$.
    $\mathbb{A}_i := \{v \notin \mathbb{A} | v$ reachable from $\mathbb{A}\}$.
    $\mathbb{A}'_i := \{v \notin \mathbb{A} |\ v$ is critical in a good system and $\exists u \in \mathbb{A}_i$ in the same system$\}$.
    Set LEVEL$(v) := i$ for all nodes in $\mathbb{A}_i$ and $\mathbb{A}'_i$.
    $\mathbb{A} := \mathbb{A} \cup \mathbb{A}_i \cup \mathbb{A}'_i$.
For each node $v \notin \mathbb{A}$, set LEVEL$(v) = \infty$.
```

Figure 1 The procedure EXPLORE1.

Furthermore, let us first assume that $t < 2W$ (the case of $t \geq 2W$ will be discussed at the end of the section). Then the rocks have weight W and the pebbles have weight w. Initially, the pebbles are arbitrarily assigned to the nodes. Let $pl(v)$ be the total weight of the pebbles assigned to node v.

▶ **Definition 2.** A node v is
- *uncritical*, if $dl(v) + pl(v) \leq 1.5t - W - w$;
- *critical*, if $dl(v) + pl(v) > 1.5t - W$;
- *hypercritical*, if $dl(v) + pl(v) > 1.5t$.

(Notice that it is possible that a node is neither uncritical nor critical.)

▶ **Definition 3.** Each tree, cycle, or isolated node in $G_{\mathbb{R}}$ is a *system*. A system is *bad* if any of the following conditions holds.
- It is a tree and has at least two critical nodes, or
- It is a cycle and has at least one critical node, or
- It contains a hypercritical node.

A system that is not bad is *good*.

If all systems are good, then orienting the edges in each system such that every node has at most one incoming edge gives us a solution with makespan at most $1.5t$. So let assume that there is at least one bad system.

We next define the *activated set* $\mathbb{A}$ of nodes constructively. Roughly speaking, we will move pebbles around the nodes in $\mathbb{A}$ so that either there is no more bad system left, or we argue that, in every feasible assignment, *some* nodes in $\mathbb{A}$ cannot handle their total loads, thereby arriving at a contradiction.

In the following, if a pebble in u can be assigned to node v, we say v is reachable from u. Node v is reachable from $\mathbb{A}$ if v is reachable from any node $u \in \mathbb{A}$. A node added into $\mathbb{A}$ is *activated*.

Informally, all nodes that cause a system to be bad are activated. A node reachable from $\mathbb{A}$ is also activated. Furthermore, suppose that a system is good and it has a critical node v (thus the system cannot be a cycle). If any other node u in the same system is activated, then so is v. We now give the formal procedure EXPLORE1 in Figure 1. Notice that in the process of activating the nodes, we also define their *levels*, which will be used later for the algorithm and the potential function.

The next proposition follows straightforwardly from EXPLORE1.

▶ **Proposition 4.** *The following holds.*
1. *All nodes reachable from $\mathbb{A}$ are in $\mathbb{A}$.*
2. *Suppose that v is reachable from $u \in \mathbb{A}$. Then* $\textsc{Level}(v) \leq \textsc{Level}(u) + 1$.
3. *If a node v is critical and there exists another node $v' \in \mathbb{A}$ in the same system, then* $\textsc{Level}(v) \leq \textsc{Level}(v')$.
4. *Suppose that node $v \in \mathbb{A}$ has $\textsc{Level}(v) = i > 0$. Then there exists another node $u \in \mathbb{A}$ with $\textsc{Level}(u) = i - 1$ so that either v is reachable from u, or there exists another node $v' \in \mathbb{A}$ reachable from u with $\textsc{Level}(v') = i$ in the same system as v and v is critical.*

After Explore1, we apply the Push operation (if possible), defined as follows.

▶ **Definition 5.** Push operation: push a pebble from u^* to v^* if the following conditions hold.
1. The pebble is at u^* and it can be assigned to v^*.
2. $\textsc{Level}(v^*) = \textsc{Level}(u^*) + 1$.
3. v^* is uncritical, or v^* is in a good system that remains good with an additional weight of w at v^*.
4. Subject to the above three conditions, choose a node u^* so that $\textsc{Level}(u^*)$ is minimized (if there are multiple candidates, pick any).

Our algorithm can be simply described as follows.

Algorithm 1: As long as there is a bad system, apply Explore1 and Push operation repeatedly. When there is no bad system left, return a solution with makespan at most $1.5t$. If at some point, push is no longer possible, declare that OPT $\geq t + 1$.

▶ **Lemma 6.** *When there is at least one bad system and the* Push *operation is no longer possible,* OPT $\geq t + 1$.

Proof. Let $\mathbb{A}(S)$ denote the set of activated nodes in system S. Recall that Ψ denotes the set of all orientations in $G_{\mathbb{R}}$ in which each node has at most one incoming edge. We prove the lemma via the following claim.

▶ **Claim 7.** *Let S be a system.*
- *Suppose that S is bad. Then*

$$W \cdot (\min_{\psi \in \Psi} \textit{number of rocks to } \mathbb{A}(S) \textit{ according to } \psi) + \sum_{v \in \mathbb{A}(S)} pl(v) + dl(v) > |\mathbb{A}(S)|t. \quad (1)$$

- *Suppose that S is good. Then*

$$W \cdot (\min_{\psi \in \Psi} \textit{number of rocks to } \mathbb{A}(S) \textit{ according to } \psi) + \sum_{v \in \mathbb{A}(S)} pl(v) + dl(v) > |\mathbb{A}(S)|t - w. \quad (2)$$

Observe that the term $|\mathbb{A}(S)|t$ is the maximum total weight that all nodes in $\mathbb{A}(S)$ can handle if OPT $\leq t$. As pebbles owned by nodes in $\mathbb{A}$ can only be assigned to the nodes in $\mathbb{A}$, by the pigeonhole principle, in all orientations $\psi \in \Psi$, and all possible assignments of the pebbles, at least one bad system S has at least the same number of pebbles in $\mathbb{A}(S)$ as the current assignment, or a good system S has at least one more pebble than it currently has in $\mathbb{A}(S)$. In both cases, we reach a contradiction.

Proof of Claim 7. First observe that in all orientations in Ψ, the nodes in $\mathbb{A}(S)$ have to receive at least $|\mathbb{A}(S)| - 1$ rocks. If S is a cycle, then the nodes in $\mathbb{A}(S)$ have to receive exactly $|\mathbb{A}(S)|$ rocks.

Next observe that none of the nodes in $\mathbb{A}(S)$ is uncritical, since otherwise, by Proposition 4(4) and Definition 5(3), the PUSH operation would still be possible. By the same reasoning, if S is a tree and $\mathbb{A}(S) \neq \emptyset$, at least one node $v \in \mathbb{A}(S)$ is critical; furthermore, if $|\mathbb{A}(S)| = 1$, this node v satisfies $dl(v) + pl(v) > 1.5t - w$, as an additional weight of w would make v hypercritical. Similarly, if S is an isolated node $v \in \mathbb{A}$, then $dl(v) + pl(v) > 1.5t - w$.

We now prove the claim by the following case analysis.

1. Suppose that S is a good system and $\mathbb{A}(S) \neq \emptyset$. Then either S is a tree and $\mathbb{A}(S)$ contains exactly one critical (but not hypercritical) node, or S is an isolated node, or S is a cycle and has no critical node. In the first case, if $|\mathbb{A}(S)| \geq 2$, the LHS of (2) is at least
$$(1.5t - W + 1) + (|\mathbb{A}(S)| - 1)(1.5t - W - w + 1) + (|\mathbb{A}(S)| - 1)W =$$
$$|\mathbb{A}(S)|t + (|\mathbb{A}(S)| - 2)(0.5t - w + 1) + t - W - w + 2 > |\mathbb{A}(S)|t - w,$$
using the fact that $0.5t \geq w$, $t \geq W$, and $|\mathbb{A}(S)| \geq 2$. If, on the other hand, $|\mathbb{A}(S)| = 1$, then the LHS of (2) is strictly more than
$$1.5t - w \geq t = |\mathbb{A}(S)|t,$$
and the same also holds for the case when S is an isolated node. Finally, in the third case, the LHS of (2) is at least
$$|\mathbb{A}(S)|(1.5t - W - w + 1) + |\mathbb{A}(S)|W > |\mathbb{A}(S)|t.$$
2. Suppose that $\mathbb{A}(S)$ contains at least two critical nodes, or that S is a cycle and $\mathbb{A}(S)$ has at least one critical node. In both cases, S is a bad system. Furthermore, the LHS of (1) can be lower-bounded by the same calculation as in the previous case with an extra term of w.
3. Suppose that $\mathbb{A}(S)$ contains a hypercritical node. Then the system S is bad, and the LHS of (1) is at least
$$(1.5t + 1) + (|\mathbb{A}(S)| - 1)(1.5t - W - w + 1) + (|\mathbb{A}(S)| - 1)W =$$
$$|\mathbb{A}(S)|t + (|\mathbb{A}(S)| - 1)(0.5t - w + 1) + 0.5t + 1 > |\mathbb{A}(S)|t,$$
where the inequality holds because $0.5t \geq w$. ◀

◀

We argue that Algorithm 1 terminates in polynomial time by the aid of a potential function, defined as

$$\Phi = \sum_{v \in \mathbb{A}} (|V| - \textsc{Level}(v)) \cdot (\text{number of pebbles at } v).$$

Trivially, $0 \leq \Phi \leq |V| \cdot |\mathbb{P}|$. The next lemma implies that Φ is monotonically decreasing after each PUSH operation. The proof can be found in the full version.

▶ **Lemma 8.** *For each node $v \in V$, let $\textsc{Level}(v)$ and $\textsc{Level}'(v)$ denote the levels before and after a* PUSH *operation, respectively. Then $\textsc{Level}'(v) \geq \textsc{Level}(v)$.*

Proof. We prove by contradiction. Suppose that there exist nodes x with $\textsc{Level}'(x) < \textsc{Level}(x)$. Choose v to be one among them with minimum $\textsc{Level}'(v)$. By the choice of v, and Definition 5(3), $\textsc{Level}'(v) > 0$ and $v \in \mathbb{A}$ after the PUSH operation. Thus, by Proposition 4(4), there exists a node u with $\textsc{Level}'(u) = \textsc{Level}'(v) - 1$, so that after PUSH,

- Case 1: v is reachable from $u \in \mathbb{A}$, or
- Case 2: there exists another node $v' \in \mathbb{A}$ reachable from $u \in \mathbb{A}$ with $\textsc{Level}'(v') = \textsc{Level}'(v)$ in the same system as v, and v is critical.

Notice that by the choice of v, in both cases, $\textsc{Level}'(u) \geq \textsc{Level}(u)$, and $u \in \mathbb{A}$ also before the Push operation. Let p be the pebble by which u reaches v (Case 1), or v' (Case 2), after Push. Before the Push operation, p was at some node $u' \in \mathbb{A}$ (u' may be u, or p is the pebble pushed: from u' to u).

By Proposition 4(2), in Case 1, $\textsc{Level}(v) \leq \textsc{Level}(u') + 1$ (as v is reachable from u' via p before Push), and $\textsc{Level}(v') \leq \textsc{Level}(u') + 1$ in Case 2. Furthermore, if in Case 2 v was already critical before push, then $\textsc{Level}(v) \leq \textsc{Level}(v')$ by Proposition 4(3) (note that $v' \in \mathbb{A}$ as it is reachable from $u' \in \mathbb{A}$). Hence, in both cases we would have

$$\textsc{Level}(v) \leq \textsc{Level}(u') + 1 \leq \textsc{Level}(u) + 1 \leq \textsc{Level}'(u) + 1 = \textsc{Level}'(v),$$

a contradiction. Note that the second inequality holds no matter $u = u'$ or not.

Finally consider Case 2 where v was not critical before the Push operation. Then a pebble $p' \neq p$ is pushed into v in the operation. Note that in this situation, v's system is a tree and contains no critical nodes before Push (by Definition 5(3)); in particular v' is not critical. Furthermore, the presence of p in u implies that $\textsc{Level}(v') \leq \textsc{Level}(u) + 1$ by Proposition 4(2), and that $v' \in \mathbb{A}$ by Proposition 4(1). As v' is not critical, $\textsc{Level}(v') > 0$, and by Proposition 4(4) there exists a node u'' with $\textsc{Level}(u'') = \textsc{Level}(v') - 1$ so that u'' can reach v' by a pebble p'' (u'' may be u and p'' may be p). As

$$\textsc{Level}(v') \leq \textsc{Level}(u) + 1 \leq \textsc{Level}'(u) + 1 = \textsc{Level}'(v) < \textsc{Level}(v),$$

the Push operation should have pushed p'' into v' instead of p' into v (see Definition 5(4)), since u'' and v' satisfy all the first three conditions of Definition 5. ◀

By Lemma 8 and the fact that a pebble is pushed to a node with higher level, the potential Φ strictly decreases after each Push operation, implying that Algorithm 1 finishes in polynomial time.

Approximation Ratio: When $t < 2W$, we apply Algorithm 1. In the case of $t \geq 2W$, we apply the algorithm of Gairing et al. [5], which either correctly reports that $\text{OPT} \geq t + 1$, or returns an assignment with makespan at most $t + W - 1 < 1.5t$.

Suppose that t is the smallest number for which an assignment is returned. Then $\text{OPT} \geq t$, and our approximation ratio is bounded by $\frac{1.5t}{\text{OPT}} \leq 1.5$. We use a theorem to conclude this section.

▶ **Theorem 9.** *With arbitrary dedicated loads on the machines, jobs of weight W that can be assigned to two machines, and jobs of weight w that can be assigned to any number of machines, we can find a 1.5 approximate solution in polynomial time.*

In the full version, we show that a slight modification of our algorithm yields an improved approximation ratio of $1 + \frac{\lfloor \frac{W}{2} \rfloor}{W}$ if $W \geq 2w$.

4 The General Case

In this section, we describe the core algorithm for the case of arbitrary job weights. This algorithm inherits some basic ideas from the previous section, but has several significantly new ingredients – mainly due to the fact that the rocks now have different weights. Before formally presenting the algorithm, let us build up intuition by looking at some examples.

For simplicity, we rescale the numbers and assume that $t = W = 1$ and $\beta = 0.7$. We aim for an assignment with makespan of at most $5/3 + 0.7/3 = 1.9$ or decide that $\text{OPT} > 1$.

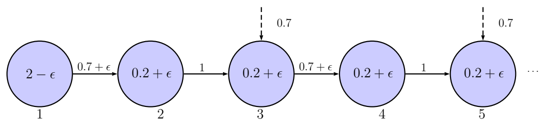

Figure 2 There are $2k+1$ nodes (the rest is repeating the same pattern). Numbers inside the shaded circles (nodes) are their pebble load.

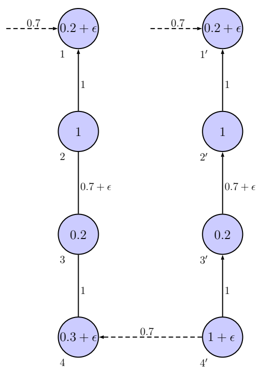

Figure 3 A naive PUSH will oscillate the pebble between nodes 4 and 4′.

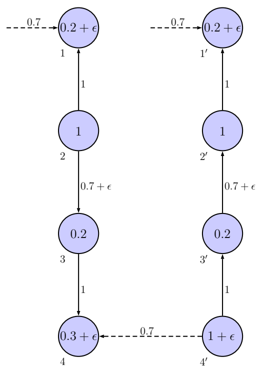

Figure 4 A fake orientation from node 2 to 3 causes node 4 to have an incoming edge, thus informing node 4′ not to push the pebble.

Consider the example in Figure 2. Note that there are $2k+1$ (for some large k) nodes (the pattern of the last two nodes repeats). Due to node 1 (which can be regarded as the analog of a critical node in the previous section), all edges are to be directed toward the right if we shoot for the makespan of 1.9. Suppose that there is an isolated node with the pebble load of $2+\epsilon$ (this node can be regarded as a bad system by itself) and it has a pebble of weight 0.7 that can be assigned to node 3, 5, 7 and so on up to $2k+1$. Clearly, we do not want to push the pebble into any of them, as it would cause the makespan to be larger than 1.9 by whatever orientation. Rather, we should activate node 1 and send its pebbles away with the aim of relieving the "congestion" in the current system (later we will see that this is activation rule 1). In this example, all odd-numbered nodes are activated, and the entire set of nodes (including even-numbered nodes) form a *conflict set* (which will be defined formally later). Roughly speaking, the conflict sets contain activated nodes and the nodes that can be reached by "backtracking" the directed edges from them. These conflict sets embody the "congestion" in the systems.

Recall that in the previous section, if the PUSH operation was no longer possible, we argued that the total load is too much for the activated nodes *system by system*. Analogously, in this example, we need to argue that in all feasible orientations, the activated set of nodes (totally $k+1$ of them) in this conflict set cannot handle the total load. However, if all edges are directed toward the left, their total load is only $(0.2+\epsilon)k+(2-\epsilon)+(0.7+\epsilon)k = 2+0.9k+\epsilon(2k-1)$, which is less than what they can handle (which is $k+1$) when k is large. As a result, we are unable to arrive at a contradiction.

To overcome this issue, we introduce another activation rule to strengthen our contradiction argument. If all edges are directed to the left, *on the average*, each activated node has a total load of about $0.2+0.7$. However, each inactivated node has, *on the average*, a total load of about $0.2+1$. This motivates our activation rule 2 : if an activated node is connected by a "relatively light" edge to some other node in the conflict set, the latter should be activated as well. The intuition behind is that the two nodes *together* will receive a relatively heavy load. We remark that it is easy to modify this example to show that if we do not apply activation rule 2, then we cannot hope for a $2-\delta$ approximation for any small $\delta > 0$. [3]

Next consider the example in Figure 3. Here nodes 2, $2'$, and $4'$ can be regarded as the critical nodes, and $\{1,2\}$, $\{1',2',3',4'\}$ are the two conflict sets. Both nodes 1 and $1'$ can be reached by an isolated node with heavy load (the bad system) with a pebble of weight 0.7. Suppose further that node $4'$ can reach node 4 by another pebble of weight 0.7. It is easy to see that a naive Push definition will simply "oscillate" the pebble between nodes 4 and $4'$, causing the algorithm to cycle.

Intuitively, it is not right to push the pebble from $4'$ into 4, as it causes the conflict set in the left system to become bigger. Our principle of pushing a pebble should be to relieve the congestion in one system, while not worsening the congestion in another. To cope with this problematic case, we use *fake orientations*, i.e., we direct edges away from a conflict set, as shown in Figure 4. Node 2 directs the edge toward node 3, which in turn causes the next edge to be directed toward node 4. With the new incoming edge, node 4 now has a total load of $1 + 0.3 + \epsilon$ to handle, and the pebble thus will not be pushed from node $4'$ to node 4.

4.1 Formal description of the algorithm

We inherit some terminology from the previous section. We say that v is *reachable* from u if a pebble in u can be assigned to v, and that v is reachable from $\mathbb{A}$ if v is reachable from any node $u \in \mathbb{A}$. Each tree, cycle, isolated node in $G_{\mathbb{R}}$ is a system. Note that there is exactly one edge between two adjacent nodes in $G_{\mathbb{R}}$ (see Proposition 1). For ease of presentation, we use the short hand vu to refer to the edge $\{v,u\}$ in $G_{\mathbb{R}}$ and w_{vu} is its weight.

The orientation of the edges in $G_{\mathbb{R}}$ will be decided dynamically. If uv is directed toward v, we call v a *father* of u, and u a *child* of v (notice that a node can have several fathers and children). We write $rl(v)$ to denote total weight of the rocks that are (currently) oriented towards v, and $pl(v)$ still denotes the total weight of the pebbles at v. An edge that is currently un-oriented is *neutral*. In the beginning, all edges in $G_{\mathbb{R}}$ are neutral.

A set $\mathbb{C}$ of nodes, called the *conflict set*, will be collected in the course of the algorithm. Let $\mathcal{D}(v) := \{u \in \mathbb{C} : u \text{ is child of } v\}$ and $\mathcal{F}(v) := \{u \in \mathbb{C} : u \text{ is father of } v\}$ for any $v \in \mathbb{C}$. A node $v \in \mathbb{C}$ is a *leaf* if $\mathcal{D}(v) = \emptyset$, and a *root* if $\mathcal{F}(v) = \emptyset$. Furthermore, a node v is *overloaded* if $dl(v) + pl(v) + rl(v) > (5/3 + \beta/3)t$, and a node $v \in \mathbb{C}$ is *critical* if there exists $u \in \mathcal{F}(v)$ such that $dl(v) + pl(v) + w_{vu} > (5/3 + \beta/3)t$. In other words, a node in the conflict set is critical if it has enough load by itself (without considering incoming rocks) to "force" an incident edge to be directed toward a father in the conflict set.

Initially, the pebbles are arbitrarily assigned to the nodes. The orientation of a subset of the edges in $G_{\mathbb{R}}$ is determined by the procedure Forced Orientations in Figure 5.

Intuitively, the procedure first finds a "source node" v, whose dedicated, pebble, and rock load is so high that it "forces" an incident edge vu to be oriented away from v. The orientation of this edge then propagates through the graph, i.e. edge-orientations induced by

[3] Looking at this particular example, one is tempted to use the idea of activating all nodes in the conflict set. However, such an activation rule will not work. Consider the following example: There are $k+2$ nodes forming a path, and the $k+1$ edges connecting them all have weight $0.95+\epsilon$. The first node has a pebble load of 1 and thus "forces" an orientation of the entire path (for a makespan of at most 1.9). The next k nodes have a pebble load of 0, and the last node has a pebble load of 0.25 and is reachable from a bad system via a pebble of weight 0.7. The conflict set is the entire path, and activating all nodes leads to a total load of $(k+1)\cdot(0.95+\epsilon)+1+0.25$, which is less than $k+2$ for large k.

```
FORCED ORIENTATIONS
While ∃ neutral edge $vu$ in $G_{\mathbb{R}}$, s.t. $dl(v) + pl(v) + rl(v) + w_{vu} > (5/3 + \beta/3)t$:
    Direct $vu$ towards $u$; MARKED := $\{u\}$.
    While ∃ neutral edge $v'u'$ in $G_{\mathbb{R}}$, s.t. $dl(v') + pl(v') + rl(v') + w_{v'u'} > (5/3 + \beta/3)t$ and $v' \in$ MARKED:
        Direct $v'u'$ towards $u'$; MARKED := MARKED $\cup \{u'\}$.
```

Figure 5 The procedure FORCED ORIENTATIONS.

the direction of vu are established. Then the next "source" is found, and so on. To simplify our proofs, we assume that ties are broken according to a fixed total order if several pairs (v, u) satisfy the conditions of the while-loops.

The following lemma describes a basic property of the procedure FORCED ORIENTATIONS, that will be used in the subsequent discussion. Its proof can be found in the full version.

▶ **Lemma 10.** *Suppose that a node v becomes overloaded during* FORCED ORIENTATIONS. *Then there exists a path $u_0u_1 \dots u_kv$ of neutral edges, such that $dl(u_0) + pl(u_0) + rl(u_0) + w_{u_0u_1} > (5/3+\beta/3)t$ before the procedure, that becomes directed from u_0 towards v during the procedure (note that u_0 could be v). Furthermore, other than u_kv, no edge becomes directed toward v in the procedure.*

Proof. We start with a simple observation. Let ab be the first edge directed in some iteration of the procedure's outer while-loop; suppose from a to b. It is easy to see that up to this moment, no edge has been directed toward a in course of the procedure. Furthermore, if another edge $a'b'$ is directed in the same iteration of the outer while-loop, then there exists a path of neutral edges, starting with ab and ending with $a'b'$, that becomes directed during this iteration. This proves the first part of the lemma.

Now suppose that some node v becomes overloaded and has more than one edge directed towards it during the procedure. Let vx and vy be the last two edges directed toward v, and note that both, vx and vy, become directed in the same iteration of the outer while-loop (because as soon as one of the two is directed toward v, the other edge satisfies the conditions of the inner while-loop). Hence, there are two different paths directed towards v (with final edges vx and vy, respectively), both of which start with the first edge that becomes directed in this iteration of the outer while-loop. This is not possible, since every system is a tree or a cycle, a contradiction. ◀

Clearly, if after the procedure FORCED ORIENTATIONS a node v still has a neutral incident edge vu, then $dl(v)+pl(v)+rl(v)+w_{vu} \leq (5/3+\beta/3)t$. Now suppose that after the procedure, none of the nodes is overloaded. Then orienting the neutral edges in each system in such a way that every node has at most one more incoming edge gives us a solution with makespan at most $(5/3 + \beta/3)t$. So assume the procedure ends with a non-empty set of overloaded nodes. We then apply the procedure EXPLORE2 in Figure 6.

Let us elaborate the procedure. In each round, we perform the following three tasks.

1. Add those nodes reachable from the nodes in $\mathbb{A}_{i-1}$ into $\mathbb{A}_i$ in case of $i > 1$; or the overloaded nodes into $\mathbb{A}_i$ in case of $i = 0$. These nodes will be referred to as Type A nodes.
2. In the sub-procedure *Conflict set construction*, nodes not in the conflict set and having a directed path to those Type A nodes in $\mathbb{A}_i$ are continuously added into the conflict set

```
EXPLORE2

Initialize $\mathbb{A} := \emptyset$; $\mathbb{C} := \emptyset$; $i := 0$. Call FORCED ORIENTATIONS.
Repeat:
  If $i = 0$: $\mathbb{A}_i := \{v|v$ is overloaded$\}$.
  Else $\mathbb{A}_i := \{v|v \notin \mathbb{A}, v$ is reachable from $\mathbb{A}_{i-1}\}$.
  If $\mathbb{A}_i = \emptyset$: stop.
  $\mathbb{C}_i := \mathbb{A}_i$; $\mathbb{A} := \mathbb{A} \cup \mathbb{A}_i$; $\mathbb{C} := \mathbb{C} \cup \mathbb{C}_i$.

  (Conflict set construction)
  While $\exists v \notin \mathbb{C}$ with a father $u \in \mathbb{C}$ or $\exists$ neutral $vu$ with $v \in \mathbb{C}$ do:
    While $\exists v \notin \mathbb{C}$ with a father $u \in \mathbb{C}$:
      $\mathbb{C}_i := \mathbb{C}_i \cup \{v\}$; $\mathbb{C} := \mathbb{C} \cup \mathbb{C}_i$.
    If $\exists$ neutral $vu$ with $v \in \mathbb{C}$:
      Direct $vu$ towards $u$; Call FORCED ORIENTATIONS.

  (Activation of nodes)
  While $\exists v \in \mathbb{C} \setminus \mathbb{A}$ satisfying one of the following conditions:
    Rule 1: $\exists u \in \mathcal{F}(v)$, such that $dl(v) + pl(v) + w_{vu} > (5/3 + \beta/3)t$
    Rule 2: $\exists u \in \mathbb{A} \cap (\mathcal{D}(v) \cup \mathcal{F}(v))$, such that $w_{vu} < (2/3 + \beta/3)t$
  Do: $\mathbb{A}_i := \mathbb{A}_i \cup \{v\}$; $\mathbb{A} := \mathbb{A} \cup \mathbb{A}_i$.

  $i := i + 1$.
```

Figure 6 The procedure EXPLORE2.

$\mathbb{C}_i$. Furthermore, the earlier mentioned *fake orientations* are applied: each node $v \in \mathbb{C}_i$, if having an incident neutral edge vu, direct it toward u and call the procedure FORCED ORIENTATIONS. It may happen that in this process, two disjoint nodes in $\mathbb{C}_i$ are now connected by a directed path P, then all nodes in P along with all nodes having a path leading to P are added into $\mathbb{C}_i$ (observe that all these nodes have a directed path to some Type A node in $\mathbb{A}_i$). We note that the order of fake orientations does not materially affect the outcome of the algorithm.

3. In the next sub-procedure *Activation of nodes*, we use two rules to activate extra nodes in $\mathbb{C}\backslash\mathbb{A}$. Rule 1 activates the critical nodes; Rule 2 activates those nodes whose father or child are already activated and they are connected by an edge of weight less than $(2/3+\beta/3)t$. We will refer to the former as Type B nodes and the latter as Type C nodes.

Observe that except in the initial call of FORCED ORIENTATIONS, no node ever becomes overloaded in EXPLORE2 (by Lemma 10 and the fact that every system is a tree or a cycle). Let us define LEVEL$(v) = i$ if $v \in \mathbb{A}_i$. In case $v \notin \mathbb{A}$, let LEVEL$(v) = \infty$. The next proposition summarizes some important properties of the procedure EXPLORE2.

▶ **Proposition 11.** *After the procedure* EXPLORE2, *the following holds.*

1. *All nodes reachable from $\mathbb{A}$ are in $\mathbb{A}$.*
2. *Suppose that $v \in \mathbb{A}$ is reachable from $u \in \mathbb{A}$. Then* LEVEL$(v) \leq$ LEVEL$(u) + 1$.

Furthermore, at the end of each round i, the following holds.

3. *Every node v that can follow a directed path to a node in $\mathbb{C} := \cup_{\tau=0}^{i}\mathbb{C}_\tau$ is in $\mathbb{C}$. Furthermore, if a node $v \in \mathbb{C}$ has an incident edge vu with $u \notin \mathbb{C}$, then vu is directed toward u.*
4. *Each node $v \in \mathbb{A}_i$ is one of the following three types.*

(a) **Type A**: *there exists another node $u \in \mathbb{A}_{i-1}$ so that v is reachable from u, or v is overloaded and is part of $\mathbb{A}_0$.*

(b) **Type B**: *v is activated via Rule 1 (hence v is critical)*[4]*, and there exists a directed path from v to $u \in A_i$ of Type A.*

(c) **Type C**: *v is activated via Rule 2, and there exists an adjacent node $u \in \cup_{\tau=0}^{i} \mathbb{A}_\tau$ so that $w_{vu} < (2/3 + \beta/3)t$ and $u \in \mathcal{D}(v) \cup \mathcal{F}(v)$.*

After the procedure EXPLORE2, we apply the PUSH operation (if possible), defined as follows.

▶ **Definition 12.** PUSH operation: push a pebble from u^* to v^* if the following conditions hold (if there are multiple candidates, pick any).

1. The pebble is at u^* and it can be assigned to v^*.
2. $\text{LEVEL}(v^*) = \text{LEVEL}(u^*) + 1$.
3. $dl(v^*) + pl(v^*) + rl(v^*) \leq (5/3 - 2/3 \cdot \beta)t$.
4. $\mathcal{D}(v^*) = \emptyset$, or $dl(v^*) + pl(v^*) + w_{v^*u} \leq (5/3 - 2/3 \cdot \beta)t$ for all $u \in \mathcal{F}(v)$.

Definition 12(3) is meant to make sure that v^* does not become overloaded after receiving a new pebble (whose weight can be as heavy as βt). Definition 12(4) says either v^* is a leaf, or adding a pebble with weight as heavy as βt does not cause v^* to become critical.

Algorithm 2: Apply EXPLORE2. If it ends with $\mathbb{A}_0 = \emptyset$, return a solution with makespan at most $(5/3 + \beta/3)t$. Otherwise, apply PUSH. If PUSH is impossible, declare that $\text{OPT} \geq t + 1$. Un-orient all edges in $G_{\mathbb{R}}$ and repeat this process.

▶ **Lemma 13.** *When there is at least one overloaded node and the* PUSH *operation is no longer possible,* $\text{OPT} \geq t + 1$.

▶ **Lemma 14.** *For each node $v \in V$, let* $\text{LEVEL}(v)$ *and* $\text{LEVEL}'(v)$ *denote the levels before and after a* PUSH *operation, respectively. Then* $\text{LEVEL}'(v) \geq \text{LEVEL}(v)$.

The proofs of the preceding two lemmas can be found in the full version. We again use the potential function

$$\Phi = \sum_{v \in \mathbb{A}} (|V| - \text{LEVEL}(v)) \cdot (\text{number of pebbles at } v)$$

to argue the polynomial running time of Algorithm 2. Trivially, $0 \leq \Phi \leq |V| \cdot |\mathbb{P}|$. Furthermore, by Lemma 14 and the fact that a pebble is pushed to a node with higher level, the potential Φ strictly decreases after each PUSH operation. This implies that Algorithm 2 finishes in polynomial time.

We can therefore conclude:

▶ **Theorem 15.** *Let $\beta \in [4/7, 1)$. With arbitrary dedicated loads on the machines, if jobs of weight greater than βW can be assigned to only two machines, and jobs of weight at most βW can be assigned to any number of machines, we can find a $5/3 + \beta/3$ approximate solution in polynomial time.*

[4] For simplicity, if a node can be activated by both Rule 1 and Rule 2, we assume it is activated by Rule 1.

References

1 Yuichi Asahiro, Jesper Jansson, Eiji Miyano, Hirotaka Ono, and Kouhei Zenmyo. Approximation algorithms for the graph orientation minimizing the maximum weighted outdegree. *J. Comb. Optim.*, 22(1):78–96, 2011.

2 Deeparnab Chakrabarty, Sanjeev Khanna, and Shi Li. On $(1, \epsilon)$-restricted assignment makespan minimization. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'15)*, pages 1087–1101. SIAM, 2015.

3 T. Ebenlendr, M. Krčál, and J. Sgall. Graph balancing: A special case of scheduling unrelated parallel machines. *Algorithmica*, 68:62–80, 2014.

4 Tomáš Ebenlendr, Marek Krčál, and Jiří Sgall. Graph balancing: A special case of scheduling unrelated parallel machines. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'08)*, pages 483–490. SIAM, 2008.

5 M. Gairing, T. Lücking, M. Mavronicolas, and B. Monien. Computing nash equilibria for scheduling on restricted parallel links. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC'04)*, pages 613–622. ACM, 2004.

6 M. Gairing, B. Monien, and A. Wocalw. A faster combinatorial approximation algorithm for scheduling unrelated parallel machines. *Theor. Comput. Sci.*, 380(1-2):87–99, 2007.

7 Chien-Chung Huang and Sebastian Ott. A combinatorial approximation algorithm for graph balancing with light hyper edges. *CoRR*, abs/1507.07396, 2015.

8 S. G. Kolliopoulos and Y. Moysoglou. The 2-valued case of makespan minimization with assignment constraints. *Information Processing Letters*, 113(1-2):39–43, 2013.

9 J.K. Lenstra, D.B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:256–271, 1990.

10 J.Y. Leung and C. Li. Scheduling with processing set restrictions: A survey. *International Journal of Production Economics*, 116:251–262, 2008.

11 J. Sgall. Private communication, 2015.

12 E. V. Shchepin and N. Vakhania. An optimal rounding gives a better approximation for scheduling unrelated machines. *Oper. Res. Lett.*, 33(2):127–133, 2005.

13 O. Svensson. Santa claus schedules jobs on unrelated machines. *SIAM J. Comput.*, 41(5):1318–1341, 2012.

14 J. Verschae and A. Wiese. On the configuration-lp for scheduling on unrelated machines. *J. Scheduling*, 17(4):371–383, 2014.

15 D. P. Willamson and D.B. Shmoys. *The design of approximation algorithms.* Cambridge University Press, 2010.