A Combinatorial Approximation Algorithm for Graph Balancing with Light Hyper Edges

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

[1]Chalmers University, Göteborg, Sweden, villars@gmail.com
[2]Max Planck Institute for Informatics, 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 [7], 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 [1, 3, 10] recently.

In this paper, we consider a special case called *graph-balancing with light hyper edges*: jobs with weights in the range of $(\beta W, W]$ can be assigned to at most two machines, where $4/7 \leq \beta < 1$, and W is the largest job weight, while there is no restriction on the number of machines a job can be assigned to if its weight is in the range of $(0, \beta W]$. Our main contribution is a $5/3 + \beta/3$ approximation algorithm, thus breaking the barrier of 2 in this special case. Unlike the several recent works [1, 3, 10] that make use of (configuration)-LPs, our algorithm is purely combinatorial.

In addition, we consider another special case where jobs have only two possible weights $\{w, W\}$, and $w < W$. Jobs with weight W can be assigned to only two machines, while there is no restriction on the others. For this special case, we give a 1.5-approximation algorithm, thus matching the general lower bound (indeed the current 1.5 lower bound is established in an even more restricted setting). Interestingly, depending on the specific values of w and W, sometimes our algorithm guarantees sub-1.5 approximation ratios.

1 Introduction

Let $\mathcal{J}$ be a set of jobs and $\mathcal{M}$ a set of 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 [7] 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 [12] lists this as one of the ten open problems.

In this paper, we consider a special case of RESTRICTED ASSIGNMENT and show that it is possible to achieve the ratio strictly better than 2. Before we formally introduce our problem and result, we first highlight several recent major advances in the problem (other related work will be discussed later), so as to put our work in a better context.

Ebenlendr, Křćal and Sgall [3] considered the GRAPH-BALANCING PROBLEM (of which our problem is a generalization), where each job can be assigned to only two machines. They strengthened the LP of Lenstra et al. [7] and designed a 1.75-approximation algorithm. For general RESTRICTED ASSIGNMENT, Svensson [10] showed that the integrality gap of the configuration-LP is at most $33/17 \approx 1.941$. Chakrabarty, Khanna, and Li [1] very recently showed that using the configuration-LP, they can obtain a $(2-\delta)$-approximation, for a certain fixed $\delta > 0$, assuming that there are only two job weights (but there is no limitation on the number of the machines a job can be assigned to).

Our Result

We call our problem GRAPH-BALANCING WITH LIGHT HYPER EDGES. We assume that all job weights w_j are integral (this assumption is just for ease of exposition and can be easily removed) and the largest given weight is W. Furthermore, let $\beta \in [4/7, 1)$. A job is *heavy* if its weight $w_j \in (\beta W, W]$, otherwise, it is *light*. A heavy job can be assigned to only two machines while a light job can be assigned to any number of machines.[1] Our problem thus can be interpreted in a graph-theoretic way: each node represents a machine; a light job is a hyper edge and a heavy job a regular edge. The goal is to find an orientation of the edges so that the maximum sum of the edges oriented towards a node is minimized.

The main contribution of this paper is a $5/3+\beta/3$ approximation algorithm for this problem, thus breaking the barrier of 2 in a special case of RESTRICTED ASSIGNMENT. 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.

We emphasize that our algorithm is purely combinatorial, in contrast to the three aforementioned results [1, 3, 10] that make use of (configuration-)LPs, thus showing that it is possible to break the barrier of 2 using a non LP-based approach. Our result raises the question whether it is possible to use other combinatorial techniques to circumvent the limitation posed by the integrality gaps of these LPs.

[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.

In addition, we consider the more special case where there are only two job weights $\{w, W\}$, $w < W$. Jobs with larger weights W can be assigned to only two machines, while there is no limitation on the rest. We give a combinatorial algorithm to achieve the ratio of 1.5, matching the general lower bound of the restricted assignment problem. Note that Ebenlendr et al. [2] show that this lower bound holds even if there are only two jobs weights and each job can be assigned to only two machines—an even more restricted setting than ours. They also showed that for two job weights and dedicated loads in the GRAPH-BALANCING PROBLEM, the integrality gap of their strongest LP is 1.75. Thus, our combinatorial approach beats the integrality gap in a slightly more general setting.

Interestingly, depending on the given value of w and W, sometimes our algorithm achieves the ratio *strictly better* than 1.5. Supposing that $w \leq \frac{W}{2}$, the ratio we get is $1 + \frac{\lfloor W/2 \rfloor}{W}$.

Our Technique

Our approach is inspired by that of Gairing et al. [4] 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 $\mathrm{OPT} \geq W$, they guarantee the approximation ratio of $\frac{t+W-1}{\mathrm{OPT}} \leq 1 + \frac{W-1}{\mathrm{OPT}} \leq 2 - \frac{1}{W}$ (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., $\mathrm{OPT} \geq t + 1$. 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 $\mathrm{OPT} \geq t + 1$, 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 $\mathrm{OPT} \leq t$, 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.

Related Work

For RESTRICTED ASSIGNMENT, besides the several recent advances mentioned earlier, see the survey of Leung and Li for other special cases [8]. Kolliopoulos and Moysoglou [6] also considered the two job weights case. In the GRAPH-BALANCING PROBLEM (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 configuration-LP for RESTRICTED ASSIGNMENT with two job weights has an integrality gap of at most 1.883 (and this is further improved to 1.833 in [1]). We note that so far there is no known instance for which the integrality gap of the configuration-LP is higher than 1.5.

For the UNRELATED MACHINE PROBLEM, Shchepin and Vakhania [9] improved the approximation ratio to $2 - 1/|\mathcal{M}|$. A combinatorial 2-approximation algorithm was given by Gairing, Monien, and Woclaw [5]. Verschae and Wiese [11] 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 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 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 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 $t \geq 2w$. 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

EXPLORE1

Initialize $\mathbb{A} := \{v | v \text{ is hypercritical, or } v \text{ is critical in a bad system}\}$.
Set $\text{LEVEL}(v) := 0$ for all nodes in $\mathbb{A}$; $i := 0$.

While $\exists v \notin \mathbb{A}$ reachable from $\mathbb{A}$ **do:**
$i := i + 1$.
$\mathbb{A}_i := \{v \notin \mathbb{A} | v \text{ reachable from } \mathbb{A}\}$.
$\mathbb{A}'_i := \{v \notin \mathbb{A} |\ v \text{ is critical in a good system and } \exists u \in \mathbb{A}_i \text{ in the same system}\}$.
Set $\text{LEVEL}(v) := i$ 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 $\text{LEVEL}(v) = \infty$.

Figure 1: The procedure EXPLORE1.

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 $\text{LEVEL}(v) \leq \text{LEVEL}(u) + 1$.*
3. *If a node v is critical and there exists another node $v' \in \mathbb{A}$ in the same system, then $\text{LEVEL}(v) \leq \text{LEVEL}(v')$.*
4. *Suppose that node $v \in \mathbb{A}$ has $\text{LEVEL}(v) = i > 0$. Then there exists another node $u \in \mathbb{A}$ with $\text{LEVEL}(u) = i - 1$ so that either v is reachable from u, or there exists another node $v' \in \mathbb{A}$ reachable from u with $\text{LEVEL}(v') = i$ 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. *$\text{LEVEL}(v^*) = \text{LEVEL}(u^*) + 1$.*
3. *v^* is uncritical, or v^* is in a good system and adding the pebble into v^* does not turn this system bad.*
4. *Subject to the above three conditions, choose a node u^* so that $\text{LEVEL}(u^*)$ 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 $\text{OPT} \geq t + 1$.

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

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 one more pebble 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

$$\begin{aligned}(1.5t - W + 1) + (|\mathbb{A}(S)| - 1)(1.5t - W - w + 1) + (|\mathbb{A}(S)| - 1)W = \\ |\mathbb{A}(S)|t + (|\mathbb{A}(S)| - 2)(0.5t - w + 1) + t - W - w + 2 > |\mathbb{A}(S)|t - w,\end{aligned}$$

   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

$$\begin{aligned}(1.5t + 1) + (|\mathbb{A}(S)| - 1)(1.5t - W - w + 1) + (|\mathbb{A}(S)| - 1)W = \\ |\mathbb{A}(S)|t + (|\mathbb{A}(S)| - 1)(0.5t - w + 1) + 0.5t + 1 > |\mathbb{A}(S)|t,\end{aligned}$$

   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.

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 (since otherwise the operation would turn the system bad); 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. [4], which either correctly reports that 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 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 appendix, 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 OPT > 1. 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 tree* (which will be defined formally later). Roughly speaking, the conflict trees contain activated nodes and the nodes that can be reached by "backtracking" the directed edges from them. These conflict trees embody the "congestion" in the systems.

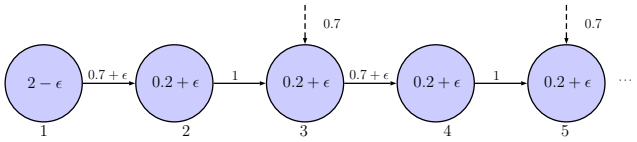

Figure 2: There are $2k+1$ nodes (the rest is repeating the same pattern). Numbers inside the shaded circles (nodes) are their pebble load.

Recall that in the previous section, if the PUSH operation was no longer possible, we argued that the total load is too much (see the proof of Lemma 6) 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 tree 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 tree, 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

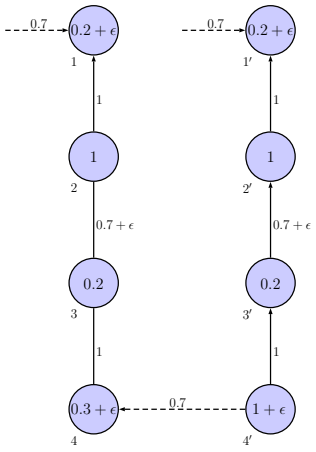

Figure 3: A naive PUSH will oscillate the pebble between nodes 4 and 4′.

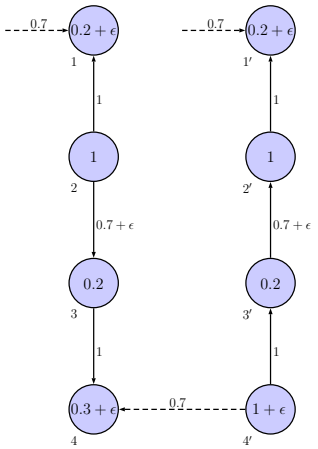

Figure 4: A fake orientation from node 2 to 3 causes node 4 to have an incoming edge, thus informing node 4′ not to push the pebble.

rule 2, then we cannot hope for a $2-\delta$ approximation for any small $\delta > 0$. [2]

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 trees. 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 tree on 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 tree, 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. 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.

[2] Looking at this particular example, one is tempted to use the idea of activating all nodes in the conflict tree. 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 tree 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.

A set $\mathbb{C}$ of nodes, called the *conflict trees*, 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 trees 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 trees.

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.

Forced Orientations

While $\exists$ 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 $\exists$ 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.

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 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.

Lemma 10. *Suppose that all edges in $G_{\mathbb{R}}$ are neutral and* Forced Orientations *is called. Then after the procedure, every overloaded node v has at most one incoming edge.*

Proof. We start with the following observation. Let vu be the first edge directed in some iteration of the outer while-loop. If $v'u'$ is directed in the same iteration of the outer while-loop, then there exists a directed path starting with the edge uv and ending with the edge $v'u'$.

Now suppose that some node w is overloaded after the procedure and has more than one incoming edge. Let wx and wy be the last two edges directed toward w, and note that both, wx and wy, become directed in the same iteration of the outer while-loop (because as soon as one of the two is directed toward w, the other edge satisfies the conditions of the inner while-loop). Hence, there are two different directed paths towards w (with final edges wx and wy, respectively), both of which start with the first edge that becomes directed in this iteration of the outer while-loop. This implies that w's system contains a cycle and a node of degree at least 3, 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 *Constructing conflict trees*, nodes not in the conflicts trees and having a directed path to those Type A nodes in $\mathbb{A}_i$ are continuously added into the conflict trees

EXPLORE2

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

(*Constructing conflict trees*)
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 (see Lemma 30).

3. In the next sub-procedure *Activation of nodes*, we use two rules to activate extra nodes in $\mathbb{C}\setminus\mathbb{A}$. 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 a similar argument as in Lemma 10). Let us define $\text{LEVEL}(v) = i$ if $v \in \mathbb{A}_i$. In case $v \notin \mathbb{A}$, let $\text{LEVEL}(v) = \infty$. 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 $\text{LEVEL}(v) \leq \text{LEVEL}(u) + 1$.*

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)*[3]*, 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. $\textsc{Level}(v^*) = \textsc{Level}(u^*) + 1$.
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* $\textsc{Level}(v)$ *and* $\textsc{Level}'(v)$ *denote the levels before and after a* PUSH *operation, respectively. Then* $\textsc{Level}'(v) \geq \textsc{Level}(v)$.

The preceding two lemma are proven in sections 4.2 and 4.3, respectively. We again use the potential function

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

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.*

4.2 Proof Of Lemma 13

Our goal is to show that in any feasible solution, the activated nodes $\mathbb{A}$ must handle a total load of more than $|\mathbb{A}|t$, which implies that $\text{OPT} \geq t+1$. For the proof, we focus on a single component K of $G_{\mathbb{R}}[\mathbb{C}]$, the subgraph of $G_{\mathbb{R}}$ induced by the conflict trees $\mathbb{C}$, and a fixed orientation $\psi \in \Psi$. Let $\psi(v)$ denote the total weight of the rocks assigned to any $v \in \mathbb{A}$ by ψ (note that $0 \leq \psi(v) \leq t$), and let $\mathbb{A}(K)$ denote the set of activated nodes in K. We will show that

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

$$\sum_{v \in \mathbb{A}(K)} pl(v) + dl(v) + \psi(v) > |\mathbb{A}(K)|t \tag{3}$$

if $\mathbb{A}(K) \neq \emptyset$. The lemma then follows by summing over all components of $G_{\mathbb{R}}[\mathbb{C}]$, and noting that the pebbles on the nodes in $\mathbb{A}$ can only be assigned to the nodes in $\mathbb{A}$ (Proposition 11(1)).

If K consists only of a single activated node v, then (3) clearly holds, as $pl(v) + dl(v) > (5/3 - 2/3 \cdot \beta)t \geq t$ (since v is a Type A node and PUSH is no longer possible). In the following, we will assume that $\mathcal{F}(v) \cup \mathcal{D}(v) \neq \emptyset$ for all $v \in \mathbb{A}(K)$.

Definition 16. *For every non-leaf $v \in \mathbb{A}(K)$, fix some node $d(v) \in \mathcal{D}(v)$, such that $w_{vd(v)} = \max_{u \in \mathcal{D}(v)} w_{vu}$.*

Definition 17. *For every non-root $v \in \mathbb{A}(K)$, fix some node $f(v) \in \mathcal{F}(v)$, such that $w_{vf(v)} = \max_{u \in \mathcal{F}(v)} w_{vu}$.*

Definition 18. *For every node $v \in \mathbb{A}(K)$ that is neither a root nor a leaf, fix some node $n(v) \in \mathcal{D}(v) \cup \mathcal{F}(v)$, such that $w_{vn(v)} = \max_{u \in \mathcal{D}(v) \cup \mathcal{F}(v)} w_{vu}$.*

Definition 19. *For every node $v \in \mathbb{A}(K)$ that was activated using Rule 2 in the final execution of* EXPLORE2*, fix some node $a(v) \in \mathbb{A}(K) \cap (\mathcal{D}(v) \cup \mathcal{F}(v))$ with $w_{va(v)} < (2/3 + \beta/3)t$, such that $a(v)$ has been activated before v.*

We classify the nodes $v \in \mathbb{A}(K)$ that are neither a root nor a leaf, into the following three types.

Type 1: $|\mathcal{D}(v)| > 1$.

Type 2: $|\mathcal{D}(v)| = 1$ and v was activated via Rule 2 (i.e., as a Type C node).

Type 3: $|\mathcal{D}(v)| = 1$ and v was not activated via Rule 2 (i.e. as a Type A or Type B node).

In the following, we summarize the inequalities that we use for the different types of nodes, in order to prove (3). We refer to them as the *load-inequalities*.

Claim 20. *For every leaf $v \in \mathbb{A}(K)$, $pl(v) + dl(v) > (5/3 + \beta/3)t - w_{vf(v)}$.*

Proof. If $v \in \mathbb{A}_i$ is activated as a Type A node, then it is either overloaded or is reachable from a node $u \in \mathbb{A}_{i-1}$. In both cases, since PUSH is no longer possible, $pl(v) + dl(v) + rl(v) > (5/3 - 2/3 \cdot \beta)t$. The claim follows as $rl(v) = 0$ and $w_{vf(v)} > \beta t$. If v is not activated as a Type A node, then v first becomes part of $\mathbb{C}$ and then becomes activated via Rule 1 or Rule 2. In this case, at the moment v becomes part of $\mathbb{C}$, it must have a father $u \in \mathbb{C}$. The edge vu becomes oriented towards u only when FORCED ORIENTATIONS is called and $dl(v) + pl(v) + rl(v) + w_{vu} > (5/3 + \beta/3)t$. The claim follows again as $rl(v) = 0$ and $w_{vf(v)} \geq w_{vu}$. □

Claim 21. *For every root $v \in \mathbb{A}(K)$ with $|\mathcal{D}(v)| = 1$, $pl(v) + dl(v) > (5/3 - 2/3 \cdot \beta)t - w_{vd(v)}$.*

Proof. As $v \in \mathbb{A}_i$ has no father in $\mathbb{C}$, it must either be overloaded or reachable from an activated node $u \in \mathbb{A}_{i-1}$. In both cases, $pl(v) + dl(v) + rl(v) > (5/3 - 2/3 \cdot \beta)t$, since the PUSH operation is no longer possible. The claim follows as $|\mathcal{D}(v)| = 1$ implies $w_{vd(v)} \geq rl(v)$. □

Claim 22. *For every root $v \in \mathbb{A}(K)$ with $|\mathcal{D}(v)| > 1$, $pl(v) + dl(v) \geq 0$.*

Proof. Trivially true. □

Claim 23. *For every Type 1 node $v \in \mathbb{A}(K)$, $pl(v) + dl(v) \geq 0$.*

Proof. Trivially true. □

Claim 24. *For every Type 2 node $v \in \mathbb{A}(K)$, $pl(v) + dl(v) > (5/3 + \beta/3)t - w_{vf(v)} - w_{vd(v)}$.*

Proof. As v is activated using Rule 2, it first becomes part of $\mathbb{C}$ without being activated. For this to happen, it must have a father $u \in \mathbb{C}$. The edge vu becomes oriented towards u only when FORCED ORIENTATIONS is called and $dl(v) + pl(v) + rl(v) + w_{vu} > (5/3 + \beta/3)t$. The claim follows as $w_{vd(v)} \geq rl(v)$ (since $|\mathcal{D}(v)| = 1$) and $w_{vf(v)} \geq w_{vu}$. □

Claim 25. *For every Type 3 node $v \in \mathbb{A}(K)$, $pl(v) + dl(v) > (5/3 - 2/3 \cdot \beta)t - w_{vn(v)}$.*

Proof. If v is overloaded, the claim directly follows from the fact that $w_{vn(v)} \geq rl(v)$. Furthermore, if $v \in A_i$ is reachable from an activated node $u \in A_{i-1}$, then the claim follows from the definition of $n(v)$ and the fact that either the third or the fourth condition of PUSH must be violated. The only other possibility for v to be activated is via Rule 1, which together with the definition of $n(v)$ implies our claim. □

To prove (3), we look at each node $v \in \mathbb{A}(K)$ separately and calculate how much it contributes to the balance under some simplifying assumptions. In the end, we will see that the nodes in $\mathbb{A}(K)$ have enough load to compensate for the assumptions we made.

Let $E_{\mathbb{A}(K)}$ denote the edges of K that are incident with the nodes $\mathbb{A}(K)$, i.e. $E_{\mathbb{A}(K)} := \{vu \in \mathbb{R} : u \in \mathbb{A}(K), v \in \mathcal{D}(u) \cup \mathcal{F}(u)\}$. We say that an edge $vu \in E_{\mathbb{A}(K)}$ is *covered* if w_{vu} appears on the right-hand side of u's and/or v's load-inequality. For example, if v is a leaf, then $vf(v)$ is covered. Every edge in $E_{\mathbb{A}(K)}$ that is not covered is called *uncovered*. Finally, we say that an edge $vu \in E_{\mathbb{A}(K)}$ is *doubly covered* if w_{vu} appears on the right-hand side of both u's and v's load-inequality.

We distinguish two cases.

4.2.1 Case 1: K is a tree.

Claim 26. *K contains $1+\sum_{v \in K : \mathcal{F}(v) \neq \emptyset}(|\mathcal{F}(v)|-1)$ many roots, and $1+\sum_{v \in K : \mathcal{D}(v) \neq \emptyset}(|\mathcal{D}(v)|-1)$ many leaves. Furthermore, every root and leaf in K is activated.*

Proof. The first part simply follows from the degree sum formula for directed graphs and the fact that K is a tree. For the second part, observe that any node $v \in \mathbb{C}$ that is not activated as Type A node, must have had a father $u \in \mathbb{C}$ already before it got added into $\mathbb{C}$ itself. This proves that every root in K is activated (as a Type A node).

If a leaf $v \in \mathbb{C}$ is not activated as Type A node, then its incident edge vu with $u \in \mathbb{C}$ is oriented toward u only when FORCED ORIENTATIONS is called and $dl(v) + pl(v) + rl(v) + w_{vu} > (5/3 + \beta/3)t$. As $v \in \mathbb{C}$ ends up a leaf, $rl(v) = 0$, and Rule 1 would have applied to v. So every leaf in K is activated. □

In our calculations, we will assume that every covered edge $vu \in E_{\mathbb{A}(K)}$ has weight $w_{vu} = t$, and that $\psi(v) = t$ for all $v \in \mathbb{A}(K)$. With these assumptions, we will show that

$$\begin{aligned} \sum_{v \in \mathbb{A}(K)} pl(v) + dl(v) + \psi(v) > |\mathbb{A}(K)|t \\ - |\{\text{doubly covered } vu \in E_{\mathbb{A}(K)} : w_{vu} < (2/3 + \beta/3)t\}| \cdot (1/3 - \beta/3)t \\ + |\{\text{uncovered } vu \in E_{\mathbb{A}(K)}\}| \cdot (t - w_{vu}) \\ + t. \end{aligned} \tag{4}$$

Let us consider the error caused by these two assumptions when we lower-bound the term $\sum_{v \in \mathbb{A}(K)} pl(v) + dl(v) + \psi(v)$, and in doing so, we will show why (4) implies (3).

Consider an edge $vu \in E_{\mathbb{A}(K)}$ that ψ assigns to a node in $\mathbb{A}(K)$, say v. Consider three possibilities.

- If vu is covered, then w_{vu} appears on the LHS of (3) as a negative term after we plug in the load-inequalities, and the two terms $\psi(v)$ and w_{vu} cancel each other. Hence, in this case, we make no error by assuming both terms to be equal to t.

- If vu is doubly covered and $w_{vu} < (2/3 + \beta/3)t$, our assumptions underestimate the load $\sum_{v \in \mathbb{A}(K)} pl(v) + dl(v) + \psi(v)$ by more than $(1/3 - \beta/3)t$.
- If vu is uncovered, then we overestimate $\psi(v)$ by at most $t - w_{vu}$.

Finally, we note that ψ must assign an edge from $E_{\mathbb{A}(K)}$ to every node in $\mathbb{A}(K)$ except for possibly one. For this special node v^* that does not receive an edge from $E_{\mathbb{A}(K)}$ under ψ, we overestimate $\psi(v^*)$ by at most t. In conclusion, when we remove our assumptions, $\sum_{v \in \mathbb{A}(K)} pl(v) + dl(v) + \psi(v)$ increases by more than $(1/3 - \beta/3)t$ per doubly covered edge $vu \in E_{\mathbb{A}(K)}$ with $w_{vu} < (2/3 + \beta/3)t$, and decreases by at most $t - w_{vu}$ per uncovered edge $vu \in E_{\mathbb{A}(K)}$, plus possibly another t for the special node v^*. Hence, if we prove inequality (4) under the aforementioned assumptions, (3) must hold after we remove the assumptions, and Lemma 13 would follow.

We now turn to proving (4) when every covered edge $vu \in E_{\mathbb{A}(K)}$ has weight $w_{vu} = t$, and $\psi(v) = t$ for all $v \in \mathbb{A}(K)$. To this end, we consider the value $pl(v) + dl(v) + \psi(v)$ as a *budget* of node v. Furthermore, we also assign budgets to edges $vu \in E_{\mathbb{A}(K)}$ that are doubly covered and have weight $w_{vu} < (2/3 + \beta/3)t$. Each of them gets a budget of $(1/3 - \beta/3)t$. Other remaining edges of $E_{\mathbb{A}(K)}$ have budget 0.

By redistributing budgets between nodes and edges, we will ensure that eventually

(i) every node in $\mathbb{A}(K)$ has a budget of at least t,

(ii) there exists a leaf in $\mathbb{A}(K)$ with budget strictly greater than $t + (2/3 + \beta/3)t$,

(iii) there exists a root in $\mathbb{A}(K)$ with budget at least $t + (2/3 - 2/3 \cdot \beta)t$,

(iv) every uncovered edge $vu \in E_{\mathbb{A}(K)}$ has a budget of at least $t - w_{vu}$, and

(v) no edge in $E_{\mathbb{A}(K)}$ has negative budget.

This would complete the proof.

We start with the leaf nodes. If $v \in \mathbb{A}(K)$ is a leaf, then (using Claim 20) it has a budget of more than $(5/3 + \beta/3)t - w_{vf(v)} + \psi(v) = (5/3 + \beta/3)t$. Using Claim 26, we can therefore add $(|\mathcal{D}(u)| - 1) \cdot (2/3 + \beta/3)t$ to the budget of every non-leaf $u \in \mathbb{A}(K)$, such that (i) and (ii) are still satisfied for all leaves.

Next we consider the roots. If $v \in \mathbb{A}(K)$ is a root and $|\mathcal{D}(v)| = 1$, then (using Claim 21) it has a budget of more than $(5/3 - 2/3 \cdot \beta)t$. If $v \in \mathbb{A}(K)$ is a root and $|\mathcal{D}(v)| > 1$, then (using Claim 22 and the load added in the previous step) it has a budget of at least $t + (|\mathcal{D}(v)| - 1) \cdot (2/3 + \beta/3)t$. In the latter case, we transfer $(2/3 - 2/3 \cdot \beta)t$ to the budget of every edge in $E_{\mathbb{A}(K)}$ that is incident with v. The budget of v thereby remains at least $t + (|\mathcal{D}(v)| - 1) \cdot (2/3 + \beta/3)t - |\mathcal{D}(v)| \cdot (2/3 - 2/3 \cdot \beta)t = (1/3 - \beta/3)t + |\mathcal{D}(v)| \cdot \beta t \geq (5/3 - 2/3 \cdot \beta)t$, where the last inequality follows from $|\mathcal{D}(v)| \geq 2$ and $\beta \geq 4/7$. Using Claim 26, we can thus add $(|\mathcal{F}(u)| - 1) \cdot (2/3 - 2/3 \cdot \beta)t$ to the budget of every non-root $u \in \mathbb{A}(K)$, such that (i) and (iii) are satisfied for all roots.

Before we move on to Type 1, 2, and 3 nodes, we take one step back and visit the leaves again, as their budget has increased again through the latest redistribution of load. Namely, every leaf $v \in \mathbb{A}(K)$ got an additional load of $(|\mathcal{F}(v)| - 1) \cdot (2/3 - 2/3 \cdot \beta)t$, which we now use to add $(2/3 - 2/3 \cdot \beta)t$ to the budget of every edge in $E_{\mathbb{A}(K)}$ that is incident with v, except to $vf(v)$ (which is surely covered). After this, (ii) and (iii) are satisfied, (i) holds for every root and every leaf, and every uncovered edge $vu \in E_{\mathbb{A}(K)}$ that is incident with a root or a leaf has a budget of at least $(2/3 - 2/3 \cdot \beta)t$.

Let us now consider the nodes of Type 1. Such a node v (using Claim 23 and the load added in previous steps) has a budget of at least $t + (|\mathcal{D}(v)| - 1) \cdot (2/3 + \beta/3)t + (|\mathcal{F}(v)| - 1) \cdot (2/3 - 2/3 \cdot \beta)t$. We transfer $(2/3 - 2/3 \cdot \beta)t$ to the budget of every edge in $E_{\mathbb{A}(K)}$ that is incident with v. Since there are $|\mathcal{D}(v)| + |\mathcal{F}(v)|$ such edges, the budget at v remains at least $t + (|\mathcal{D}(v)| - 1) \cdot (2/3 + \beta/3)t - (|\mathcal{D}(v)| + 1) \cdot (2/3 - 2/3 \cdot \beta)t = (|\mathcal{D}(v)| + 1)\beta t - (1/3 + 2/3 \cdot \beta)t \geq t$, as $|\mathcal{D}(v)| \geq 2$ and $\beta \geq 4/7$.

Next we consider the nodes of Type 2. Such a node v (using Claim 24 and the load added in previous steps) has a budget of more than $(2/3 + \beta/3)t + (|\mathcal{F}(v)| - 1) \cdot (2/3 - 2/3 \cdot \beta)t$. We

transfer $(2/3 - 2/3 \cdot \beta)t$ to the budget of every edge in $E_{\mathbb{A}(K)}$ that is incident with v, except to $vf(v)$ and $vd(v)$ (which are surely covered). Since there are $|\mathcal{F}(v)| - 1$ such edges, the resulting budget at v is still more than $(2/3 + \beta/3)t$. We now reduce the budget of the edge $va(v)$ by $(1/3 - \beta/3)t$ and add this load to v's budget, which is then more than t. We will show later that this last step (reducing the budget of $va(v)$) does not cause a violation of (v).

Finally, we consider the nodes of Type 3. Such a node v (using Claim 25 and the load added in previous steps) has a budget of more than $(5/3 - 2/3 \cdot \beta)t + (|\mathcal{F}(v)| - 1) \cdot (2/3 - 2/3 \cdot \beta)t$. We transfer $(2/3 - 2/3 \cdot \beta)t$ to the budget of every edge in $E_{\mathbb{A}(K)}$ that is incident with v, except to $vn(v)$ (which is surely covered). Since there are $|\mathcal{F}(v)|$ such edges, the resulting budget at v is still more than t.

After the above redistributions of load, (i), (ii), and (iii) are satisfied. Furthermore, suppose that some edge $vu \in E_{\mathbb{A}(K)}$ is uncovered and has weight $w_{vu} \geq (2/3 + \beta/3)t$. Then at least once, we have added $(2/3 - 2/3 \cdot \beta)t$ to the budget of this edge, and we never reduced it. Therefore it has a budget of at least $(2/3 - 2/3 \cdot \beta)t \geq (1/3 - \beta/3)t \geq t - w_{vu}$, and (iv) holds for this edge. If, on the other hand, an uncovered edge $vu \in E_{\mathbb{A}(K)}$ has weight $w_{vu} < (2/3 + \beta/3)t$, then both u and v are in $\mathbb{A}(K)$ (due to activation rule 2), and $(2/3 - 2/3 \cdot \beta)t$ was added twice to the budget of vu. Furthermore, if this budget got reduced at some point, then at most once ($u = a(v)$ and $v = a(u)$ cannot happen simultaneously). The final budget of vu is thus at least $2 \cdot (2/3 - 2/3 \cdot \beta)t - (1/3 - \beta/3)t = t - \beta t > t - w_{vu}$. Hence, for such an edge the assertion (iv) also holds.

Finally, for (v), observe that the only point where we reduce the budget of a covered edge $vu \in E_{\mathbb{A}(K)}$ and add it to v's budget, is when v is of Type 2, $w_{vu} < (2/3 + \beta/3)t$, and $u = a(v)$. Furthermore, both u and v have to be in $\mathbb{A}(K)$ (due to activation rule 2). In this case, the budget of vu is reduced exactly once, by a value of $(1/3 - \beta/3)t$. If vu is doubly covered, then it had an initial budget of $(1/3 - \beta/3)t$, and its budget therefore remains non-negative. If, on the other hand, vu is covered but not doubly covered, then at some point its budget was increased by $(2/3 - 2/3 \cdot \beta)t$. Hence, the final budget is at least $(2/3 - 2/3 \cdot \beta)t - (1/3 - \beta/3)t = (1/3 - \beta/3)t \geq 0$. This concludes the proof.

4.2.2 Case 2: K is a cycle.

Claim 27. *K contains $\sum_{v \in K : \mathcal{F}(v) \neq \emptyset} (|\mathcal{F}(v)| - 1)$ many roots, and $\sum_{v \in K : \mathcal{D}(v) \neq \emptyset} (|\mathcal{D}(v)| - 1)$ many leaves. Furthermore, every root and leaf in K is activated.*

Proof. The first part simply follows from the degree sum formula for directed graphs and the fact that K is a cycle. The second part is analogous to Claim 26. □

We will again assume that every covered edge $vu \in E_{\mathbb{A}(K)}$ has weight $w_{vu} = t$, and that $\psi(v) = t$ for all $v \in \mathbb{A}(K)$. With these assumptions, we will show that

$$\begin{aligned} \sum_{v \in \mathbb{A}(K)} pl(v) + dl(v) + \psi(v) > |\mathbb{A}(K)|t \\ - |\{\text{doubly covered } vu \in E_{\mathbb{A}(K)} : w_{vu} < (2/3 + \beta/3)t\}| \cdot (1/3 - \beta/3)t \\ + |\{\text{uncovered } vu \in E_{\mathbb{A}(K)}\}| \cdot (t - w_{vu}). \end{aligned} \tag{5}$$

By the same arguments as in Case 1, the error caused by the above two assumptions when we lower-bound the term $\sum_{v \in \mathbb{A}(K)} pl(v) + dl(v) + \psi(v)$ is:

- we underestimate the term by more than $(1/3 - \beta/3)t$ per doubly covered edge $vu \in E_{\mathbb{A}(K)}$ with $w_{vu} < (2/3 + \beta/3)t$,
- we overestimate the term by at most $t - w_{vu}$ per uncovered edge $vu \in E_{\mathbb{A}(K)}$.

Note that, since K is a cycle, ψ must assign an edge from $E_{\mathbb{A}(K)}$ to every node in $\mathbb{A}(K)$, and thus there is no special node v^* as in Case 1. Hence, if we prove inequality (5) under the aforementioned assumptions, (3) must hold after we remove the assumptions, and Lemma 13 would follow.

We now prove (5) when every covered edge $vu \in E_{\mathbb{A}(K)}$ has weight $w_{vu} = t$, and $\psi(v) = t$ for all $v \in \mathbb{A}(K)$. Again, we consider the value $pl(v) + dl(v) + \psi(v)$ as a *budget* of node v. Furthermore, we also assign budgets to edges $vu \in E_{\mathbb{A}(K)}$ that are doubly covered and have weight $w_{vu} < (2/3 + \beta/3)t$. Each of them gets a budget of $(1/3 - \beta/3)t$. Other remaining edges of $E_{\mathbb{A}(K)}$ have budget 0.

By redistributing budgets between nodes and edges, we will ensure that eventually

(i) every node in $\mathbb{A}(K)$ has a budget of at least t,

(ii) at least one node in $\mathbb{A}(K)$ has a budget strictly greater than t,

(iii) every uncovered edge $vu \in E_{\mathbb{A}(K)}$ has a budget of at least $t - w_{vu}$, and

(iv) no edge in $E_{\mathbb{A}(K)}$ has negative budget.

This would complete the proof.

We start with the leaf nodes. If $v \in \mathbb{A}(K)$ is a leaf, then (using Claim 20) it has a budget of more than $(5/3 + \beta/3)t - w_{vf(v)} + \psi(v) = (5/3 + \beta/3)t$. Using Claim 27, we can therefore add $(|\mathcal{D}(u)| - 1) \cdot (2/3 + \beta/3)t$ to the budget of every non-leaf $u \in \mathbb{A}(K)$, such that (i) is still satisfied for all leaves.

Next we consider the roots. If $v \in \mathbb{A}(K)$ is a root and $|\mathcal{D}(v)| = 1$, then (using Claim 21) it has a budget of more than $(5/3 - 2/3 \cdot \beta)t$. If $v \in \mathbb{A}(K)$ is a root and $|\mathcal{D}(v)| > 1$, then (using Claim 22 and the load added in the previous step) it has a budget of at least $t + (|\mathcal{D}(v)| - 1) \cdot (2/3 + \beta/3)t$. In the latter case, we transfer $(2/3 - 2/3 \cdot \beta)t$ to the budget of every edge in $E_{\mathbb{A}(K)}$ that is incident with v. The budget of v thereby remains at least $t + (|\mathcal{D}(v)| - 1) \cdot (2/3 + \beta/3)t - |\mathcal{D}(v)| \cdot (2/3 - 2/3 \cdot \beta)t = (1/3 - \beta/3)t + |\mathcal{D}(v)| \cdot \beta t \geq (5/3 - 2/3 \cdot \beta)t$, where the last inequality follows from $|\mathcal{D}(v)| \geq 2$ and $\beta \geq 4/7$. Using Claim 27, we can thus add $(|\mathcal{F}(u)| - 1) \cdot (2/3 - 2/3 \cdot \beta)t$ to the budget of every non-root $u \in \mathbb{A}(K)$, such that (i) is satisfied for all roots.

Before we move on to Type 1, 2, and 3 nodes, we take one step back and visit the leaves again, as their budget has increased again through the latest redistribution of load. Namely, every leaf $v \in \mathbb{A}(K)$ got an additional load of $(|\mathcal{F}(v)| - 1) \cdot (2/3 - 2/3 \cdot \beta)t$, which we now use to add $(2/3 - 2/3 \cdot \beta)t$ to the budget of every edge in $E_{\mathbb{A}(K)}$ that is incident with v, except to $vf(v)$ (which is surely covered). After this, (i) holds for every root and every leaf, and every uncovered edge $vu \in E_{\mathbb{A}(K)}$ that is incident with a root or a leaf has a budget of at least $(2/3 - 2/3 \cdot \beta)t$.

As K is a cycle, there cannot be a node of Type 1, since every $v \in \mathbb{A}(K)$ with $|\mathcal{D}(v)| > 1$ is a root.

Let us now consider the nodes of Type 2. Such a node v (using Claim 24 and the load added in previous steps) has a budget of more than $(2/3 + \beta/3)t + (|\mathcal{F}(v)| - 1) \cdot (2/3 - 2/3 \cdot \beta)t$. We transfer $(2/3 - 2/3 \cdot \beta)t$ to the budget of every edge in $E_{\mathbb{A}(K)}$ that is incident with v, except to $vf(v)$ and $vd(v)$ (which are surely covered). Since there are $|\mathcal{F}(v)| - 1$ such edges, the resulting budget at v is still more than $(2/3 + \beta/3)t$. We now reduce the budget of the edge $va(v)$ by $(1/3 - \beta/3)t$ and add this load to v's budget, which is then more than t. We will show later that this last step (reducing the budget of $va(v)$) does not cause a violation of (iv).

Finally, we consider the nodes of Type 3. Such a node v (using Claim 25 and the load added in previous steps) has a budget of more than $(5/3 - 2/3 \cdot \beta)t + (|\mathcal{F}(v)| - 1) \cdot (2/3 - 2/3 \cdot \beta)t$. We transfer $(2/3 - 2/3 \cdot \beta)t$ to the budget of every edge in $E_{\mathbb{A}(K)}$ that is incident with v, except to $vn(v)$ (which is surely covered). Since there are $|\mathcal{F}(v)|$ such edges, the resulting budget at v is still more than t.

After the above redistributions of load, (i) is satisfied. Furthermore, (ii) holds as at least one node must be of Type 2, Type 3, or a leaf, and for all these cases the load-inequality is a strict inequality. Finally, the proof of (iii) and (iv) is exactly analogous to the proof of (iv) and (v) in Case 1.

4.3 Proof of Lemma 14

In the following, let $E(V')$ denote the set of edges both of whose endpoints are in V' and $\delta(V')$ the set of edges exactly one of whose endpoints is in V', for each $V' \subseteq V$.

We prove the lemma by the following two steps.

Step 1: We create a clone of the pebble that is pushed from u^* to v^* and put this cloned pebble at v^* (by cloning, we mean the new pebble has the same weight and the same set of machines it can be assigned to) and keep the old one at u^*. We apply EXPLORE2 to this new instance and argue that the outcome is "essentially the same" as if the cloned pebble were not there. More precisely, we show

Lemma 28. *Suppose that* EXPLORE2 *is applied to the original instance (*before PUSH*) and the new instance with the cloned pebble at v^*. Then at the end of each round i, $\mathbb{A}_i = \mathbb{A}_i^\dagger$ and $\mathbb{C}_i = \mathbb{C}_i^\dagger$, where $\mathbb{A}_i$, $\mathbb{A}_i^\dagger$ are the activated sets in the original and the new instances respectively, and $\mathbb{C}_i$ and $\mathbb{C}_i^\dagger$ are the conflict trees in the original and the new instances respectively.*

Step 2: We then remove the original pebble at u^* but keep the clone at v^* (the same as the original instance after PUSH). Reapplying EXPLORE2, we then show that in each round, the set of activated nodes and the conflict trees cannot enlarge. To be precise, we show[4]

Lemma 29. *Suppose that* EXPLORE2 *is applied to the new instance with the cloned pebble put at v^* and the original instance (*after PUSH*). Then at the end of each round i,*

1. $\bigcup_{\tau=0}^{i} \mathbb{A}'_\tau \subseteq \bigcup_{\tau=0}^{i} \mathbb{A}_\tau^\dagger$;
2. $\bigcup_{\tau=0}^{i} \mathbb{C}'_\tau \subseteq \bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger$;
3. *An edge not in $E(\bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger)$, if oriented in the original instance (after* PUSH*), must have the same orientation as in the new instance.*

Here $\mathbb{A}_i^\dagger$, $\mathbb{A}'_i$ are the activated sets in the new and the original instance (after PUSH*), respectively, and $\mathbb{C}_i^\dagger$ and $\mathbb{C}'_i$ are the conflict trees in the new and the original instances (after* PUSH*), respectively.*

Lemma 28 and Lemma 29(1) together imply Lemma 14 and we will prove the two lemmas in Sections 4.3.2 and 4.3.3 respectively.

The following lemma is convenient for proving Lemmas 28 and 29 and we will prove it first. It states that the "non-determinism" in the order of fake orientations does not matter, allowing us to let the two instances "mimic" the behavior of each other when we compare the conflict trees in the main proofs.

Lemma 30. *In the sub-procedure* Constructing conflict trees*, independent of the order of the edges being directed away from the new conflict trees $\mathbb{C}_i$, the final outcome is the same in the following sense.*

1. *The sets of nodes in $\mathbb{C}_i$ is the same.*
2. *Every edge not in $E(\mathbb{C}_i)$ has the same orientation.*

4.3.1 Proof of Lemma 30

We plan to break each system into a set of subsystems and use the following lemma recursively to prove the lemma.

Lemma 31. *Let T be a tree of neutral edges in the beginning of the sub-procedure* Constructing conflict trees *whose nodes are all in $V \setminus \bigcup_{\tau=0}^{i-1} \mathbb{C}_\tau$ and consist of only the following two types:*

1. *Type α: a node v that (1) is already in $\mathbb{C}_i$ or has a directed path to a node in $\mathbb{C}_i$ in the beginning of the sub-procedure, or (2) at the end of all possible executions of the sub-procedure, it always has a directed path to some node in $\mathbb{C}_i \setminus T$.*

[4]Note that here we still refer to the instance with the cloned pebble at v^* as the *new* instance.

2. *Type β: a node v that (1) is not in $\mathbb{C}_i$ and does not have a directed path to a node in $\mathbb{C}_i$ in the beginning of the sub-procedure, and (2) at the end of all possible executions of the sub-procedure, it never has a directed path to some node in $\mathbb{C}_i$ via edges not in T. Furthermore, (3) all its incident neutral edges in the beginning of the sub-procedure are either in T, or never become directed towards v in any execution.*

Then the two properties of Lemma 30 hold. Namely, at the end of any execution, the final set $\mathbb{C}_i \cap T$ is the same and every edge in $T \backslash E(\mathbb{C}_i)$ has the same orientation.

Intuitively, Type α nodes in T are those bound to be part of $\mathbb{C}_i$ in any execution, while Type β nodes may or may not become part of $\mathbb{C}_i$. If a Type β node does become part of $\mathbb{C}_i$, then it must have a directed path to some Type α node in T via the edges in T after the execution. Notice also that by definition, a Type β node cannot be overloaded (otherwise, it is part of $\mathbb{A}_0 \subseteq \mathbb{C}_0$).

Proof. Let us first observe the outcome of an arbitrary execution of this sub-procedure. There can be two possibilities.

- **Case 1**. The entire tree T ends up being part of $\mathbb{C}_i$.
- **Case 2**. A set of sub-trees $T_1, T_2, \cdots$ become part of $\mathbb{C}_i$. The remaining nodes $T \backslash \bigcup_j T_j = \overline{F}$ form a forest. Each node $v \in \overline{F}$, if it has a non-$\overline{F}$ neighbor in T, then this neighbor is in some tree $T_j \subseteq \mathbb{C}_i$ and their shared edge is directed toward v.

The following claim is easy to verify and useful for our proof.

Claim 32. *Let $v \in T$ be a Type β node, and suppose that v has an incident edge in T that becomes outgoing during the execution of the sub-procedure. Then one of its incident edges in T must become incoming first, and furthermore $dl(v)+pl(v)+rl^i(v)+\sum_{u:vu\in T} w_{vu} > (5/3+\beta/3)t$, where $rl^i(v)$ is the rock load of v in the beginning of the sub-procedure.*

We now consider the two cases separately.

Case 1: Suppose that in a different execution, the outcome is Case 2, i.e., there remains a forest $\overline{F} \subseteq T$ not being part of $\mathbb{C}_i$.

Choose a tree $\overline{T}$ in $\overline{F}$ and then choose any node in $\overline{T}$ as the root $\overline{r}$. Define the level of a node in $\overline{T}$ as its distance to $\overline{r}$. Consider the set of nodes v with the largest level l: they must be leaves of $\overline{T}$. By Proposition 11(3), in the new execution, all non-$\overline{F}$ neighbors of v in T direct their incident edges connecting v towards v. As a result, by Claim 32 and the fact that v becomes part of $\mathbb{C}_i$ in the original execution, v of level l must direct its incident edge in $\overline{T}$ toward its neighbor of level $l-1$ in $\overline{T}$. Nodes of level $l-1$ then have incoming edges from their neighbors of level l and from their non-$\overline{F}$ neighbors in T. So again they direct the edges in $\overline{T}$ towards the nodes of level $l-2$ in $\overline{T}$. Repeating this argument, we conclude that $\overline{r}$ receives all its incident edges in T in the new execution, a contradiction to Claim 32. This proves Case 1.

Case 2: Let us divide the incident edges in T of a node $v \in \overline{F}$ into three categories according to the outcome of the original execution: incoming $E_{\mathfrak{i}}(v)$, outgoing $E_{\mathfrak{o}}(v)$, and neutral $E_{\mathfrak{n}}(v)$. Notice that by Proposition 11(3), all edges connecting v to its non-$\overline{F}$ neighbors in T are in $E_{\mathfrak{i}}(v)$. Moreover, the following facts should be clear: at the end of any other execution, (1) an edge $e \in E_{\mathfrak{o}}(v)$ must be directed away from v if all edges in $E_{\mathfrak{i}}(v)$ are directed towards v, and (2) an edge in $E_{\mathfrak{i}}(v) \cup E_{\mathfrak{n}}(v)$ can be directed away from v only if beforehand some edge in $E_{\mathfrak{o}}(v) \cup E_{\mathfrak{n}}(v)$ is directed towards v, or v ends up being part of $\mathbb{C}_i$.

Claim 33. *Let $\overline{F} \subseteq T$ be the forest not becoming part of $\mathbb{C}_i$ in the original execution. In any other execution of the sub-procedure,*

1. *given $v \in \overline{F}$, it never happens that an edge $e \in E_{\mathfrak{o}}(v) \cup E_{\mathfrak{n}}(v)$ is directed towards v or an edge in $E_{\mathfrak{i}}(v)$ is directed away from v;*

2. *none of the nodes in* $\overline{F}$ *ever becomes part of* $\mathbb{C}_i$.

Proof. Suppose that (2) is false and $v \in \overline{F}$ is the first node becoming part of $\mathbb{C}_i$. Then some edge $e = v_0u \in E_{\mathfrak{i}}(v_0)$, where v_0 and v are connected in $\overline{F}$ and $u \in T$ is a non-$\overline{F}$ neighbor of v_0, is directed towards u beforehand. So (1) must be false first. Let $e' = v'u'$ be the first edge violating (1). (At this point, no node in $\overline{F}$ is part of $\mathbb{C}_i$ yet). If $e' \in E_{\mathfrak{o}}(v') \cup E_{\mathfrak{n}}(v')$ is directed toward v', then node u' directs edge e' towards v' because it first has another edge $e'' \in E_{\mathfrak{o}}(u') \cup E_{\mathfrak{n}}(u')$ coming toward itself. Then e'' should be the edge chosen, a contradiction. If $e' \in E_{\mathfrak{i}}(v')$ is directed away from v', then some edge $e'' \in E_{\mathfrak{o}}(v') \cup E_{\mathfrak{n}}(v')$ is directed toward v' first, again implying that e'' should be chosen instead, another contradiction. Thus (1) and (2) hold. □

Claim 34. *Suppose that* $T_j \subseteq \mathbb{C}_i$ *in the original execution. Then in any other execution,*

1. $T_j \subseteq \mathbb{C}_i$*;*
2. *Every edge* $e = vu$ *with* $v \in T_j$ *and* $u \in \overline{F}$ *is directed toward* u.

Proof. For (1), we argue that T_j itself satisfies the condition of Lemma 31 and is exactly Case 1. For this, we need to show that a Type β node v of T in T_j is also a Type β node in T_j, i.e., v never has a directed path to some node in $\mathbb{C}_i$ via edges not in T_j. As v is a Type β node in T, it suffices to show that it cannot have a directed path to some Type α node in $T \backslash T_j$ via edges in T. Suppose there is such a path P. Then P must go through some node $u \in \overline{F}$, implying that u becomes part of $\mathbb{C}_i$ in this execution, a contradiction to Claim 33(2). This proves (1). (2) follows from Claim 33(2) and Proposition 11(3). □

What remains to be done is to show that all edges in $\overline{F}$ have the same orientation in any other execution. Let $L_0 \subseteq \overline{F}$ be the set of nodes v satisfying $|E_{\mathfrak{i}}(v) \cap \overline{F}| = 0$ and $L_{i>0} \subseteq \overline{F}$ be the set of nodes which can be reached from a node in L_0 by a directed path in $\overline{F}$ of maximum length exactly i after the original execution. In any other execution, by Claim 34(2), given $v \in L_0$, all edges in $E_{\mathfrak{i}}(v)$ are directed towards v, so all edges in $E_{\mathfrak{o}}(v) \cap \overline{F}$ are directed away from v. Now an inductive argument on i, combined with Claim 33(1), completes the proof of Case 2.

□

Proof. (of Lemma 30) We now explain how to make use of Lemma 31 to prove Lemma 30. For this, we decompose each system into a set of subsystems that satisfy the conditions required in Lemma 31.

First consider a system that is not a cycle. In the beginning of the sub-procedure *Constructing conflict trees*, let F be the forest consisting of the nodes in $V \setminus \bigcup_{\tau=0}^{i-1} \mathbb{C}_\tau$ and the edges that are neutral. We can assume that all nodes having a directed path to $\mathbb{C}_i$ are (already) in $\mathbb{C}_i$ as well.

Create a graph H whose node set are the connected components (trees) of F. If a non-$\mathbb{C}_i$ node in such a tree has a directed edge (we refer to the beginning of the sub-procedure) to some other non-$\mathbb{C}_i$ node in another tree, draw an arc from the node representing the former tree to the node representing the latter tree in H. (Intuitively, an arc in H indicates the possibility that a node in the former tree becomes part of $\mathbb{C}_i$ because of a directed edge to a node in $\mathbb{C}_i$ in the latter tree). As the entire system is not a cycle, some node in H must have out-degree 0. It is easy to verify that the particular tree corresponding to this node satisfies the conditions in Lemma 31, so the lemma can be applied to it.

We now find the next tree satisfying the conditions of Lemma 31 by redefining the graph H as follows. Observe that the "processed" tree (the one we applied Lemma 31 to) has exactly two types of non-$\mathbb{C}_i$ nodes in the beginning of the sub-procedure: those that always become part of $\mathbb{C}_i$ (i.e., in every possible execution of the sub-procedure) and those that never become part of $\mathbb{C}_i$. Nodes in other trees that, in the beginning, have a directed edge to the former type of

nodes are bound to become part of $\mathbb{C}_i$ (i.e., they satisfy the conditions of a Type α node in their tree). Nodes in other trees with a directed edge to the latter type of nodes are not to become part of $\mathbb{C}_i$ because of them. So in H, we can just remove the corresponding arcs and the node representing the already processed tree. In the updated H, the node with out-degree 0 is the next tree, on which Lemma 31 can be applied. Repeating this procedure, we are done with the first case (when the system is not a cycle).

Finally, consider the case that the entire system is a cycle. For the special case that the entire cycle consists of neutral edges, it is easy to verify that Lemma 30 holds. So suppose that the set of neutral edges form a forest (precisely, a set of disjoint paths). We can proceed as before—build H and find a vertex in H with out-degree 0 and recurse—except for the special case that H is a directed cycle V_1, V_2,... in the beginning. Observe that the last node $v \in V_1$ has a directed edge to the first node $u \in V_2$ and neither v nor u is in $\mathbb{C}_i$. Similarly, the last node of V_2 is also not in $\mathbb{C}_i$ and neither is the first node of V_3 and so on. In this case, it is easy to see that Lemma 30 holds for the entire system. □

4.3.2 Proof of Lemma 28

When Explore2 is applied on the original instance before Push, suppose that v^* joins the conflict trees in round k, i.e., $v^* \in \mathbb{C}_k$. We first make the following claim.

Claim 35. *Apply* Explore2 *to the new instance. In round k, immediately after the sub-procedure* Constructing conflict trees*, the following holds.*

1. $\mathbb{A}_\tau = \mathbb{A}_\tau^\dagger$*, for* $0 \le \tau \le k$*,*
2. $\mathbb{C}_\tau = \mathbb{C}_\tau^\dagger$*, for* $0 \le \tau \le k$*,*
3. *Edges not in* $E(\bigcup_{\tau=0}^{k} \mathbb{C}_\tau)$ *have the same orientations in both instances.*

We will prove the claim shortly after. In the following, we will show that $\mathbb{A}_k = \mathbb{A}_k^\dagger$ at the end of round k. Combining this with Claim 35(2)(3) and Lemma 30, an inductive argument proves that Lemma 28 is true also from round k onwards.

Recall that by the definition of Push, at the end of Explore2 in the original instance, either (1) $\mathcal{D}(v^*) = \emptyset$, or (2) $dl(v^*) + pl(v^*) + w_{v^*u} \le (5/3 - 2/3 \cdot \beta)t$ for all $u \in \mathcal{F}(v^*)$. We consider these two cases separately.

Case 1: Suppose that $\mathcal{D}(v^*) = \emptyset$ in the original instance at the end of Explore2. We will show that at the end of round k, $\mathbb{A}_k = \mathbb{A}_k^\dagger$ and in particular $v^* \in \mathbb{A}_k = \mathbb{A}_k^\dagger$. By Claim 35(1), we just have to argue that a node is activated by Rule 1 or Rule 2 in the original instance if and only if it is activated by one of these two rules in the new instance, in round k.

For v^*, recall that it is part of $\mathbb{C}_k$. It becomes so by either (1) being a Type A node in $\mathbb{A}_k$, or (2) having an outgoing edge v^*u and $u \in \bigcup_{\tau=0}^{k} \mathbb{C}_\tau$. For the former case, Claim 35(1) shows that $v^* \in \mathbb{A}_k^\dagger$. For the latter case, as $\mathcal{D}(v^*) = \emptyset$ at the end of Explore2 in the original instance, in round k, $dl(v^*) + pl(v^*) + w_{v^*u} > (5/3 + \beta/3)t$, and hence Rule 1 applies to v^*. In the new instance, the preceding inequality still holds since the pebble load of v^* is increased by the cloned pebble. As $u \in \bigcup_{\tau=0}^{k} \mathbb{C}_\tau^\dagger$ (Claim 35(2)), Rule 1 again applies to v^* (note that u is still a father of v^*, since otherwise v^* would be overloaded and part of both $\mathbb{A}_0^\dagger$ and $\mathbb{A}_0$).

For other nodes $v \neq v^*$, as $pl(v) + dl(v)$ are the same in both instances, if v is activated by Rule 1 in the original instance, then it is so too in the new instance, and vice versa. We have established that the set of nodes activated by Rule 1 is the same in both instances. Now by Claim 35(2), the set of nodes activated by Rule 2 is again the same in both instances. Therefore, $\mathbb{A}_k = \mathbb{A}_k^\dagger$ at the end of round k.

Case 2: Suppose that $dl(v^*) + pl(v^*) + w_{v^*u} \le (5/3 - 2/3 \cdot \beta)t$ for all $u \in \mathcal{F}(v^*)$ in the original instance. Then v^* cannot be a Type B node in the original instance, i.e., it is not activated by Rule 1 (but it is possible that v^* is activated by Rule 2 or as a Type A node). We now argue that in the new instance, in round k, v^* cannot be activated by Rule 1 either.

By the definition of PUSH (specifically Definition 12(3)(4)), in the original instance, each father and child $u \in \bigcup_{\tau=0}^{k} \mathbb{C}_\tau$ of v^* satisfies $dl(v^*) + pl(v^*) + w_{v^*u} \leq (5/3 - 2/3\beta)t$ (notice that when we compare original and new instance, a father can become a child and vice versa). Therefore, even with the cloned pebble (of weight at most βt) in the new instance, Rule 1 still cannot be applied to v^* in round k.

For other nodes $v \neq v^*$, it is easy to see that v is activated by Rule 1 in the original instance if and only if in the new instance in round k. We have established that the set of nodes activated by Rule 1 is the same in both instances. Now by Claim 35(2), the set of nodes activated by Rule 2 is again the same in both instances. Therefore, $\mathbb{A}_k = \mathbb{A}_k^\dagger$ at the end of round k.

Proof of Claim 35: Consider the moment at the end of round $k-1$ when EXPLORE2 is applied on the original instance before PUSH. In the special case of $k = 0$, we refer to the moment immediately after FORCED ORIENTATIONS is called in the initialization of EXPLORE2.

In this moment, let us put the cloned pebble at v^* and invoke FORCED ORIENTATIONS. This causes a (possibly empty) set of neutral edges $\overline{E}$ to become directed. Let V_0 be the set of nodes which are the heads or tails of the now directed edges in $\overline{E}$. Let V_1 be the set of nodes that can be arrived at from nodes in V_0 following the other directed edges E^* (i.e., those that are already oriented at the end of round $k-1$ before the cloned pebble is put at v^*). Observe that $v^* \in V_0$ can reach any node in $V_0 \cup V_1$ by following the directed edges in $\overline{E} \cup E^*$. Let $E_{\mathfrak{i}}(v)$, $E_{\mathfrak{o}}(v)$, and $E_{\mathfrak{n}}(v)$ denote the set of incident incoming, outgoing, neutral edges of each node $v \in V$ after we put the cloned pebble and called FORCED ORIENTATIONS. It should be clear that (1) $\overline{E} \subseteq \bigcup_{v \in V_0} E_{\mathfrak{o}}(v)$, (2) $\bigcup_{v \in V_0 \cup V_1} E_{\mathfrak{o}}(v) \cap \delta(V_0 \cup V_1) = \emptyset$, and (3) none of the nodes in V_0 is overloaded at the end of round $k-1$ (and hence also not in subsequent rounds).

Claim 36. *When* EXPLORE2 *is applied on the original instance before* PUSH*,*

1. *If an edge e is in $\overline{E} \cap E_{\mathfrak{o}}(v)$ for some $v \in V_0$, then at the end of round k, edge e is also an outgoing edge of v (independent of the order of fake orientations);*
2. *At the end of round $k-1$, none of the nodes in $V_0 \cup V_1$ is part of the conflict trees built so far, i.e. $(V_0 \cup V_1) \cap \bigcup_{\tau=0}^{k-1} \mathbb{C}_\tau = \emptyset$.*

Proof. Consider the edge $v^*u \in \overline{E} \cap E_{\mathfrak{o}}(v^*)$. As v^* is part of $\mathbb{C}_k$, at the end of round k, v^*u cannot be neutral. As it is directed toward u after the added cloned pebble,

$$w_{v^*u} + pl(v^*) + dl(v^*) + rl^{k-1}(v^*) + w > (5/3 + \beta/3)t, \tag{6}$$

where w is the weight of the cloned pebble and $rl^{k-1}(v^*)$ is the weight of the rocks assigned to v^* at the end of round $k-1$. Suppose for a contradiction that edge v^*u is directed toward v^* at the end of round k. Recall that by Definition 12(3), for the pebble to be pushed from u^* to v^* in the original instance, $pl(v^*) + dl(v^*) + rl(v^*) \leq (5/3 - 2/3 \cdot \beta)t$, where $rl(v^*)$ is the weight of the rocks assigned to v^* at the end of EXPLORE2. Then

$$dl(v^*) + pl(v^*) + (rl^{k-1}(v^*) + w_{v^*u}) + w \leq dl(v^*) + pl(v^*) + rl(v^*) + w \leq (5/3 + \beta/3)t,$$

a contradiction to inequality (6). So we establish that v^*u is directed toward u at the end of round k. Consider u and its incident edge $uu' \in \overline{E} \cap E_{\mathfrak{o}}(u)$. The fact that v^*u causes uu' to be directed toward u' implies that at the end of round k, uu' cannot be directed toward u or stay neutral. Repeating this argument, we prove (1).

If a node in $V_0 \cup V_1$ is part of $\bigcup_{\tau=0}^{k-1} \mathbb{C}_\tau$, then either v^* is part of $\bigcup_{\tau=0}^{k-1} \mathbb{C}_\tau$, a contradiction to the assumption that v^* joins the conflict trees in round k, or some node in $V_0 \backslash \{v^*\}$ has an incident edge in $\overline{E}$ directed away from it at the end of round $k-1$ (see Proposition 11(3)), a contradiction to the definition of $\overline{E}$. This proves (2).

□

Claim 36(2) has the important implication that, in the original instance, the set of nodes $V_0 \cup V_1$ is "isolated" from the rest of the graph up to the end of round $k-1$ in EXPLORE2: they do not have a directed path to nodes in $\bigcup_{\tau=0}^{k-1} \mathbb{C}_\tau$ and they are not reachable from nodes in $\bigcup_{\tau=0}^{k-2} \mathbb{A}_\tau$.

Claim 37. *Suppose that $k \geq 1$. When* EXPLORE2 *is applied on the new instance, at the end of round $k-1$,*

1. *Every edge $e \in E_{\mathfrak{o}}(v)$ (respectively $E_{\mathfrak{i}}(v)$, $E_{\mathfrak{n}}(v)$) for any $v \in V_0 \cup V_1$ is an outgoing (respectively incoming, neutral) edge of v in the new instance;*
2. *$\mathbb{A}_\tau = \mathbb{A}_\tau^\dagger$ for $0 \leq \tau \leq k-1$;*
3. *$\mathbb{C}_\tau = \mathbb{C}_\tau^\dagger$, for $0 \leq \tau \leq k-1$;*
4. *Every edge not in $E(\bigcup_{\tau=0}^{k-1} \mathbb{C}_\tau) \cup \overline{E}$ has the same orientation in both instances.*

Proof. By Claim 36(2), none of the nodes in $V_0 \cup V_1$ is overloaded in the original instance, as $k \geq 1$. By Lemma 30, we may assume that both instances decide their fake orientations based on the same fixed total order. Let us define the following events for both instances:

- β: An edge in $E(V_0 \cup V_1)$ becomes directed.
- α_1: An edge not in $E(V_0 \cup V_1)$ becomes directed.
- α_2: The sub-procedure *Activation of nodes* is executed.
- α_3: A new round starts and a set of (Type A-) nodes is activated.
- α_4: The internal while-loop of *Constructing conflict trees* is executed and a set of nodes is added into the conflict trees.

Using an inductive argument, the following fact can easily be verified:

> As long as no edge in $E_{\mathfrak{n}}(v) \cup E_{\mathfrak{i}}(v)$ becomes outgoing for any $v \in V_0 \cup V_1$, the sequences of α-events are the same in both instances (but possibly intermitted by different sequences of β-events) up to the end of round $k-1$. Furthermore, right after two corresponding α-events in the original and new instance, the sets of conflict trees and activated nodes, and the direction of all edges not in $E(V_0 \cup V_1)$ are the same in both instances.

To prove (1), consider the first moment in the new instance when an edge $e \in E_{\mathfrak{n}}(v) \cup E_{\mathfrak{i}}(v)$ becomes outgoing for any $v \in V_0 \cup V_1$ before the end of round $k-1$. For this to happen, as v is not overloaded in the original instance, there exists another edge $e' = vu \in E_{\mathfrak{n}}(v) \cup E_{\mathfrak{o}}(v)$ becoming incoming for v first. By the above fact, it must be the case that $u \in V_0 \cup V_1$. Then $e' \in E_{\mathfrak{n}}(u) \cup E_{\mathfrak{i}}(u)$, and e' becomes outgoing for u before e becomes outgoing for v, a contradiction.

We next show that every edge $e \in E_{\mathfrak{o}}(v)$ is an outgoing edge for $v \in V_0 \cup V_1$ at the end of round $k-1$ in the new instance. Assume that v^*'s system is not a cycle. Then $E_{\mathfrak{i}}(v^*) \subseteq \delta(V_0 \cup V_1)$, and all these edges are incoming at the end of round $k-1$, implying that all edges in $E_{\mathfrak{o}}(v^*)$ must be outgoing. Now an inductive argument on the rest of the nodes $v \in V_0 \cup V_1$ (based on their distance to v^*) establishes that $e \in E_{\mathfrak{o}}(v)/E_{\mathfrak{i}}(v)/E_{\mathfrak{n}}(v)$ is an outgoing/incoming/neutral edge of v at the end of round $k-1$ in the new instance. The cycle-case follows by a similar argument. This completes the proof of (1).

Finally, combining (1) with the above fact, the rest of the claim follows. □

Claim 38. *Suppose that $k = 0$. When* EXPLORE2 *is applied on the new instance, at the end of the initialization (after* FORCED ORIENTATIONS*),*

1. *Every edge $e \in E_{\mathfrak{o}}(v)$ (respectively $E_{\mathfrak{i}}(v)$, $E_{\mathfrak{n}}(v)$) for any $v \in V_0 \cup V_1$ is an outgoing (respectively incoming, neutral) edge of v in the new instance;*
2. *Every edge not in $E(V_0 \cup V_1)$ has the same orientation in both instances;*

3. The set of overloaded nodes are the same in both instances.

Proof. In the new instance, we claim that no edge in $E_{\mathfrak{n}}(v) \cup E_{\mathfrak{i}}(v)$ becomes outgoing for any $v \in V$. Suppose not and $e \in E_{\mathfrak{n}}(v) \cup E_{\mathfrak{i}}(v)$ is first such edge. If this happens because another edge $e' = vu \in E_{\mathfrak{o}}(v) \cap E_{\mathfrak{n}}(u)$ is directed toward v first, then e' should have been chosen. So v must be overloaded in the original instance and by Lemma 10, e is the only edge in $E_{\mathfrak{i}}(v)$ and $dl(v) + pl(v) + w_{e=vu_0} > (5/3 + \beta/3)t$. (Notice that $v \neq v^*$).

Consider the first time in the initialization of the original instance, $e = vu_0$ is directed toward v to cause it to become overloaded. For this to happen, either $dl(u_0) + pl(u_0) + w_{vu_0} > (5/3 + \beta/3)t$, or u_0 already has some incoming edges E_{u_0} to cause e to be directed toward v. In the former case, we know that the pair (u_0, v) precedes (v, u_0) in the total order of edges so it should still be in the new instance, a contradiction to our assumption that vu_0 is chosen to be directed toward u_0. So u_0 already has some incoming edges in the original instance. In the new instance, when vu_0 is directed toward u_0, it cannot be that all edges of E_{u_0} are directed toward u_0. So at least one such $u_1u_0 \in E_{u_0}$ remains neutral (they cannot be outgoing because of the choice of $e = vu_0$). Repeating this argument, as the system is a cycle or a tree, in the new instance, we find a path of neutral edges $vu_0u_1 \cdots$ immediately before $e = vu_0$ is directed toward u_0 and such a path ends at a node u_z where $dl(u_z) + pl(u_z) + w_{u_z u_{z-1}} > (5/3 + \beta/3)t$, and the pair (u_z, u_{z-1}) precedes the pair (v, u_0). This contradicts the assumption that $e = vu_0$ is chosen to be directed toward u_0.

So we establish that no edge in $E_{\mathfrak{n}}(v) \cup E_{\mathfrak{i}}(v)$ becomes outgoing for any $v \in V$. To complete the proof of (1) and (2), it suffices to show that after initialization of the new instance, all edges in $E_{\mathfrak{i}}(v)$ are still incoming. For $v \notin V_0 \cup V_1$, consider the directed tree T_v, where each node in T_v can follow a directed path in T_v to v in the original instance after initialization. (Note that $T_v \cap (V_0 \cup V_1) = \emptyset$). Define the levels of nodes T_v as their distances to v in T_v. Nodes v with the highest levels must be sources of the Forced Orientations, i.e., $dl(v) + pl(v) + w_{vu} > (5/3+\beta/3)t$ where u is the neighbor of v in T_v. In the new instance, such an edge cannot remain neutral after initialization. So vu is directed toward u. Now an inductive argument on the level of nodes in T_v proves that all incident edges of v are incoming, implying that all edges of $E_{\mathfrak{o}}(v)$ are outgoing. The case $v \in V_0 \cup V_1$ can be proved the same way as in the previous claim.

Finally, (3) follows from (1)(2) and the fact that no node in V_0 is overloaded in both instances. □

We now show that in round k, after the sub-procedure *Constructing conflict trees*, the outcome of the two instances are the same, to complete the proof of Claim 35. Notice that by Claim 37(1)(4) and Claim 38(1)(2), at the end of round $k-1$, the orientations of all edges not in $E(\bigcup_{\tau=0}^{k-1} \mathbb{C}_\tau)$ are the same in both instances with the only exception that $\overline{E}$ are oriented in the new instance but stay neutral in the original instance. Furthermore, by Claim 37(2) and Claim 38(3), the same set of Type A nodes are added into $\mathbb{A}_k^\dagger$, $\mathbb{A}_k$, $\mathbb{C}_k^\dagger$, $\mathbb{C}_k$ in the beginning of round k.

Let $V_1' \subseteq V_1$ be the set of nodes that can be reached by a directed path from v^* in the original instance at the end of round $k-1$ (such a path does not use edges in $\overline{E}$). Let us first suppose the system containing v^* is a tree. In the following, when we say a "sub-tree" of some edge in $E_{\mathfrak{n}}(v)$ for some $v \in V_0 \cup V_1$, we mean the sub-tree outside of $V_0 \cup V_1$ connected by such an edge. We now make use of Lemma 30 to let the two instances mirror each other's behavior. Consider how v^* first becomes part of $\mathbb{C}_k$ in the original instance.

Case 1: before the sub-procedure, v^* or some node in V_1' becomes a Type A node. Then v^* becomes part of $\mathbb{C}_k$ in both instances in the beginning of the sub-procedure. In this case, in the original instance, let v^* direct all edges in $\overline{E}$ away from v^*. Now the two instances are exactly the same.

Case 2: in the sub-procedure, due to fake orientations in the sub-trees of edges in $\cup_{v \in V_1'} E_{\mathfrak{n}}(v)$, some nodes in V_1' (hence v^*) become part of $\mathbb{C}_k$. In this case, in both instances, apply the fake orientations to these sub-trees first. Then v^* becomes part of $\mathbb{C}_k$ in both instances. Let the original instance direct edges in $\overline{E}$ away from v^*. Now the two instances are the same.

Case 3: the above two cases do not apply. Consider the previous exectuion of EXPLORE2 of the original instance in round k. $E_{\mathfrak{n}}(v^*)$ can be partitioned into $E_{\mathfrak{n}\to\mathfrak{i}}(v^*)$ and $E_{\mathfrak{n}\to\mathfrak{o}}(v^*)$, the former (latter) being those neutral edges becoming incoming (outgoing) inside the sub-procedure.

Observe that (1) $E_{\mathfrak{n}\to\mathfrak{i}}(v^*) \neq \emptyset$, otherwise v^* cannot become part of $\mathbb{C}_k$ in the original instance (see Claim 36(1)), (2) As any rock edge is heavier than the cloned pebble, if any edge $E_{\mathfrak{n}\to\mathfrak{i}}(v^*)$ is directed toward v^* in the original instance in round k, then all edges in $\overline{E}$ will be directed away from v^* (in the way described in Claim 36(1)), and (3) in round k, in the new instance, if no edge $E_{\mathfrak{n}\to\mathfrak{o}}(v^*)$ is directed toward v^*, then it cannot happen that a proper subset $E' \subset E_{\mathfrak{n}\to\mathfrak{i}}(v^*)$ directed toward v^* causes another edge in $E_{\mathfrak{n}\to\mathfrak{i}}(v^*)\backslash E'$ to be directed away from v^*. To see (3), by Definition 12.3, $\sum_{e\in E_{\mathfrak{n}\to\mathfrak{i}}(v^*)\cup E_{\mathfrak{i}}(v^*)} w_{v^*u} + w \leq (5/3 - 2/3\cdot\beta) + w < (5/3+\beta/3)t$, where w is the weight of the cloned pebble.

In the original instance, apply the fake orientations to a sub-tree of an edge $e_0 \in E_{\mathfrak{n}\to\mathfrak{i}}(v^*) \cup E_{\mathfrak{i}}(v^*)$ and let the new instance mimic. Eventually e_0 is directed toward v^* in both instances and, in the original instance, it causes all neutral edges in $\overline{E}$ to be directed away from v^*. Now the two instances have the same orientations in both instances except that some edges in $E_{\mathfrak{n}\to\mathfrak{o}}(v)$ and in their sub-trees are already oriented in the new instance while not in the original instance (this is because in the new instance, the pebble load at v^* is heavier). Then let the original instance apply the fake orientations to all sub-trees of edges $e \in (E_{\mathfrak{n}\to\mathfrak{i}}(v^*)\backslash\{e_0\}) \cup \bigcup_{v\in V_0\cup V_1\backslash\{v^*\}} E_{\mathfrak{n}}(v)$ and let the new instance mimic. In the end, v^* must be part of the conflict trees $\mathbb{C}_k$ and $\mathbb{C}_k^\dagger$ in both instances. Finally, in both instances, direct all remaining neutral edges in $E_{\mathfrak{n}\to\mathfrak{o}}(v^*)$ away from v^* and now the two instances are identical. This finishes the proof of the tree case.

Next suppose that the system containing v^* is a cycle. In the original instance, v^* joins $\mathbb{C}_k$ in two possible ways. Either v^* or some node in V_1' becomes a Type A node (then this is the same Case 1 as above), or in the beginning of the sub-procedure, there is a path v_α, v_1, v_2, …, v_β of neutral edges, where v^* is one of the v_i's and v_α and v_β are in $\mathbb{C}_k$ (note that it is possible that $v_\alpha = v_\beta$). Furthermore, the fake orientation from v_α or v_β is enough to make the entire path directed one way or another. In this case, each v_i can receive at most an edge at the end of round $k-1$ in the original instance. Therefore, when we put the cloned pebble at v^*, either v^* has a directed path to one of $\{v_\alpha, v_\beta\}$, or $V_0 \cup V_1 = \{v^*\}$. In both cases, it is easy to see that we can first let the two instances direct the remaining neutral edges along P in round k, and then the two instances are identical. This proves the cycle case and the entire proof of Claim 35.

4.3.3 Proof of Lemma 29

Our idea is to make use of Lemma 30: we will apply EXPLORE2 simultaneously to both instances and let the new instance mimic the behavior of the original instance. In the following, we implicitly assume that nodes having a directed path to $\bigcup_{\tau=0}^{i}\mathbb{C}_\tau^\dagger$ (respectively $\bigcup_{\tau=0}^{i}\mathbb{C}_\tau'$) are part of it in the new (original) instance. The lemma below explains how the mimicking is done.

Lemma 39. *In round $i \geq 0$, suppose that Lemma 29(2)(3) hold and both instances are in the sub-procedure* Constructing conflict trees*. Let the original instance apply an arbitrary fake orientation. Then the new instance can apply a number of fake orientations so that Lemma 29(2)(3) still hold.*

Proof. In the original instance, suppose that an edge $e_0 = v_0u_0$ is directed toward u_0 and FORCED ORIENTATIONS is invoked. A tree T_{u_0} of neutral edges are then further directed away from u_0. Given two incident edges e, e' of a node $v \in T_{u_0}$, we write $e \prec e'$ if e is closer to u_0 than e'. We make an important observation.

Claim 40. *Suppose that $v \in T_{u_0}$ and $v \notin \bigcup_{\tau=0}^{i}\mathbb{C}_\tau^\dagger$. Furthermore, suppose that $e, e' \in T_{u_0}$ are incident on v and $e \prec e'$. Then v can take at most one of them in the new instance, i.e., if either of them is directed toward v, then the other must be directed away from v.*

In the special case of $v = u_0 \notin \bigcup_{\tau=0}^{i}\mathbb{C}_\tau^\dagger$, assuming that e is an incident edge of u_0 in T_{u_0}, u_0 can take at most one of $e_0 = v_0u_0$ and e.

Proof. The dedicated load $dl(v)$, the pebble load $pl(v)$ are at least as heavy in the new instance as in the original. An edge not in $E(\bigcup_{\tau=0}^{i} C_\tau^\dagger)$, if oriented in the original instance, must be oriented in the same way in the new instance. So the rock load $rl(v)$ is also at least as heavy in the new instance as in the original. Thus, if in the original instance, e being directed toward v causes e' to be directed away from v, then v can take at most one of them in the original, and hence in the new instance.

The second part of the claim follows from the same reasoning.

□

Our goal is to make sure that the edges $T_{u_0} \setminus E(\bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger)$ are directed in the new instance the same way as in the original instance. We make another observation.

Claim 41. *In the new instance, assume that $e = vu \in T_{u_0}$.*

1. *If $e = vu$ is directed toward u_0, then both $u, v \in \bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger$.*
2. *If $e = vu$ is directed away from u_0 and its head is $u \notin \bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger$, then the entire sub-tree of T_{u_0} rooted at u is also directed away from u_0 and none of its nodes is in $\bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger$*

Proof. For (1), suppose that e is directed toward v and v is closer to u_0 then u. If $v \in \bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger$, then so is u and the claim holds. So assume that $v \notin \bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger$. Consider the incident edge $e' \in T_{u_0}$ of v and $e' \prec e$. By Claim 40, e must also be directed toward u_0. Repeating this argument, we find a sequence of edges directed toward u_0 and they either end up at a node in $\bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger$ (then implying that v is part of $\bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger$, a contradiction), or at u_0 and $u_0 \notin \bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger$. By Claim 40, in the new instance, edge v_0u_0 must also be directed toward v_0, again implying that v is part of $\bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger$, a contradiction.

(2) is the consequence of Claim 40 and our assumption that all nodes having a directed path to $\bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger$ are part of it. □

In the new instance, the set of neutral edges in T_{u_0} form a set of node-disjoint trees T_1, T_2, ..., where each tree T_j has the root node r_j that is closest to u_0 in T_j (r_j could be u_0 itself). Observe that no node in T_j can be part of $\bigcup_{\tau=0}^{i-1} \mathbb{C}_\tau^\dagger$ by Proposition 11(3). In case $r_j \neq u_0$, it follows from Claim 41 that $r_j \in \mathbb{C}_i^\dagger$. (So at least one node in T_j is part of $\mathbb{C}_i^\dagger$). In case $r_j = u_0$, it is possible that $r_j \notin \bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger$.

For the case that $r_j \neq u_0 \in \mathbb{C}_i^\dagger$, let the new instance direct the neutral edges in T_j incident on r_j away from u_0. If some edge $e = vu \in T_j$ (assuming that v is closer to u_0 than u) remains neutral after this, by Claim 40, v must be part of $\mathbb{C}_i^\dagger$. Then again let v direct all remaining neutral edges in T_j away from u_0 and continue this process until all edges in T_j are directed away from u_0. For the case that $r_j = u_0 \notin \bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger$, observe that edge u_0v_0 must still be neutral in the new instance: if it is directed toward v_0, then u_0 is part of $\bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger$; if it is directed toward u_0, then by the last part of Claim 40, $T_j = \emptyset$. So let the new instance direct v_0u_0 toward u_0 and invoke Forced Orientations. If there are still some edges in T_j remaining neutral, continue the fake orientation from other nodes in $T_j \cap \mathbb{C}_i^\dagger$ as above until all edges in T_j are directed away from u_0.

By the above mimicking, we guarantee that all edges in $T_{u_0} \setminus E(\bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger)$ are directed the same way in both instances. Next we argue that if a node $v \in T_{u_0}$ is added into $\mathbb{C}_i'$, then it is either already in $\bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger$, or is added into $\mathbb{C}_i^\dagger$ as well after the mimicking. For v to be added into $\mathbb{C}_i'$, it must have a directed path P in T_{u_0} to some node $\hat{v} \in \mathbb{C}_i' \cap T_{u_0}$ after v_0u_0 is oriented toward u_0 in the original instance. Note that $\hat{v}$ is also in $\bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger$ before the mimicking. Consider the first edge $e = \overline{u}\overline{v}$ in P from v that is oriented before the mimicking in the new instance and assume $\overline{u}$ is closer to v than $\overline{v}$. There are a few possibilites:

1. Suppose that $\overline{u} \in \bigcup_{\tau=0}^{i-1} \mathbb{C}_\tau^\dagger$, then by Proposition 11(3), either there is another edge e' before e in P that is oriented (a contradiction to the choice of e), or e is already the first edge in P, implying that $\overline{u} = v \in \bigcup_{\tau=0}^{i-1} \mathbb{C}_\tau^\dagger$.
2. Suppose that $\overline{u} \in \mathbb{C}_i^\dagger$. Then after the mimicking, in the new instance, there is a path from v to some node in $\mathbb{C}_i^\dagger$ (either $\overline{u}$ or some other node between v and $\overline{u}$ along P).
3. Suppose that $\overline{u} \notin \bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger$. If e is directed toward $\overline{u}$, then it is a contradiction to Claim 41(1). If it is directed toward $\overline{v}$, then $\overline{v}$ must be in $\bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger$, otherwise, it contradicts Claim 41(2) (since $\hat{v}$ is in the sub-tree rooted at $\overline{v}$). But in this case, $\overline{u} \in \bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger$, another contradiction.

So the only possibility remaining is that the entire path P from v to $\hat{v}$ is neutral in the new instance before the mimicking, and by Proposition 11(3), $\hat{v} \notin \bigcup_{\tau=0}^{i-1} \mathbb{C}_\tau^\dagger$, implying that $\hat{v} \in \mathbb{C}_i^\dagger$. Therefore, after the mimicking, in the new instance, v must have a directed path to some node in $\mathbb{C}_i^\dagger$ (either $\hat{v}$ or some node between v and $\hat{v}$ along P). So we have established Lemma 29(2) after the mimicking. Lemma 29(3) also holds since we have established earlier that edges in $T_{u_0} \backslash E(\bigcup_{\tau=0}^{i} \mathbb{C}_\tau^\dagger)$ have the same orientation in both instances after the mimicking (even before the nodes are added into the conflict trees in the new instance).

□

We use the above lemma to prove Lemma 29 for the case of $i \geq 1$.

Lemma 42. *Suppose that Lemma 29 holds at the end of round $i-1$ for $i > 0$. Then it holds still at the end of round i.*

Proof. In round i, it is easy to verify that the Lemma 29 is true in the beginning of the sub-proceure *Constructing conflict trees*. Now let the original instance apply all the fake orientations and let the new instance mimic, using Lemma 39. Next let the new instance finish off its fake orientations arbitrarily. It is easy to see that Lemma 29 holds at the end of round i.

□

We now handle the more difficult case of round 0. Unlike the later rounds, Lemma 29 does not hold in the beginning of the sub-procedure *Constructing conflict trees*: the set of overloaded nodes can be different in the two instances and the conflict trees of the new instance may not be the super set of those in the original instance.

In the following, we postpone the fake orinetations of the original instance and just let the new instance perform some fake orientations until Lemma 29(2)(3) hold.

Lemma 43. *In round 0, let $\mathbb{C}_0'$ be the conflict trees of the original instance in the beginning of the sub-procedure* Constructing conflict trees*. In the new instance, as long as an edge $e = vu \in E(\mathbb{C}_0')$ remains neutral and v is part of $\mathbb{C}_0^\dagger$, direct e toward u. Then finally, $\mathbb{C}_0' \subseteq C_0^\dagger$.*

Proof. [5] Consider a connected component (in the underlying graph G) C in $\mathbb{C}_0'$ in the original instance. It is easy to see that because every node $v \in C$ can follow a directed path to some overloaded node in C, v cannot receive all incident edges in C without becoming overloaded. Suppose the lemma does not hold. Then, in the new instance, there is a tree $\overline{T} \subseteq C$ remaining outside of $\mathbb{C}_0^\dagger$. All edges of C connecting $\overline{T}$ to the rest of the nodes in $C \backslash \overline{T}$ are directed toward $\overline{T}$. By induction, we can show that there is a node v *in* $\overline{T}$ which has only incoming edges, implying that $v \in \mathbb{C}_0^\dagger$, a contradiction.

□

Lemma 44. *In round 0, let $\mathbb{C}_0'$ be the conflict trees of the original instance in the beginning of the sub-procedure* Constructing conflict trees*. Suppose that $\mathbb{C}_0' \subseteq \mathbb{C}_0^\dagger$. In the new instance, as long as there is an edge $e = vu \notin E(\mathbb{C}_0')$ so that (1) it is directed toward u in the original*

[5] The proof here is very similar to the proof of Lemma 31, Case 1. So we only sketch the ideas.

instance in the beginning of the sub-procedure, (2) it is currently neutral in the new instance, and (3) $v \in \mathbb{C}_0^\dagger$ and $u \notin \mathbb{C}_0^\dagger$, let e be directed toward u in the new instance. Then finally, an edge $e \notin E(\mathbb{C}_0^\dagger)$, if directed in the original instance, is directed the same way in the new instance.

Proof. Let $E_{\mathfrak{i}}(v)$ and $E_{\mathfrak{o}}(v)$ denote the set of incoming and outgoing edges of a node $v \notin \mathbb{C}'_0$ in the original instance in the beginning of the sub-procedure in round 0. In the new instance, if an edge in $E_{\mathfrak{o}}(v)$ becomes incoming or neutral, then an edge in $E_{\mathfrak{i}}(v)$ must become neutral or outgoing, otherwise v becomes overloaded (this is true even if $E_{\mathfrak{i}}(v) = \emptyset$).

We now prove the lemma by contradiction. Suppose that edge $e_0 = v_0u \notin E(\mathbb{C}_0^\dagger)$ is directed toward u in the original instance while it is neutral or directed toward v_0 in the new instance after the fake orientations required in the lemma. Clearly $v_0 \notin \mathbb{C}_0^\dagger$ (hence $v_0 \notin \mathbb{C}'_0$). So either an edge $E_{\mathfrak{i}}(v_0)$ becomes outgoing or neutral, or v_0 is overloaded. The latter would imply $v_0 \in \mathbb{C}_0^\dagger$. So there is an edge $e_1 = v_1v_0 \in E_{\mathfrak{i}}(v_0)$ becoming neutral or outgoing. By the same reasoning, $v_1 \notin \mathbb{C}_0^\dagger$, otherwise, a contradiction occurs. Continuing, we can find a path $P = v_0v_1 \cdots$ so that none of v_is is in $\mathbb{C}_0^\dagger$ and in the original instance, P is directed from $v_{i>0}$ down to v_0. As each system is either a tree or a cycle, the process cannot continue forever. Either eventually a contradiction arises, or the last node in P is again the node u. In the latter case, the system is a cycle and none of its nodes is part of $\mathbb{C}'_0 \subseteq \mathbb{C}_0^\dagger$; furthermore, it is oriented clockwise in the original instance but counter-clockwise in the new instance (or the other way around). It is easy to see that this case cannot really happen.

□

By Lemmas 43 and 44, Lemma 29(2)(3) hold and then we can apply Lemma 39 to finish off all the remaining fake orientations in both instances while maintaining Lemma 29(2)(3).

The last thing to prove is that $\mathbb{A}'_0 \subseteq \mathbb{A}_0^\dagger$ after the activation rules are applied to both instances. In the original instance, if a node v is activated by Rule 1, then its father $u \in \mathbb{C}'_0$ is also part of $\mathbb{C}_0^\dagger$, implying that in the new instance, either v is overloaded or is again activated by Rule 1. If v is overloaded in the original instance, by Lemma 10, either its own pebble and dedicated load is already more than $(5/3+\beta/3)t$, or it has a child $u \in \mathbb{C}'_0$ so that $pl(v)+dl(v)+w_{vu} > (5/3+\beta/3)t$. Thus, in the new instance, v either is overloaded, or (as $u \in \mathbb{C}'_0 \subseteq \mathbb{C}_0^\dagger$) becomes a child of u and is activated by Rule 1. So we are sure Type A and Type B nodes of the original instance in $\mathbb{A}'_0$ must be part of $\mathbb{A}_0^\dagger$. Finally, as Lemma 29(2) holds, nodes of $\mathbb{A}'_0$ activated by Rule 2 must also be part of $\mathbb{A}_0^\dagger$. This completes the proof of round 0 and the entire proof of Lemma 29.

References

[1] D. Chakrabarty, S. Khanna, and S. Li. On $(1, \epsilon)$-restricted assignment makespan minimization. In *SODA*, 2015.

[2] T. Ebenlendr, M. Křćal, and J. Sgall. Graph balancing: A special case of scheduling unrelated parallel machines. In *SODA*, 2008.

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

[4] M. Gairing, T. Lücking, M. Mavronicolas, and B. Monien. Computing nash equilibria for scheduling on restricted parallel links. In *STOC*, pages 613–622, 2004.

[5] 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.

[6] 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.

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

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

[9] E. V. Shchepin and N. Vakhania. An optimal rounding gives a better approximation for scheduling unrelated machines. *Operation Research Letters*, 33(2):127–133, 2005.

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

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

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

A Improved Ratio for the 2-Valued Case

Suppose that $W \geq 2w$.

As before, we first assume that $t < 2W$, and discuss the case $t \geq 2W$ at the end of the section. We modify our previous algorithm as follows:

Definition 45. *A node v is*

- uncritical, *if $dl(v) + pl(v) \leq t + \lfloor \frac{W}{2} \rfloor - W - w$,*
- critical, *if $dl(v) + pl(v) > t + \lfloor \frac{W}{2} \rfloor - W$,*
- hypercritical, *if $dl(v) + pl(v) > t + \lfloor \frac{W}{2} \rfloor$.*

Modified 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 $t + \lfloor \frac{W}{2} \rfloor$. If at some point, PUSH is no longer possible, declare that OPT $\geq t + 1$.

The proof of Lemma 8 remains the same, and to establish Lemma 6 we just need to re-do the proof of Claim 7.

New Proof of Claim 7: By the same reasoning as before,

- none of the nodes in $\mathbb{A}(S)$ is uncritical,
- 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 $dl(v) + pl(v) > t + \lfloor \frac{W}{2} \rfloor - w$,
- if S is an isolated node $v \in \mathbb{A}$, then $dl(v) + pl(v) > t + \lfloor \frac{W}{2} \rfloor - w$.

We now re-do the 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

$$(t + \lfloor \frac{W}{2} \rfloor - W + 1) + (|\mathbb{A}(S)| - 1)(t + \lfloor \frac{W}{2} \rfloor - W - w + 1) + (|\mathbb{A}(S)| - 1)W =$$
$$|\mathbb{A}(S)|t - W + |\mathbb{A}(S)|(\lfloor \frac{W}{2} \rfloor + 1) - (|\mathbb{A}(S)| - 1)w >$$
$$|\mathbb{A}(S)|t + \frac{(|\mathbb{A}(S)| - 2)W}{2} - (|\mathbb{A}(S)| - 1)w \geq |\mathbb{A}(S)|t - w,$$

   where the first inequality holds because $\lfloor \frac{W}{2} \rfloor + 1 > \frac{W}{2}$ and the last inequality holds because $|\mathbb{A}(S)| \geq 2$ and $W \geq 2w$. If, on the other hand, $|\mathbb{A}(S)| = 1$, then the LHS of (2) is strictly more than

$$t + \lfloor \frac{W}{2} \rfloor - w \geq t = |\mathbb{A}(S)|t,$$

   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

$$|\mathbb{A}(S)|(t + \lfloor \frac{W}{2} \rfloor - W - w + 1) + |\mathbb{A}(S)|W > |\mathbb{A}(S)|t.$$

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

$$(t + \lfloor \frac{W}{2} \rfloor + 1) + (|\mathbb{A}(S)| - 1)(t + \lfloor \frac{W}{2} \rfloor - W - w + 1) + (|\mathbb{A}(S)| - 1)W =$$
$$|\mathbb{A}(S)|(t + \lfloor \frac{W}{2} \rfloor + 1) - (|\mathbb{A}(S)| - 1)w > |\mathbb{A}(S)|t,$$

where the last inequality holds because $W \geq 2w$.

Approximation Ratio: When $t \geq 2W$, we can again use the Gairing et al's algorithm [4], which either correctly reports that OPT $\geq t + 1$, or returns an assignment with makespan at most $t + W - 1$.

Suppose that t is the smallest number for which an assignment is returned (then OPT $\geq t$). Then the approximation ratio is

$$\frac{t + \lfloor \frac{W}{2} \rfloor}{\text{OPT}}, \text{if } t < 2W; \qquad \frac{t + W - 1}{\text{OPT}}, \text{if } t \geq 2W.$$

The former is bounded by $1 + \frac{\lfloor \frac{W}{2} \rfloor}{W}$, since OPT $\geq W$; the latter is bounded by $1 + \frac{W-1}{2W} \leq 1 + \frac{\lfloor \frac{W}{2} \rfloor}{W}$, since OPT $\geq t \geq 2W$. We can thus conclude:

Theorem 46. *Suppose that $W \geq 2w$. 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 $1 + \frac{\lfloor \frac{W}{2} \rfloor}{W}$ approximate solution in polynomial time.*