Combinatorial Algorithm for Restricted Max-Min Fair Allocation

Chidambaram Annamalai*, Christos Kalaitzis†, Ola Svensson‡

September 3, 2014

Abstract

We study the basic allocation problem of assigning resources to players so as to maximize fairness. This is one of the few natural problems that enjoys the intriguing status of having a better estimation algorithm than approximation algorithm. Indeed, a certain configuration-LP can be used to estimate the value of the optimal allocation to within a factor of $4 + \varepsilon$. In contrast, however, the best known approximation algorithm for the problem has an unspecified large constant guarantee.

In this paper we significantly narrow this gap by giving a 13-approximation algorithm for the problem. Our approach develops a local search technique introduced by Haxell [Hax95] for hypergraph matchings, and later used in this context by Asadpour, Feige, and Saberi [AFS12]. For our local search procedure to terminate in polynomial time, we introduce several new ideas such as *lazy updates* and *greedy players*. Besides the improved approximation guarantee, the highlight of our approach is that it is purely combinatorial and uses the configuration-LP only in the analysis.

Keywords: approximation algorithms, fair allocation, efficient local search

*School of Basic Sciences, EPFL. Email: chidambaram.annamalai@epfl.ch.

†School of Computer and Communication Sciences, EPFL. Email: christos.kalaitzis@epfl.ch. Supported by ERC Starting Grant 335288-OptApprox.

‡School of Computer and Communication Sciences, EPFL. Email: ola.svensson@epfl.ch. Supported by ERC Starting Grant 335288-OptApprox.

1 Introduction

We consider the Max-Min Fair Allocation problem, a basic combinatorial optimization problem, that captures the frequent dilemma of how to allocate resources to players in a fair manner. A problem instance is defined by a set $\mathcal{R}$ of indivisible resources, a set $\mathcal{P}$ of players, and a set of nonnegative values $\{v_{ij}\}_{i\in\mathcal{P}, j\in\mathcal{R}}$ where each player i has a value v_{ij} for a resource j. An allocation is simply a partition $\{R_i\}_{i\in\mathcal{P}}$ of the resource set and the valuation function v_i for any player i is additive, i.e., $v_i(R_i) = \sum_{j\in R_i} v_{ij}$. Perhaps the most natural fairness criterion in this setting is the max-min objective which, for a given allocation, is the largest $\tau \geqslant 0$ such that every player receives resources of value at least τ. Thus, the goal in this problem is to find an allocation $\{R_i\}_{i\in\mathcal{P}}$ that maximizes

$$\min_{i\in\mathcal{P}} \sum_{j\in R_i} v_{ij}.$$

This problem has also been given the name Santa Claus problem as interpreting the players as kids and the resources as presents leads to Santa's annual allocation problem of making the least happy kid as happy as possible.

A closely related problem is the classic scheduling problem of Scheduling Jobs on Unrelated Machines to Minimize Makespan. That problem has the same input as above and the only difference is the objective function: instead of maximizing the minimum we wish to minimize the maximum. In the scheduling context, this corresponds to minimizing the time at which all jobs (resources) have been completed by the machines (players) they were scheduled on. In a seminal paper, Lenstra, Shmoys, and Tardos [LST90] showed that the scheduling problem admits a 2-approximation algorithm by rounding a certain linear programming relaxation often referred to as the Assignment-LP. Their approximation algorithm has in fact the often stronger guarantee that the returned solution has value at most OPT $+v_{\max}$, where $v_{\max} := \max_{i\in\mathcal{P}, j\in\mathcal{R}} v_{ij}$ is the maximum value of a job (resource).

From the similarity between the two problems, it is natural to expect that the techniques developed for the scheduling problem are also applicable in this context. It is perhaps surprising, however, that the guarantees have not carried over so far. While a rounding of the Assignment-LP has been shown [BD05] to provide an allocation of value at least OPT $-v_{\max}$, this guarantee deteriorates with increasing $v_{\max}$. Since in hard instances of the problem (when $v_{\max} \approx OPT$) there can be players who are assigned only one resource in an optimal allocation, this result provides no guarantee in general. The lack of guarantee is in fact intrinsic to the Assignment-LP for Max-Min Fair Allocation as the relaxation is quite weak and has an unbounded integrality gap: the optimal value to the linear program can be a polynomial factor more than the optimal value of an integral solution.

To overcome the limitations of the Assignment-LP, Bansal and Sviridenko [BS06] proposed to use a stronger relaxation, called configuration-LP, for Max-Min Fair Allocation. Their paper contains several results on the strength of the configuration-LP, one negative and many positive. The negative result says that even the stronger configuration-LP has an integrality gap that grows as $\Omega(\sqrt{|\mathcal{P}|})$. Their positive results apply for the interesting case when $v_{ij} \in \{0, v_j\}$, called Restricted Max-Min Fair Allocation. For this case they give an $O(\log\log|\mathcal{P}|/\log\log\log|\mathcal{P}|)$-approximation algorithm, a substantial improvement over the integrality gap of the Assignment-LP. Notice that the restricted version has the following natural interpretation: each resource j has a fixed value v_j but it is interesting only for some subset of the players.

Bansal and Sviridenko further showed that the solution to a certain combinatorial problem on set systems would imply a constant integrality gap. This was later settled positively by Feige [Fei08a]

by using a proof technique that repeatedly used the Lovasz Local Lemma. At the time of Feige's result, however, it was not known if his arguments were constructive, i.e., if it led to a polynomial time algorithm for finding a solution with the same guarantee. This was later shown to be the case by Haeupler et al. [HSS11], who constructivized the various applications of the Lovasz Local Lemma in the paper by Feige [Fei08a]. This led to the first constant factor approximation algorithm for Restricted Max-Min Fair Allocation, albeit with a large and unspecified constant. This approach also requires the solution of the exponentially large configuration-LP by using the ellipsoid algorithm.

A different viewpoint and rounding approach for the problem was initiated by Asadpour, Feige, and Saberi [AFS12]. Their approach uses the perspective of hypergraphs matchings as one can naturally interpret the problem as a bipartite hypergraph matching problem with bipartitions $\mathcal{P}$ and $\mathcal{R}$. Indeed, in a solution of value τ each player i is matched to a subset R_i of resources of total value at least τ which corresponds to a hyperedge (i, R_i). Previously, Haxell [Hax95] provided sufficient conditions for bipartite hypergraphs to admit a perfect matching, generalizing the well known graph analog–Hall's theorem. Her proof is algorithmic in the sense that when the sufficient conditions hold, then a perfect matching can be found using a local search procedure that will terminate after at most exponentially many iterations. Haxell's techniques were successfully adapted by Asadpour et al. [AFS12] to the Restricted Max-Min Fair Allocation problem to obtain a beautiful proof showing that the configuration-LP has an integrality gap of at most 4. As the configuration-LP can be solved to any desired accuracy in polynomial time, this gives a polynomial time algorithm to estimate the value of an optimal allocation up to a factor of $4 + \varepsilon$, for any $\varepsilon > 0$. However, it does not provide a solution with the same guarantee.

The above results lend the Restricted Max-Min Fair Allocation problem an intriguing status that few other natural problems enjoy (see [Fei08b] for a comprehensive discussion on the difference between estimation and approximation algorithms). Another problem with a similar status is the restricted version of the aforementioned scheduling problem. The techniques in [AFS12] inspired the last author to show [Sve12] that the configuration-LP estimates the optimal value within a factor $33/17 + \varepsilon$ improving on the factor of 2 by Lenstra et al. [LST90]. Again, the rounding algorithm in [Sve12] is not known to terminate in polynomial time. We believe that this situation illustrates the need for new tools that improve our understanding of the configuration-LP especially in the context of basic allocation problems in combinatorial optimization.

Our results. Our main result improves the approximation guarantee for the Restricted Max-Min Fair Allocation problem. Note that $6 + 2\sqrt{10} \approx 12.3$.

Theorem 1.1. *For every $\varepsilon > 0$, there exists a combinatorial $(6 + 2\sqrt{10} + \varepsilon)$-approximation algorithm for the* Restricted Max-Min Fair Allocation *problem that runs in time $n^{O(1/\varepsilon^2 \log(1/\varepsilon))}$ where n is the size of the instance.*

Our algorithm has the advantage of being completely combinatorial. It does not solve the exponentially large configuration-LP. Instead, we use it only in the analysis. As our hidden constants are small, we believe that our algorithm is more attractive than solving the configuration-LP for a moderate ε. Our approach is based on the local search procedure introduced in this context by Asadpour et al. [AFS12], who in turn were inspired by the work of Haxell [Hax95]. Asadpour et al. raised the natural question if local search procedures based on *alternating trees* can be made to run in polynomial time. Prior to this work, the best running time guarantee was a quasi-polynomial time alternating tree algorithm by Polacek and the last author [PS12]. The main idea in that paper was to show that the local search can be restricted to alternating paths of length $O(\log n)$ (according

to a carefully chosen length function), where n is the number of players and resources. This restricts the search space of the local search giving the running time of $n^{O(\log n)}$. To further reduce the search space seems highly non-trivial and it is not where our improvement comes from. Rather, in contrast to the previous local search algorithms, we do not update the partial matching as soon as an alternating path is found. Instead, we wait until we are guaranteed a significant number of alternating paths, which then intuitively guarantees large progress. We refer to this concept as *lazy updates*. At the same time, we ensure that our alternating paths are short by introducing *greedy players* into our alternating tree: a player may claim more resources than she needs in an approximate solution.

To best illustrate these ideas we have chosen to first present a simpler algorithm in Section 3. The result of that section still gives an improved approximation guarantee and a polynomial time local search algorithm. However, it is not combinatorial as it relies on a preprocessing step that uses the solution of the configuration-LP. Our combinatorial algorithm is then presented in Section 4. Although we believe that it is easier to understand Section 4 after reading Section 3, the paper is written so that the interested reader can skip the details of the simpler algorithm and directly go to Section 4. The virtue of explaining the simpler algorithm first is that it allows us to postpone some of the complexities of the combinatorial algorithm until later, while still demonstrating the key ideas mentioned above.

Further related work. As mentioned before, the configuration-LP has an integrality gap of $\Omega\left(\sqrt{|\mathcal{P}|}\right)$ for the general Max-Min Fair Allocation problem. Asadpour and Saberi [AS07] almost matched this bound by giving a $O(\sqrt{|\mathcal{P}|}\log^3(|\mathcal{P}|))$-approximation algorithm; later improved by Saha and Srinivisan [SS10] to $O(\sqrt{|\mathcal{P}|\log|\mathcal{P}|}/\log\log|\mathcal{P}|)$. The current best approximation is $O(n^\varepsilon)$ due to Bateni et al. [BCG09] and Chakraborty et al. [CCK09]; for any $\varepsilon > 0$ their algorithms run in time $O(n^{1/\varepsilon})$. This leaves a large gap in the approximation guarantee for the general version of the problem as the only known hardness result says that it is NP-hard to approximate the problem to within a factor less than 2 [BD05]. The same hardness also holds for the restricted version.

2 The Configuration LP

Recall that a solution to the Max-Min Fair Allocation problem of value τ is a partition $\{R_i\}_{i\in\mathcal{P}}$ of the set of resources so that each player receives a set of value at least τ, i.e., $v_i(R_i) \geqslant \tau$ for $i \in \mathcal{P}$. Let $\mathcal{C}(i,\tau) = \{C \subseteq \mathcal{R} : v_i(C) \geqslant \tau\}$ be the set of configurations that player i can be allocated in a solution of value τ. The configuration-LP has a decision variable $x_{i,C}$ for each player $i \in \mathcal{P}$ and each $C \in \mathcal{C}(i,\tau)$. The intuition is that the variable $x_{i,C}$ takes value 1 if and only if she is assigned the bundle C. The configuration-LP is now a feasibility program with two sets of constraints: the first set says that each player should receive (at least) one configuration and the second set says that each item should be assigned to at most one player. The formal definition is given in the left box of Figure 1.

It is easy to see that if $CLP(\tau_0)$ is feasible, then so is $CLP(\tau)$ for all $\tau \leqslant \tau_0$. We say that the value of the configuration LP is τ_{OPT} if it is the largest value such that the above program is feasible. Since every feasible allocation is a feasible solution of configuration LP, τ_{OPT} is an upper bound on the value of the optimal allocation and therefore $CLP(\tau)$ constitues a valid relaxation.

We note that the LP has exponentially many variables; however, it is known that one can approximately solve it to any desired accuracy by designing a polynomial time (approximate) separation algorithm for the dual [BS06]. For our combinatorial algorithm, the dual shall play an important role in our analysis. By associating the sets of variables $\{y_i\}_{i\in\mathcal{P}}$ and $\{z_j\}_{j\in\mathcal{R}}$ to the

$$\sum_{C \in \mathcal{C}(i,\tau)} x_{i,C} \geqslant 1, \qquad \forall\, i \in \mathcal{P},$$

$$\sum_{i, C : j \in C, C \in \mathcal{C}(i,\tau)} x_{i,C} \leqslant 1, \qquad \forall\, j \in \mathcal{R},$$

$$x \geqslant 0.$$

$$\max \sum_{i \in \mathcal{P}} y_i - \sum_{j \in \mathcal{R}} z_j$$

$$y_i \leqslant \sum_{j \in C} z_j, \qquad \forall i \in \mathcal{P}, \forall C \in \mathcal{C}(i,\tau),$$

$$y, z \geqslant 0.$$

Figure 1: The configuration LP for a guessed optimal value τ on the left and its dual on the right.

constraints in the primal corresponding to players and resources respectively, and letting the primal have the objective function of minimizing the zero function, we obtain the dual of $CLP(\tau)$ shown in the right box of Figure 1.

3 Polynomial time alternating tree algorithm

To illustrate our key ideas we first describe a simpler alternating tree algorithm that works on *clustered* instances. This setting, while equivalent to the general problem up to constant factors, allows for a simpler exposition of our key ideas. Specifically, we will prove the following.

Theorem 3.1. *There is a polynomial time* 36*-approximation for* RESTRICTED MAX-MIN FAIR ALLOCATION.

We note, however, that producing such clustered instances requires solving the configuration-LP. We will show how to bypass the clustering step and avoid solving the configuration-LP in Section 4. For now though, we start by giving an intuitive description of the local search technique underlying the proof of Theorem 3.1. The italicized terms used in the intuitive description below will be defined later in Sections 3.1 and 3.2.

Alternating trees. Suppose that we have a *partial matching* M that currently does not match a player $p_0 \in \mathcal{P}$. We may select a set of resources $R \subseteq \mathcal{R}$ and form the edge (p_0, R) (which we call an *addable* edge) to match player p_0. If R is disjoint from all items in M then we have augmented our existing matching M. Otherwise, R intersects with some existing *blocking* edges in M which prevent us from including (p_0, R) into M. Suppose $p_1, \ldots, p_k$ are the players blocking the addition of (p_0, R). Now we form a new *layer* with these players and continue to look for addable edges for these players in the hope that eventually the value of resources in R blocked by the players $p_1, \ldots, p_k$ decreases so that (p_0, R) becomes *immediately addable*, i.e., there are resources in R and not appearing in M of sufficiently large value.

Thus, alternating trees are the natural hypergraph analogs of alternating paths for graph matchings. In this section we will formulate an alternating tree algorithm for the RESTRICTED MAX-MIN FAIR ALLOCATION that terminates in polynomial time.

Key ideas. We achieve this by employing two new ideas: *lazy updates* and *greedy players*. Previous alternating tree algorithms [AFS12, PS12] used immediately addable edges in a greedy way to modify the partial matching. Instead we show that performing lazy updates by waiting until immediately addable edges are available for a large number of players in the alternating tree allows

for making larger progress in every step where the partial matching is updated. Furthermore, the players in our alternating tree are greedy in the sense that they take up more resources than necessary. This allows us to argue that each layer in our alternating tree grows exponentially in size (measured by the number of blockers) so that *collapse* operations must happen often. Coupled with our lazy update strategy, this enables us to prove the polynomial running time guarantee we are aiming for.

Preliminaries. We let $\tau > 0$ be a guess on the value of the optimal solution. Throughout this section $\beta = 36$[1] will be the constant referring to the approximation guarantee of the allocation produced by the algorithm. We shall show that when $CLP(\tau)$ is feasible, our algorithm will terminate with a solution of value at least τ/β for the given instance of RESTRICTED MAX-MIN FAIR ALLOCATION. Combining this with a standard binary search then yields a β-approximation algorithm.

In the sequel we use the notation $S_{\leqslant i}$ to denote the union $S_0 \cup \cdots \cup S_i$.

3.1 Thin and fat edges, and partial matchings

We partition the resource set $\mathcal{R}$ into $\mathcal{R}_f := \{i \in \mathcal{R} : v_i \geqslant \tau/\beta\}$ and $\mathcal{R}_t := \{i \in \mathcal{R} \mid v_i < \tau/\beta\}$, fat and thin resources respectively. Note that in a β-approximate solution, a player is satisfied if she is assigned a single fat resource whereas she needs several thin resources. Keeping that in mind, we define *thin* and *fat edges*.

Definition 3.2 (Thin and fat edges)**.** For any $\delta \geqslant 1$, a δ-edge is a tuple (p, R) where $p \in \mathcal{P}$ and $R \subseteq \mathcal{R}$ such that R is either a minimal collection (by inclusion) of thin resources of value τ/δ for p or, a single fat resource that p is interested in. In the first case we call (p, R) a *thin* edge and in the second case a *fat* edge. Note that a thin δ-edge has value at most $\tau/\delta + \tau/\beta$ due to the minimality of the edge.

A collection of β-edges for distinct players is called a *partial matching* if the set of resources used by the edges in the collection are disjoint. We denote a partial matching by M. We say that M matches a player $p \in \mathcal{P}$ if there exists an edge in M that contains p. Using this terminology, an allocation is simply a partial matching that matches all the players in $\mathcal{P}$.

Our approach will be to show that as long as M does not match all players in $\mathcal{P}$ we can increase the size of M. This ensures that starting with an empty partial matching and repeating this procedure at most $|\mathcal{P}|$ times we will obtain an allocation of value at least τ/β. Thus, it suffices to develop such an augmenting algorithm. This is precisely what our alternating tree algorithm will do. As its description requires some concepts to be defined, we first formally define these concepts in Section 3.2. We then state a preprocessing step in Section 3.3 before describing the alternating tree algorithm in Section 3.4.

3.2 Addable edges, blocking edges, and layers

We now formally define the concepts referred to in the intuitive description of an alternating tree. The constant $\alpha = 5/2$ in the following definition regulates the "greediness" of the players.

Definition 3.3 (Addable, immediately addable, and blocking edges)**.** A thin α-edge (p, R) is said to be *addable* if there exists no subset $R' \subseteq R$ disjoint from edges in M of value at least τ/β. Otherwise, when there exists such an R', it is called an *immediately addable edge*.

[1] As our goal is to expose the main ideas, we have not optimized the constants in this section.

A thin β-edge $(p, R) \in M$ is said to be a *blocking edge* for another edge (p', R') if $R \cap R' \neq \emptyset$. We also say that the player p blocks (p', R') in this case since the corresponding blocking edge (p, R) can be inferred from M.

Notice how addable edges hold resources of value totalling at least $\tau/\alpha = 2\tau/5$ (significantly more than the $\tau/36$ that is required in a 36-approximate solution).

We will construct our alternating tree in *layers*, where each layer maintains a set of blocking edges B_i that prevent us from using the addable edges A_i.

Definition 3.4 (Layers)**.** For $i \geqslant 1$, layer L_i is a tuple (A_i, B_i) where A_i is a set of addable and immediately addable edges, B_i is a collection of blocking edges. The first layer L_0 is defined as the tuple $(\emptyset, \{p_0, \emptyset\})$ where p_0 is some player not matched by M. We refer to the players in the edges B_i as P_i.

We will consistently use ℓ to track the index of the last layer in our alternating tree. Thus, our alternating tree will be composed of layers $L_0, \ldots, L_\ell$ and will maintain a partial matching M throughout its execution. M will be updated via *collapse* operations. If there are many immediately addable edges in A_i then we can use this to update our partial matching M by *collapsing* layer L_{i-1}, i.e., using the immediately addable edges in A_i to remove blockers in B_{i-1}. The precise criterion for collapsing a layer controls the "laziness" of our update step.

Definition 3.5 (Collapsibility)**.** We call a layer L_i collapsible if there are at least $\mu|P_i|$ many immediately addable edges in A_{i+1}, where $\mu = 1/500$.

The definition is made so that every collapse operation performed on a layer L_i will remove at least μ fraction of the players in P_i that are blocking addable edges in A_i.

We now have the concepts necessary to state our algorithm.

3.3 Clustering step

First we describe the preprocessing phase that produces the *clustered* instances referred to earlier. The clustering step that we use is the following reduction due to Bansal and Sviridenko.

Theorem 3.6 (Clustering Step [BS06])**.** *Assuming that CLP(τ) is feasible, we can partition the set of players $\mathcal{P}$ into m clusters $N_1, \ldots, N_m$ in polynomial time such that*

1. *Each cluster N_k is associated with a distinct subset of $|N_k| - 1$ fat items from $\mathcal{R}_f$ such that they can be assigned to any subset of $|N_k| - 1$ players in N_k, and*
2. *there is a feasible solution x to CLP(τ) such that $\sum_{i \in N_k} \sum_{C \in C_t(i,\tau)} x_{iC} \geqslant 1/2$ for each cluster N_k, where $C_t(i, \tau)$ denotes the set of configurations for player i comprising only thin items.*

Note that the player that is not assigned a fat item can be chosen arbitrarily and independently for each cluster in the above theorem. Therefore, after this reduction, it suffices to allocate a thin β-edge for one player in each cluster to obtain a β-approximate solution for the original instance. Indeed, Theorem 3.6 guarantees that we can assign fat edges for the remaining players. For the rest of the section we assume that our instance has been grouped into clusters $N_1, \ldots, N_m$ by an application of Theorem 3.6. The second property of these clusters is that each cluster is fractionally assigned at least 1/2 LP-value of thin configurations. We will use this to prove the key lemma in this section, Lemma 3.8.

We now focus only on allocating one thin β-edge per cluster and forget about fat items completely. This makes the algorithm in Section 3.4 simpler than our final combinatorial algorithm, where we will also need to handle the assignment of fat items to players.

3.4 Description of the algorithm

Recall that it suffices to match exactly one player from each cluster with a thin β-edge. With this in mind, we say that a cluster N_k is matched by M if there exists some player $p \in N_k$ such that p is matched by M. The input is a partial matching M that matches at most one player from each cluster $N_1, \ldots, N_m$, and a cluster N_0 that is not matched by M.

We are now ready to describe our alternating tree algorithm.

Initialization. Select some player $p_0 \in N_0$. The goal is to extend M so as to also match N_0 in addition to the clusters already matched by M. Set $A_0 \leftarrow \emptyset$, $B_0 \leftarrow (p_0, \emptyset)$. The first layer L_0 is now defined by the tuple (A_0, B_0). Set $\ell \leftarrow 0$.

Iterative step. The iterative step, comprising two phases, is repeated until N_0 is matched by M.

Build phase. Set $A_{\ell+1} \leftarrow \emptyset$. We now define the thin α-edges called *candidate edges* that are added to $A_{\ell+1}$. Let $p, q \in N_k$ be players (not necessarily distinct) such that $q \in P_\ell$ and no player in N_k already appears in some edge in $A_{\ell+1}$. Then, the thin α-edge (p, R) is a candidate edge if the set R is disjoint from the resources appearing in $A_{\leqslant \ell+1} \cup B_{\leqslant \ell}$.

While there exists a candidate edge we add it to $A_{\ell+1}$, and *pair* player q with the addable edge (p, R). Note that the pairing is unique, i.e., a player is paired with at most one addable edge and vice-versa. When there are no more candidate edges, let $B_{\ell+1}$ be the set of blockers of $A_{\ell+1}$. Set $L_{\ell+1} \leftarrow (A_{\ell+1}, B_{\ell+1})$ and increment ℓ.

Collapse phase. If there exists a collapsible layer, then let L_t be the earliest collapsible layer. For each player in $q \in P_t$ that is paired with an immediately addable edge (p, R) from A_{t+1} swap q's blocking edge in M with (p, R'), where R' is a τ/β-minimal subset of R disjoint from the resources in B_{t+1}. This step modifies B_t (by removing some edges from it) and M (by swapping pairs of edges). Note that it still preserves the property that at most one player from each cluster is matched by M. Now discard all the layers with index greater than t and set ℓ to be t. As the collapse operation could have created immediately addable edges in L_t we repeat the collapse phase until there are no collapsible layers.

See Figure 2 for an illustration of a collapse operation.

Remark 3.7. We note that it is possible to find candidate edges in polynomial time. Suppose that we are building layer $L_{\ell+1}$. Let $q \in P_\ell$ such that no player in N_k already appears in some edge in $A_{\ell+1}$. For each $p \in N_k$, we can check in polynomial time if player p has a set R of resources of value at least τ/α disjoint from the resources already appearing in the sets $A_{\leqslant \ell+1} \cup B_{\leqslant \ell}$.

3.5 Analysis of the algorithm

We will show that the algorithm in Section 3.4 terminates in polynomial time, which then implies Theorem 3.1. Recall that α is the parameter that regulates the "greediness" of the players in the alternating tree while β is the approximation guarantee, and μ dictates when we collapse a layer.

The key lemma that we prove in this section is that in each layer L_i we have sufficiently many addable edges i.e., at least a constant fraction of the players in P_{i-1} are paired with some addable edges in A_i.

Lemma 3.8 (Many addable and immediately edges)**.** *Assuming that CLP(τ) is feasible, at the beginning of each iterative step, $|A_{i+1}| \geqslant |P_{\leqslant i}|/5$ for each $i = 0, \ldots, \ell - 1$.*

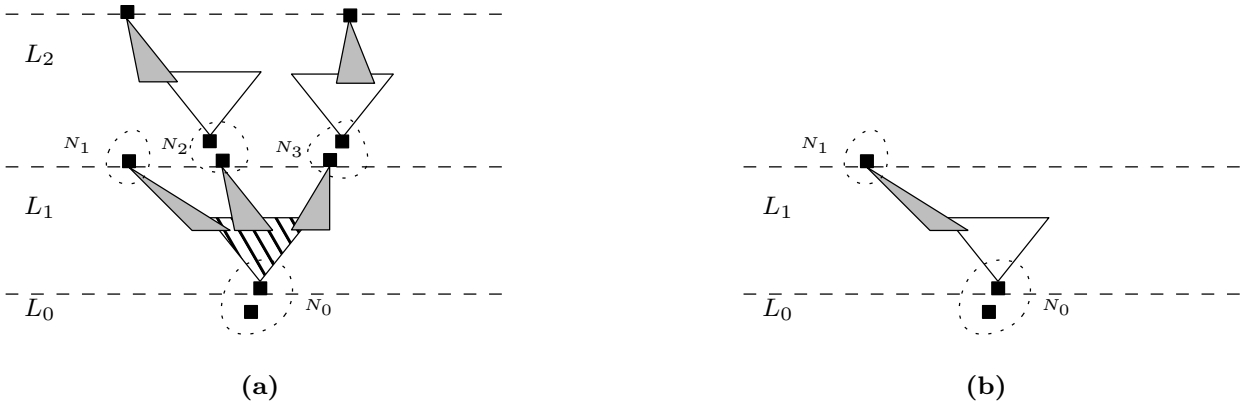

Figure 2: Example of an alternating tree (a) before and (b) after a collapse operation on layer L_1. Shaded edges are blocking edges; addable and immediately addable edges are striped and unshaded respectively; and clusters are indicated as dotted blobs around players.

We defer the proof of this statement for now and explain its consequences. As thin items are of value less than $\tau/36$, and each edge in $A_{\leqslant \ell}$ is a thin α-edge of value at least $2\tau/5$, this implies that if layer L_{i-1} is not collapsible then the number of blocking edges in L_i must be quite large. This means that the number of blocking edges will grow quickly when the layers of the alternating tree are not collapsible. We now prove this in the next lemma.

Lemma 3.9 (Exponential growth)**.** *Assuming that CLP(τ) is feasible, at the beginning of the iterative step* $|P_{i+1}| > 13|P_{\leqslant i}|/10$ *for* $i = 0, \ldots, \ell - 1$.

Proof. Fix an i such that $0 \leqslant i < \ell$. By the definition of the algorithm, L_i is not collapsible at the beginning of the iterative step. This means that there are at least $|A_{i+1}| - \mu|P_i|$ many edges in A_{i+1} which are not immediately addable. As each addable edge of A_{i+1} (except at most $\mu|P_i|$ many) has resources of value at least $\tau/\alpha - \tau/\beta$ that are blocked, we can lower bound the total value of blocked resources appearing in A_{i+1} by

$$\left(\frac{\tau}{\alpha} - \frac{\tau}{\beta}\right)(|A_{i+1}| - \mu|P_i|)\,.$$

Further, since each edge in B_{i+1} is of value at most $2\tau/\beta$ by minimality, the total value of such resources is upper bounded by $|P_{i+1}| \cdot 2\tau/\beta$. In total,

$$\left(\frac{\tau}{\alpha} - \frac{\tau}{\beta}\right)(|A_{i+1}| - \mu|P_i|) \leqslant |P_{i+1}|\frac{2\tau}{\beta} \implies |P_{i+1}| \geqslant \frac{(\beta - \alpha)(1/5 - \mu)}{2\alpha}|P_{\leqslant i}| > 13|P_{\leqslant i}|/10,$$

where we have used Lemma 3.8 to bound $|A_{i+1}|$ by $|P_{\leqslant i}|/5$ from below. □

Since the number of blocking edges grows exponentially from layer to layer, an immediate consequence of Lemma 3.9 is that the total number of layers in the alternating tree at any step in the algorithm is at most $O(\log |\mathcal{P}|)$. This means that we have to encounter a collapse operation after at most logarithmatically many iterative steps. Since our collapse operation updates a constant fraction ($\mu = 1/500$) of the players in P_i when layer L_i is collapsed, intuitively we make large progress whenever we update M during a collapse step. We prove this by maintaining a signature vector $s := (s_0, \ldots, s_\ell, \infty)$ during the execution of the algorithm, where

$$s_i := \lfloor \log_{1/(1-\mu)} |P_i| \rfloor.$$

Lemma 3.10. *The signature vector always reduces in lexicographic value across each iterative step, and the coordinates of the signature vector are always non-decreasing, i.e.,* $s_0 \leqslant s_1 \ldots \leqslant s_\ell$.

Proof. Let s and s' be the signature vectors at the beginning and at the end of some iterative step. We now consider two cases depending on whether a collapse operation occurs in this iterative step.

Case 1. **No layer was collapsed.** Clearly, $s' = (s_0, \ldots, s_\ell, s'_{\ell+1}, \infty)$ has smaller lexicographic value compared to s.

Case 2. **At least one layer was collapsed.** Let $0 \leqslant t \leqslant \ell$ be the index of the last layer that was collapsed during this iterative step. As a result of the collapse operation suppose the layer P_t changed to P'_t. Then we know that $|P'_t| < (1-\mu)|P_t|$. Since none of the layers with indices less than t were affected during this procedure, $s' = (s_0, \ldots, s_{t-1}, s'_t, \infty)$ where $s'_t = \lfloor \log_{1/(1-\mu)} |P'_t| \rfloor \leqslant \lfloor \log_{1/(1-\mu)} |P_t| \rfloor - 1 = s_t - 1$. This shows that the lexicographic value of the signature vector decreases.

In both cases, the fact that the coordinates of s' are non-decreasing follows from Lemma 3.9 and the definition of the coordinates of the signature vector. □

Choosing the "∞" coordinate of the signature vector to be some value larger than $\log_{1/(1-\mu)} |\mathcal{P}|$ (so that Lemma 3.10 still holds), we see that each coordinate of the signature vector is at most U and the number of coordinates is also at most U where $U = O(\log |\mathcal{P}|)$. Thus, the sum of the coordinates of the signature vector is always upper bounded by U^2. We now prove that the number of such signature vectors is polynomial in $|\mathcal{P}|$.

A partition of an integer N is a way of writing N as the sum of positive integers (ignoring the order of the summands). The number of partitions of an integer N can be upper bounded by $e^{O(\sqrt{N})}$ by a result of Hardy and Ramanujan [HR18][2]. Since each signature vector corresponds to some partition of an integer at most U^2, we can upper bound the total number of signature vectors by $\sum_{i \leqslant U^2} e^{O(\sqrt{i})} = |\mathcal{P}|^{O(1)}$. Since each iteration of the algorithm takes only polynomial time along with Lemma 3.10 this proves Theorem 3.1.

Before we return to the proof of the key lemma in this section, Lemma 3.8, let us note an important property of the algorithm.

Fact 3.11. *Let q be a player from some cluster N_k. Notice that if a player q is part of some blocking edge in layer L_i, i.e., $q \in P_i$, and further q has not been paired with an addable or immediately addable edge then it means that none of the players in N_k have a set of resources of value at least τ/α disjoint from the resources already appearing in the tree.*

Proof of Lemma 3.8. Notice that since the set A_i is initialized when L_i is created and not modified until L_{i-1} is collapsed, it is sufficient to verify the inequality when we construct the new layer $L_{\ell+1}$ in the build phase. The proof is now by contradiction. Suppose $|A_{\ell+1}| < |P_{\leqslant \ell}|/5$ after the build phase. Let $\mathcal{N} \subseteq \{N_1, \ldots, N_m\}$ be the clusters whose players appear in the alternating tree and are not paired with any addable or immediately addable edges. We have that, $|\mathcal{N}| = |P_{\leqslant \ell}| - |A_{\leqslant \ell+1}|$.

Recall that $C_t(i, \tau)$ denotes the set of configurations for player i comprising only thin items. By Theorem 3.6 there exists an x that is feasible for $CLP(\tau)$ such that $\sum_{i \in N_k} \sum_{C \in C_t(i,\tau)} x_{iC} = 1/2$ for each cluster N_k. Now form the bipartite hypergraph $\mathcal{H} = (\mathcal{N} \cup \mathcal{R}_t, E)$ where we have vertices for clusters and thin items in $\mathcal{R}$, and edges (N_k, C) for every cluster N_k and thin configuration C pair such that

[2]The asymptotic formula for the number of partitions of N is $\frac{1}{4N\sqrt{3}} \exp\left(\pi \sqrt{\frac{2N}{3}}\right)$ as $N \to \infty$ [HR18].

$x_{pC} > 0$ and $p \in N_k$. To each edge (N_k, C) in $\mathcal{H}$ assign the weight $\left(\sum_{i \in N_k} x_{iC}\right) \sum_{j \in C} v_j$. The total weight of edges in $\mathcal{H}$ is at least $|\mathcal{N}|\tau/2$. Let Z denote the thin items appearing in the alternating tree and let $v(Z) = \sum_{j \in Z} v_j$ denote their value. Now remove all items appearing in the alternating tree from this hypergraph to form $\mathcal{H}'$ which has edges $(N_k, C \setminus Z)$ for each edge (N_k, C) in $\mathcal{H}$. The weight of $(N_k, C \setminus Z)$ is similarly defined to be $\left(\sum_{i \in N_k} x_{iC}\right) \sum_{j \in C \setminus Z} v_j$.

Let us upper bound the total value of thin items appearing in the alternating tree, Z. Consider some layer L_j. The total value of resources in thin α-edges in A_j is at most $(\tau/\alpha + \tau/\beta)|A_j|$ by the minimality of the edges. The value of resources in B_j not already present in some edge in A_j is at most $(\tau/\beta)|B_j|$ also by minimality of the thin β-edges in B_j. Therefore, $v(Z)$ is at most

$$\sum_{j=1}^{\ell} \left((\frac{\tau}{\alpha} + \frac{\tau}{\beta})|A_j| + (\frac{\tau}{\beta})|B_j| \right) + |A_{\ell+1}| \left(\frac{\tau}{\alpha} + \frac{\tau}{\beta} \right) < |A_{\leqslant \ell+1}| \left(\frac{\tau}{\alpha} + \frac{\tau}{\beta} \right) + |P_{\leqslant \ell}| \frac{\tau}{\beta}.$$

As the sum of the edge weights in $\mathcal{H}$ is at least $(|\mathcal{N}|/2)(\tau)$, the sum of edge weights in $\mathcal{H}'$ is at least $|\mathcal{N}|\tau/2 - v(Z)$. And by Fact 3.11, the sum of edge weights in $\mathcal{H}'$ must be strictly smaller than $(\mathcal{N}/2)(\tau/\alpha)$. Thus,

$$\frac{(|P_{\leqslant \ell}| - |A_{\leqslant \ell+1}|)}{2} \tau - |A_{\leqslant \ell+1}| \left(\frac{\tau}{\alpha} + \frac{\tau}{\beta} \right) - |P_{\leqslant \ell}| \frac{\tau}{\beta} < \frac{(|P_{\leqslant \ell}| - |A_{\leqslant \ell+1}|)}{2} \frac{\tau}{\alpha}. \tag{*}$$

Note that $|A_{\leqslant \ell+1}|$ appears with a larger negative coefficient (in absolute terms) on the left-hand-side than on the right-hand-side. Therefore, if (*) holds then it also holds for an upper bound of $|A_{\leqslant \ell+1}|$. We shall compute such a bound and reach a contradiction.

We start by computing an upper bound on $|A_{j+1}|$, the number of addable edges in layer L_{j+1} for $j = 0, \ldots, \ell - 1$. Since layer L_j is not collapsible, it means that except for at most $\mu|P_j|$ edges in A_{j+1}, the remainder have at least $\tau/\alpha - \tau/\beta$ value of resources blocked by the edges in B_{j+1}. Using this,

$$\left(\frac{\tau}{\alpha} - \frac{\tau}{\beta} \right) \left(|A_{j+1}| - \mu|P_j| \right) \leqslant |P_{j+1}| \frac{2\tau}{\beta} \overset{\text{summing over } j}{\Longrightarrow} \left(\frac{\tau}{\alpha} - \frac{\tau}{\beta} \right) (|A_{\leqslant \ell}| - \mu|P_{\leqslant \ell-1}|) \leqslant |P_{\leqslant \ell}| \frac{2\tau}{\beta}.$$

Rearranging terms we have,

$$|A_{\leqslant \ell}| \leqslant |P_{\leqslant \ell}| \frac{2\alpha}{\beta - \alpha} + \mu|P_{\leqslant \ell-1}| \leqslant |P_{\leqslant \ell}| \left(\frac{2\alpha}{\beta - \alpha} + \mu \right).$$

Substituting this upper bound in (*) along with our assumption $|A_{\ell+1}| < |P_{\leqslant \ell}|/5$ we get (after some algebraic manipulations)

$$|P_{\leqslant \ell}| \left(1 - \frac{1}{\alpha} - \frac{2}{\beta} \right) - |P_{\leqslant \ell}| \left(\frac{2\alpha}{\beta - \alpha} + \mu + 1/5 \right) \left(1 + \frac{1}{\alpha} + \frac{2}{\beta} \right) < 0.$$

This is a contradiction because if we substitute in the values of α, β, and μ the left-hand-side is positive. Recall that $\alpha = 5/2, \beta = 36$, and $\mu = 1/500$. □

4 Combinatorial Algorithm

In this section we describe the combinatorial algorithm underlying our main result, Theorem 1.1. For a fixed $0 < \varepsilon \leqslant 1$, let $\beta := 2(3 + \sqrt{10}) + \varepsilon$, $\alpha := 2$, and $\mu := \varepsilon/100$. As in the description of the simple algorithm, β will be our approximation guarantee, α regulates how greedy the players are,

and μ regulates the lazy updates. Let also $\tau > 0$ be a guess on the value of the optimal solution. As before we shall show that if $CLP(\tau)$ is feasible, then our algorithm will terminate in polynomial time with a solution of value at least τ/β. Combining this with a standard binary search then yields a combinatorial β-approximation algorithm.

Before describing the algorithm in Section 4.2, we introduce in Section 4.1 several concepts, most of which are similar to the ones used in the simpler algorithm. The main novelty comes from the fact that we need to deal with alternating paths between players and fat resources. Towards this end, we introduce the new concepts (not present in the simpler algorithm) of disjoint path networks and alternating paths.

4.1 Addable and blocking edges, layers, disjoint path network and alternating paths

We partition $\mathcal{R}$ into $\mathcal{R}_f = \{i \in \mathcal{R} : v_i \geqslant \tau/\beta\}$ and $\mathcal{R}_t = \{i \in \mathcal{R} \mid v_i < \tau/\beta\}$, fat and thin resources respectively. Note that in a β-approximate solution, a player is satisfied if she is assigned a single fat resource whereas she needs several thin resources.

Addable and blocking edges and layers.

Definition 4.1 (δ-edge)**.** A δ-edge is a pair (p, R) where $p \in \mathcal{P}$ and $R \subseteq \mathcal{R}$ such that R is either a minimal collection of thin resources of value τ/δ for p or a single fat resource that p is interested in. In the first case we call (p, R) a thin edge and in the second case a fat edge.

Note that a thin δ-edge has value at most $\tau/\delta + \tau/\beta$ due to the minimality of the edge. A collection of β-edges for distinct players is called a *partial matching* if the set of resources used by the edges in the collection are disjoint. We denote a partial matching by M. Our goal will be to show that as long as M does not assign all the players in $\mathcal{P}$ we can increase the size of the partial matching M.

As in the simpler algorithm, we shall use the notion of addable edges, blocking edges and layers in our combinatorial algorithm. Blocking edges will always be thin β-edges while addable edges and immediately addable edges will always be thin α-edges.

Definition 4.2 (Blocking edge)**.** A thin β-edge $(p, R) \in M$ is said to be a blocking edge for another edge (p', R') if $R \cap R' \neq \emptyset$. We also say that the player p blocks (p', R') in this case since the corresponding blocking edge (p, R) can be inferred from M.

The definition of addable edge below reflects the "greedy" choice of a player, i.e., he selects a thin edge with more resources (of total value at least τ/α) than he needs in a β-approximate solution. Moreover, such an edge is said to be immediately addable if it contains enough resources disjoint from the current matching in order to satisfy that player.

Definition 4.3 (Addable and immediately addable edges)**.** A thin α-edge (p, R) is said to be immediately addable if there exists a subset $R' \subseteq R$ disjoint from edges in M and has value at least τ/β. Otherwise, it is called an addable edge.

In the following definition, notice that, unlike in the simpler algorithm, A_i does not contain immediately addable edges. We store them separately in a dynamically updated set I.

Definition 4.4 (Layers)**.** For $i \geqslant 1$, layer L_i is a tuple (A_i, B_i, d_i) where A_i is a set of addable edges, B_i is a collection of edges from M, and d_i is a positive integer. We refer to the players in the edges B_i as P_i. The first layer L_0 is defined as the tuple $(\emptyset, \{p_0, \emptyset\}, 0)$.

The intuition behind the definition of a layer is that it maintains a set of blocking edges B_i that prevent us from using the addable edges A_i. The number d_i associated with a layer captures the number of players in $P_{\leqslant i-1}$ that we could update at the time when layer L_i was created if all edges in $A_{\leqslant i} \cup I$ were to become immediately addable. This a number that is not needed for the description of the algorithm and it does not change after a layer is constructed. However, it will simplify the analysis.

State of the algorithm.

We consistently use ℓ to denote the index of the last layer in our algorithm. The *state of the algorithm* at any moment is completely captured by the set of layers $\{L_0, \ldots, L_\ell\}$, a set of immediately addable edges I, and the partial matching M. We denote the state by the tuple $(\ell, \{L_0, \ldots, L_\ell\}, I, M)$. We use the notation $A_{\leqslant i}$ to denote the union $A_0 \cup \cdots \cup A_i$. We similarly define $B_{\leqslant i}, P_{\leqslant i}$, etc.

Disjoint path network.

We now introduce one of the new concepts compared to the simpler algorithm. In order to deal with general alternating paths and argue about their structure we define a disjoint path flow network. In the sequel, we use the term disjoint path to denote a vertex disjoint path. For a partial matching M, let $H_M = (\mathcal{P} \cup \mathcal{R}_f, E_M)$ be the directed graph defined as follows: there is a vertex for each player in $\mathcal{P}$ and each fat resource in $\mathcal{R}_f$. There is an arc from a player in $p \in \mathcal{P}$ to a fat resource $f \in \mathcal{R}_f$ if p is interested in f unless the edge $(p, \{f\})$ appears in M in which case the direction of the arc is *reversed*. Note that the graph H_M changes only when the assignment of fat resources to players in M changes. Let $S, T \subseteq V$ be a set of sources and sinks respectively that are not necessarily disjoint. Let $\mathrm{F}_M(S, T)$ denote this flow network and let $\mathrm{DP}_M(S, T)$ denote the value of an optimal solution, i.e., the maximum number of disjoint paths from the sources S to the sinks T in the graph H_M.

In our algorithm, S and T will contain only vertices in H_M corresponding to players in $\mathcal{P}$. To specify a sink we sometimes specify an addable edge since the corresponding sink vertex can be deduced from it. For example, if we write $\mathrm{DP}_M(P_{\leqslant \ell}, I)$ then we mean the maximum number of disjoint paths that start at a player in $P_{\leqslant \ell}$ and end in a player that has an immediately addable edge in I.

Definition 4.5 (Canonical decomposition of I)**.** Given a state $(\ell, \{L_0, \ldots, L_\ell\}, I, M)$ of the algorithm, we call a collection of disjoint subsets $\{I_0, I_1, \ldots, I_\ell\}$ of I a *canonical decomposition* if

1. $|I_{\leqslant i}| = \mathrm{DP}_M(P_{\leqslant i}, I_{\leqslant i}) = \mathrm{DP}_M(P_{\leqslant i}, I)$ for $i = 0, 1, \ldots, \ell$;
2. there exists an optimal solution W to $\mathrm{F}_M(P_{\leqslant \ell}, I)$ such that, for $i = 0, 1 \ldots, \ell$, $|I_i|$ paths in W go from players $Q_i \subseteq P_i$ to the sinks in I_i. We denote these paths by W_i. We also refer to W as the canonical solution corresponding to the decomposition.

A canonical decomposition maximizes the number of alternating paths to immediately addable edges from players in layers with small indices. The intuition is that updating a player in a layer of small index constitutes a larger progress than updating a player in a layer of larger index. The concept will be crucial in our analysis to show that we can always guarantee that the algorithm makes sufficient progress. Let us first show that we can calculate a canonical decomposition in polynomial time.

Lemma 4.6. *Given a state $(\ell, \{L_0, \ldots, L_\ell\}, I, M)$ of the algorithm, we can find a canonical decomposition of I in polynomial time.*

Proof. We shall construct an optimal solution W to the flow network with sources $P_{\leqslant \ell}$ and sinks I iteratively. We begin by calculating an optimal solution when only considering the sources P_0, and then we augment it to obtain an optimal solution to the case with sources $P_{\leqslant 1}$, and so on until we have an optimal solution with sources $P_{\leqslant \ell}$. In that optimal solution, define I_i to contain those addable edges in I that were reached by players in P_i. The properties are now easy to prove. By construction, W is an optimal solution of $\mathrm{F}_M(P_{\leqslant l}, I)$. By the definition of I_i, $|I_{\leqslant i}| = \mathrm{DP}_M(P_{\leqslant i}, I_{\leqslant i})$ for each $i = 0, \ldots, \ell$. The fact that $\mathrm{DP}_M(P_{\leqslant i}, I_{\leqslant i}) = \mathrm{DP}_M(P_{\leqslant i}, I)$ for each $i = 0, 1, \ldots, \ell$ follows because $\mathrm{DP}_M(P_{\leqslant i}, I_{\leqslant i}) < \mathrm{DP}_M(P_{\leqslant i}, I)$ would contradict the fact that $|I_{\leqslant i}|$ was the value of the maximum flow in the network $\mathrm{F}_M(P_{\leqslant i}, I)$. □

The following key lemma says that if we update a layer using alternating paths then it does not interfere with future updates of the remaining layers of smaller index.

Lemma 4.7. *Consider a state $(\ell, \{L_0, \ldots, L_\ell\}, I, M)$ of the algorithm and a canonical decomposition $I_0, I_1, \ldots, I_\ell$ of I together with the canonical solution W. For $i = 0, \ldots, \ell$, let W_i be the $|I_i|$ paths that go from the players in $Q_i \subseteq P_i$ to sinks in I_i. Then, for $i = 0, 1, \ldots, \ell - 1$, there exists an optimal solution X to $\mathrm{F}_M(P_{\leqslant i}, A_{\leqslant i+1} \cup I_{\leqslant i})$ that is also an optimal solution to $\mathrm{F}_M(P_{\leqslant i}, A_{\leqslant i+1} \cup I)$ whose paths are disjoint from the paths in W_{i+1} and additionally uses all the sinks in $I_{\leqslant i}$. Moreover, such a solution can be computed in polynomial time.*

Proof. Consider a fixed i. We shall form an optimal solution X to $\mathrm{F}_M(P_{\leqslant i}, A_{\leqslant i+1} \cup I_{\leqslant i})$ that is also an optimal solution to $\mathrm{F}_M(P_{\leqslant i}, A_{\leqslant i+1} \cup I)$ and its paths are disjoint from the paths in W_{i+1} and uses all the sinks in $I_{\leqslant i}$. The initial solution will be the set of paths $W_{\leqslant i}$ from the canonical solution W which has cardinality $|I_{\leqslant i}|$. We now augment this solution using augmenting paths to the set of sinks $A_{\leqslant i+1}$. Note that throughout this execution each vertex in $I_{\leqslant i}$ will be used as a sink by some path and therefore X will use all these sinks. Further, the procedure to calculate X clearly runs in polynomial time. We shall now verify the remaining properties of X. First, suppose towards contradiction that some iteration used an augmenting path P intersecting a path in W_{i+1}. However, this would imply that there exists an augmenting path that uses a sink in I_{i+1}. We could then increase the set of disjoint paths from players in $P_{\leqslant i}$ to sinks in I to be greater than $I_{\leqslant i}$ which contradicts the property $\mathrm{DP}_M(P_{\leqslant i}, I_{\leqslant i}) = \mathrm{DP}_M(P_{\leqslant i}, I)$ of the canonical decomposition. Similarly, suppose X is not an optimal solution to $\mathrm{F}_M(P_{\leqslant i}, A_{\leqslant i+1} \cup I)$. Then there exists an augmenting path to an edge in $I \setminus I_{\leqslant i}$ which again contradicts the property $\mathrm{DP}_M(P_{\leqslant i}, I_{\leqslant i}) = \mathrm{DP}_M(P_{\leqslant i}, I)$ of the canonical decomposition. □

Alternating paths.

In the disjoint path network H_M with sources S and sinks T we call a path P with start and end vertices u and v respectively an alternating path if $u, v \in \mathcal{P}$ and $u \in S$ is a source and $v \in T$ is a sink. Our algorithm shall update the partial matching by following such alternating paths. Suppose v has an immediately addable edge, i.e., $(v, R) \in I$, then, by *alternating along the path P* in H_M we refer to the following procedure of modifying the partial matching M:

1. First remove all fat blocking edges on the path P from M and then add the remaining fat edges on P to M. That is,

$$M \leftarrow M \setminus \{(p, \{f\}) \mid (f, p) \in P\} \cup \{(p, \{f\}) \mid (p, f) \in P\}.$$

2. Then remove from M the blocking edge corresponding to the source u

3. Finally, add to M some T/β-minimal subset $R' \subseteq R$ of resources that are disjoint from the edges in M from the immediately addable edge (v, R) corresponding to the sink v.

This procedure is well defined since H_M is a bipartite graph and vertices of p alternate between the sets $\mathcal{P}$ and $\mathcal{R}_f$ of the bipartition. Moreover, as (v, R) is an immediately addable edge, the set R' is guaranteed to exist. Finally, it is easy to see that modifying M in this way using an alternating path does not change the number of fat edges in M.

4.2 Description of Algorithm

As before we present an algorithm that takes a partial matching M and augments it to one of larger size. We start with a partial matching M that assigns the maximum possible number of fat edges. Note that this can be done in polynomial time by simply solving the maximum matching problem in the graph $H_\emptyset$. The input to our algorithm is then such a partial matching M. As our algorithm only updates M by using alternating paths as described above, we will maintain the invariant that M assigns a maximum number of fat resources. Assuming that we can extend M to match one more player in polynomial time, our final algorithm therefore runs in polynomial time.

We are now ready to describe the algorithm for extending M to match an additional player.

Initialization. Select a player p_0 not matched by M. The goal is to extend M so as to also match p_0 in addition to the players already matched by M. Set $A_0 \leftarrow \emptyset$, $B_0 \leftarrow (p_0, \emptyset)$, and $d_0 \leftarrow 0$. The first layer L_0 is now defined by the tuple (A_0, B_0, d_0). Set $\ell \leftarrow 0$.

Iterative step. The iterative step consists of two phases, the first of which is always executed.

Build phase. Set $A_{\ell+1} \leftarrow \emptyset$. We now define the thin edges called *candidate edges* that are added either to the $A_{\ell+1}$ or I. A thin α-edge $c = (p, R)$ is a candidate edge if

1. the set R is disjoint from the resources appearing in $A_{\leqslant \ell+1} \cup B_{\leqslant \ell} \cup I$, and
2. $\mathrm{DP}_M(P_{\leqslant \ell}, A_{\leqslant \ell+1} \cup I + c) > \mathrm{DP}_M(P_{\leqslant \ell}, A_{\leqslant \ell+1} \cup I)$.

While there exists a candidate edge c, we add it to I if c is immediately addable or to $A_{\ell+1}$ otherwise. When this is no longer possible, let $B_{\ell+1}$ be the set of blockers of $A_{\ell+1}$, and set $d_{\ell+1} \leftarrow \mathrm{DP}_M(P_{\leqslant l}, A_{\leqslant l+1} \cup I)$. Now construct a new layer by setting $L_{\ell+1} \leftarrow (A_{\ell+1}, B_{\ell+1}, d_{\ell+1})$ and then by incrementing ℓ. We now proceed to the collapse phase.

Collapse phase. Compute the canonical decomposition $I_0 \cup \cdots \cup I_\ell$ of I. We call a layer $L_i, 0 \leqslant i \leqslant \ell$, collapsible if $|I_i| \geqslant \mu |P_i|$. If there exists a collapsible layer, then let L_t be the earliest collapsible layer and do the following:

1. Compute the optimal solution W corresponding to the canonical decomposition of I and use Lemma 4.7 to compute an optimal solution X to $\mathrm{F}_M(P_{\leqslant t-1}, A_{\leqslant t} \cup I_{\leqslant t-1})$ whose paths are disjoint from W_t. Alternate along the disjoint paths in W_t to modify the partial matching M. This step changes the sets B_t and M.
2. Set I to be $I_0 \cup \cdots \cup I_{t-1}$. For each newly formed immediately addable edges $a \in A_t$ do the following: $A_t \leftarrow A_t \setminus \{a\}$ and if X has a path that ends at a then $I \leftarrow I + a$.
3. Discard all the layers with index greater than t and set ℓ to be t. Repeat the collapse phase until there are no collapsible layers.

Repeat the iterative step until p_0 is matched by M.

Remark 4.8. A candidate edge is a thin edge whose addition increases the number of players from $P_{\leqslant \ell}$ who can simultaneously reach addable or immediately addable edges in the flow network $\mathrm{F}_M(P_{\leqslant \ell}, A_{\leqslant \ell+1} \cup I)$ through *disjoint paths*. Intuitively, this constitutes an improvement since if some of the candidate edges become immediately addable then we can alternate along those disjoint paths to update our solution. This will satisfy some players from $P_{\leqslant \ell}$ (those that reached immediately addable edges) which would then remove some blockers turning more addable edges into immediately addable edges, etc.

4.3 Analysis

From the description of the algorithm, it is clear that if it terminates, then it has increased the size of the matching by also matching player p_0. In this Section, we shall show that the algorithm terminates in polynomial time (for a fixed $\varepsilon > 0$) whenever $CLP(\tau)$ is feasible. This proves Theorem 1.1 as we can then repeat the above algorithm at most $|\mathcal{P}|$ times until all players are matched. Note also that we do not need to solve the configuration-LP as we can combine the above algorithm with a standard binary search. We first show that the algorithm satisfies some invariants. We then use these invariants to bound the running time.

4.3.1 Invariants

We show the following invariants.

Lemma 4.9. *At the beginning of each execution of the iterative step,*

1. $\mathrm{DP}_M(P_{\leqslant \ell}, I) = |I|$;
2. $\mathrm{DP}_M(P_{\leqslant i-1}, A_{\leqslant i} \cup I) \geqslant d_i$ *for each* $i = 1, \ldots, \ell$.

Proof. Both invariants trivially hold before the first execution of the iterative step. Assume that they are true before the r-th execution of the iterative step. We now verify them before the $r+1$-th iterative step. During the r-th iterative step there are two cases to consider:

No layer was collapsed. Let $L_{\ell+1}$ denote the layer that was constructed during the build phase. We start by verifying the first invariant. If no candidate edge is added to I during this phase then $|I| \geqslant \mathrm{DP}_M(P_{\leqslant \ell+1}, I) \geqslant \mathrm{DP}_M(P_{\leqslant \ell}, I) = |I|$. Suppose that $c_1, \ldots, c_k$ were the candidate edges added to the set I in that order. When candidate edge c_i was added to the set I, we have that

$$\mathrm{DP}_M(P_{\leqslant \ell}, A_{\leqslant \ell} \cup I \cup \{c_1, \ldots, c_{i-1}\} + c_i) > \mathrm{DP}_M(P_{\leqslant \ell}, A_{\leqslant \ell} \cup I \cup \{c_1, \ldots, c_{i-1}\}),$$

which implies that

$$\mathrm{DP}_M(P_{\leqslant \ell}, I \cup \{c_1, \ldots, c_{i-1}\} + c_i) > \mathrm{DP}_M(P_{\leqslant \ell}, I \cup \{c_1, \ldots, c_{i-1}\})$$

since the first inequality implies that there exists an augmenting path with c_i as the sink. Along with the induction hypothesis, these inequalities imply that

$$\mathrm{DP}_M(P_{\leqslant \ell+1}, I \cup \{c_1, \ldots, c_k\}) \geqslant \mathrm{DP}_M(P_{\leqslant \ell}, I \cup \{c_1, \ldots, c_k\}) = |I| + k = |I \cup \{c_1, \ldots, c_k\}|.$$

The inequality for $i = \ell + 1$ in the second invariant holds by the definition of $d_{\ell+1}$ during this phase. The remaining inequalities follow from the induction hypothesis since none of the previous layers $L_0, \ldots, L_\ell$ were altered during this phase and no elements from I were discarded.

At least one layer was collapsed. Let t denote the index of the last layer that was collapsed during the r-th iterative step. Let $(\ell, \{L_0, \ldots, L_{t'}\}, I, M)$ denote the state of the algorithm at the beginning of the last execution of the collapse phase during the r-th iterative step ($t' \geqslant t$). Let I' denote $I_0 \cup \ldots I_{t-1} \cup \{a_1, \ldots, a_k\}$ where $a_1, \ldots, a_k$ are the edges added in Step 2 of the collapse phase and let M' denote the partial matching after Step 1 of the collapse phase. The first invariant, $\mathrm{DP}_{M'}(P_{\leqslant t}, I') = |I'|$, now follows from Lemma 4.7. Indeed, the solution X used all the sinks in $I_0 \cup \ldots I_{t-1} \cup \{a_1, \ldots, a_k\}$ which equals I'; and these paths form a solution to $\mathrm{F}_{M'}(P_{\leqslant t}, I')$ as they are disjoint from the paths in W_t. Notice that we do not use the induction hypothesis in this case.

For the second invariant, we need to verify inequalities for $i = 1, \ldots, t$. When $i < t$, none of the sets A_i were altered during this iterative step since t was the earliest layer that was collapsed. Further, although M changes during the collapse phase, by Lemma 4.7 this change cannot reduce the number of disjoint paths from $P_{\leqslant i}$ to $A_{\leqslant i+1} \cup I$. For $i = t$, the number of disjoint paths from $P_{\leqslant t-1}$ to $A_{\leqslant t} \cup I$ cannot reduce because of Step 2 in the algorithm that maintains X as a feasible solution by the same arguments as for the first invariant. □

4.3.2 Polynomial running time

In this section, we use the invariants to show that the algorithm terminates in polynomial time assuming $CLP(\tau)$ is feasible. We start with two lemmas that show that d_i cannot be too small. The first holds in general and the second holds if $CLP(\tau)$ is feasible.

Lemma 4.10. *At the beginning of each iteration, we have that $d_i \geqslant |A_{\leqslant i}|$ for every $i = 0, \ldots, \ell$.*

Proof. We prove this by induction on the variable $r \geqslant 0$ that counts the number of times the iterative step has been executed. For $r = 0$ the statement is trivial. Suppose that it is true for $r \geqslant 0$. We shall show that it holds before the $r + 1$-th iterative step. If the iteration collapses a layer then no new layer was added and as d_i's remain unchanged and $A_{\leqslant i}$ may only decrease, the statement is true in this case. Now suppose that no layer was collapsed in this iteration and let $L_{\ell+1} = (A_{\ell+1}, B_{\ell+1}, d_{\ell+1})$ be the newly constructed layer in this phase. Suppose that $A_{\ell+1} = \{a_1, \ldots, a_k\}$ denotes the set of candidate edges added to $A_{\ell+1}$ indexed by the order in which they were added. When candidate edge a_i was added to the set $A_{\ell+1}$, we have that

$$\mathrm{DP}_M(P_{\leqslant \ell}, A_{\leqslant \ell} \cup I \cup \{a_1, \ldots, a_{i-1}\} + a_i) > \mathrm{DP}_M(P_{\leqslant \ell}, A_{\leqslant \ell} \cup I \cup \{a_1, \ldots, a_{i-1}\}).$$

Using Lemma 4.9 and the induction hypothesis,

$$\mathrm{DP}_M(P_{\leqslant \ell-1}, A_{\leqslant \ell} \cup I) \geqslant d_\ell \geqslant |A_{\leqslant \ell}|.$$

Using the previous inequalities,

$$d_{\ell+1} = \mathrm{DP}_M(P_{\leqslant \ell}, A_{\leqslant \ell+1} \cup I) \geqslant |A_{\leqslant \ell}| + k \geqslant |A_{\leqslant \ell+1}|.$$

The remaining inequalities continue to hold since, for any $0 \leqslant i \leqslant \ell + 1$, d_i does not change after layer L_i is built and the set A_i may only reduce in size in the collapse phase. Therefore, the inequalities continue to hold before the $r + 1$-th iterative step as well. □

Lemma 4.11. *Assuming $CLP(\tau)$ is feasible, at the beginning of each execution of the iterative step*

$$\mathrm{DP}_M(P_{\leqslant i-1}, A_{\leqslant i} \cup I) \geqslant d_i \geqslant \gamma |P_{\leqslant i-1}|, \text{ where } \gamma = \frac{1}{3}(\sqrt{10} - 2).$$

for every $i = 1, \ldots, \ell$.

Remark 4.12. The above condition is the only one that needs to be satisfied for the algorithm to run in polynomial time. Therefore, in a binary search, the algorithm can abort if the above condition is violated at some time; otherwise it will terminate in polynomial time.

Proof. We will prove that $d_i \geqslant \gamma|P_{\leqslant i-1}|$ for $i = 1, \ldots, \ell$ as the second invariant from Lemma 4.9 then implies the claim. Notice that d_i is defined only at the time when layer L_i is created and not altered thereafter. So it suffices to verify that: Assuming $d_i \geqslant \gamma|P_{\leqslant i-1}|$ for $i = 1, \ldots, \ell$, then for the newly constructed layer $L_{\ell+1}$, $d_{\ell+1} \geqslant \gamma|P_{\leqslant \ell}|$ also.

Suppose towards contradiction that

$$d_{\ell+1} = \mathrm{DP}_M(P_{\leqslant \ell}, A_{\leqslant \ell+1} \cup I) < \gamma|P_{\leqslant \ell}|.$$

Then, as no layer is collapsible, we have $|I_i| < \mu|P_i|$ for $i = 0, \ldots, \ell$, which by the first invariant of Lemma 4.9 implies

$$|I| = \mathrm{DP}_M(P_{\leqslant \ell}, I) < \mu|P_{\leqslant \ell}|.$$

Moreover, by Lemma 4.10 and the second invariant of Lemma 4.9 we have

$$|A_{\leqslant \ell+1}| \leqslant d_{\ell+1} = \mathrm{DP}_M(P_{\leqslant \ell}, A_{\leqslant \ell+1} \cup I) < \gamma|P_{\leqslant \ell}|.$$

Hence, we have that $|A_{\leqslant \ell+1} \cup I| < (\mu + \gamma)|P_{\leqslant \ell}|$. The rest of the proof is devoted to showing that this causes the dual of the $CLP(\tau)$ to become unbounded which leads to the required contradiction by weak duality. That is, we can then conclude that if $CLP(\tau)$ is feasible then $d_{\ell+1} \geqslant \gamma|P_{\leqslant \ell}|$.

Consider the flow network $\mathrm{F}_M(P_{\leqslant \ell}, A_{\leqslant \ell+1} \cup I \cup Z)$ with $P_{\leqslant \ell}$ as the set of sources and $A_{\leqslant \ell} \cup I \cup Z$ as the collection of sinks where,

$$Z := \{p \in \mathcal{P} \mid p \text{ has } \tau/\alpha \text{ value resources disjoint from the sets } A_{\leqslant \ell+1}, I \text{ and } B_{\leqslant \ell}\}.$$

Since there are no more candidate edges the maximum number of vertex disjoint paths from $P_{\leqslant l}$ to the sinks equals $\mathrm{DP}_M(P_{\leqslant \ell}, A_{\leqslant \ell+1} \cup I)$ which, by assumption, is less than $\gamma|P_{\leqslant \ell}|$. Therefore, by Menger's theorem there exists a set $K \subseteq V$ of vertices of cardinality less than $\gamma|P_{\leqslant \ell}|$ such that in $H_M - K$ the sources $P_{\leqslant \ell} \setminus K$ and the sinks are disconnected, i.e., no sink is reachable from any source in $P_{\leqslant \ell} \setminus K$. We now claim that we can always choose such a vertex cut so that it is a subset of the players.

Claim 4.13. There exists a vertex cut $K \subseteq \mathcal{P}$ separating $P_\ell \setminus K$ from the sinks of cardinality less than $\gamma|P_{\leqslant \ell}|$

Proof. Take any minimum cardinality vertex cut K separating $P_\ell \setminus K$ from the sinks. We already saw that $|K| < \gamma|P_{\leqslant \ell}|$. Observe that every fat resource that is reachable from $P_{\leqslant \ell} \setminus K$ must have outdegree exactly one in H_M. It cannot be more than one since M is a collection of disjoint edges, and it cannot be zero since we could then increase the number of fat edges in M which contradicts that we started with a partial matching that maximized the number of fat edges. Therefore in the vertex cut K, if there are vertices corresponding to fat resources, we can replace each fat resource with the unique player to which it has an outgoing arc to, to obtain another vertex cut also of the same cardinality that contains only vertices corresponding to players. □

Now call the induced subgraph of $H_M - K$ on the vertices that are reachable from $P_{\leqslant \ell} \setminus K$ as H'. Using H' we define the assignment of values to the dual variables in the dual of $CLP(\tau)$ as follows:

$$y_i := \begin{cases} (1-1/\alpha) & \text{if player } i \text{ is in } H', \\ 0 & \text{otherwise,} \end{cases}$$

$$z_j := \begin{cases} v_j/\tau & \text{if } j \text{ is a thin resource that appears in } A_{\leqslant \ell+1} \cup I \cup B_{\leqslant \ell}, \\ (1-1/\alpha) & \text{if } j \text{ is a fat resource in } H', \\ 0 & \text{otherwise.} \end{cases}$$

We first verify that the above assignment is feasible. Since all the dual variables are non-negative we only need to verify that $y_i \leqslant \sum_{j \in C} z_j$ for every $i \in \mathcal{P}$ and $C \in \mathcal{C}(i, \tau)$. Consider a player i that is given a positive y_i value by the above assignment. Let $C \in \mathcal{C}(i, \tau)$ be a configuration for player i of value at least τ. There are two cases we need to consider.

Case 1. **C is a thin configuration.** Suppose that $\sum_{j \in C} z_j < (1 - 1/\alpha)$. Then, by our assignment of z_j values, this implies that there exists a set $R \subseteq C$ such that R is disjoint from the resources in $A_{\leqslant \ell+1} \cup I \cup B_{\leqslant \ell}$ and $\sum_{j \in R} p_j \geqslant \tau/\alpha$. Together this contradicts the fact that H' has no sinks since i is then a sink (it is in Z).

Case 2. **C is a fat configuration.** Let j be the fat resource in C. As i was reachable in H' and K is a subset of the players, j is also present in H'. Thus, by our assignment, $z_j = 1 - 1/\alpha$.

Having proved that our assignment of y_i and z_j values constitutes a feasible solution to the dual of $CLP(\tau)$, we now compute the objective function value $\sum_i y_i - \sum_j z_j$ of the above assignment. To do so we adopt the following charging scheme: for each fat resource j in H', charge its z_j value against the unique player i such that the outgoing edge (j, i) belongs to H'. The charging scheme accounts for the z_j values of all the fat resources except for the fat resources that are leaves in H'. There are at most $|K_1|$ such fat resources, where $K_1 \subseteq K$ is the set of players to which the uncharged fat items have an outgoing arc to. Moreover, note that K_1 only consists of players that are matched in M by fat edges. Since $P_{\leqslant \ell}$ does not have any players matched by fat edges in M, no player in $K_2 := P_{\leqslant \ell} \cap K$ is present in K_1, i.e., $K_1 \cap K_2 = \emptyset$. Finally, note that no player in $P_{\leqslant \ell} \setminus K = P_{\leqslant \ell} - K_2$ has been charged. Thus, considering all players in $\mathcal{P}$ but only fat configurations, we have

$$\begin{aligned} \sum_{i \in \mathcal{P}} y_i - \sum_{j \in \mathcal{R}_f} z_j &\geqslant (1-1/\alpha)(|P_{\leqslant \ell}| - |K_2|) - (1-1/\alpha)|K_1| \\ &= (1-1/\alpha)(|P_{\leqslant \ell}| - (|K_1| + |K_2|)) \\ &> (1-1/\alpha)(1-\gamma)|P_{\leqslant \ell}|. \end{aligned}$$

We now compute the total contribution of thin resources, i.e., $\sum_{j \in \mathcal{R}_t} z_j$. The total value of thin resources from the addable edges $A_{\leqslant \ell+1}$ and immediately addable edges I is at most $(1/\alpha + 1/\beta)|A_{\leqslant \ell+1} \cup I|$. Besides the resources appearing in $A_{\leqslant \ell+1} \cup I$, the total value of resources appearing only in the blockers $B_{\leqslant \ell}$ is at most $(1/\beta)(|B_{\leqslant \ell}|) < (1/\beta)(|P_{\leqslant \ell}|)$, by the minimality of the blocking edges. Indeed, if a blocking edge has more than $1/\beta$ resources not appearing in an edge in $A_{\leqslant \ell+1} \cup I$ then those resources would form a thin β-edge which contradicts its minimality.

Thus, in total,

$$\sum_{i \in \mathcal{P}} y_i - \sum_{j \in \mathcal{R}} z_j > (1-\gamma)\left(1 - \frac{1}{\alpha}\right)|P_{\leqslant \ell}| - (\mu + \gamma)\left(\frac{1}{\alpha} + \frac{1}{\beta}\right)|P_{\leqslant \ell}| - \frac{1}{\beta}|P_{\leqslant \ell}|.$$

So, the dual of $CLP(\tau)$ is unbounded when

$$(1-\gamma)\left(1-\frac{1}{\alpha}\right)-(\mu+\gamma)\left(\frac{1}{\alpha}+\frac{1}{\beta}\right)-\frac{1}{\beta}\geqslant 0 \Leftrightarrow \gamma \leqslant \frac{\alpha\beta-(1+\mu)(\alpha+\beta)}{\alpha\beta+\alpha}.$$

Recall that $\beta = 2(3+\sqrt{10})+\varepsilon$, $\alpha = 2$, and $\mu = \varepsilon/100$. For $\varepsilon > 0$ the last inequality is equivalent to $206\sqrt{10}+3\varepsilon \leqslant 676$, which is valid for $\varepsilon \leqslant 1$.

□

We now use the previous lemma to show that if we create a new layer then the number of blocking edges (or players) in that layer will increase rapidly. This will allow us to bound the number of layers to be logarithmic and also to bound the running time.

Lemma 4.14 (Exponential growth)**.** *At each execution of the iterative step of the algorithm, we have*

$$|P_i| \geqslant \delta |P_{\leqslant i-1}|, \text{ where } \delta := \varepsilon/100,$$

for each $i = 1, \ldots, \ell$.

Proof. Suppose towards contradiction that the statement is false and let t be the smallest index that violates it, i.e., $|P_t| < \delta |P_{\leqslant t-1}|$. Since no layer $L_0, \ldots, L_t$ is collapsible at the beginning of the iterative step, $|I_i| < \mu |P_i|$ for $0 \leqslant i \leqslant t$. Hence,

$$|I_{\leqslant t}| < \mu |P_{\leqslant t}| < \mu(1+\delta)|P_{\leqslant t-1}|.$$

Further,

$$|A_{\leqslant t}| + |I_{\leqslant t}| \geqslant \mathrm{DP}_M(P_{\leqslant t-1}, A_{\leqslant t} \cup I) \geqslant \gamma |P_{\leqslant t-1}|,$$

where the first inequality is trivial while the second one follows from Lemma 4.11. This gives us

$$|A_{\leqslant t}| > (\gamma - \mu(1+\delta))\,|P_{\leqslant t-1}|.$$

We now obtain an upper bound on the total number of addable edges in the sets $A_0, \ldots, A_\ell$ by counting the value of blocked resources in each layer L_i yielding

$$|A_i|\,(\tau/\alpha - \tau/\beta) \leqslant |B_i|\,(2\tau/\beta) \overset{\text{summing over } i \text{ and rearranging}}{\Longrightarrow} |A_{\leqslant t}| \leqslant |B_{\leqslant t}| \frac{2\alpha}{\beta-\alpha}.$$

Since $|B_{\leqslant t}| < |P_{\leqslant t}|$ and $|P_{\leqslant t}| < (1+\delta)|P_{\leqslant t-1}|$ we have the bound

$$|A_{\leqslant t}| < \frac{2\alpha}{\beta-\alpha}(1+\delta)|P_{\leqslant t-1}|.$$

Therefore we will have a contradiction when

$$\frac{2\alpha}{\beta-\alpha}(1+\delta) \leqslant \gamma - (1+\delta)\mu.$$

It can be verified that for any $\varepsilon > 0$ the above inequality is equivalent to

$$22400 + 6\left(52+\sqrt{10}\right)\varepsilon + 3\varepsilon^2 \leqslant 9400\sqrt{10},$$

which is true for $\varepsilon \in [0,1]$ leading to the required contradiction. □

We are now ready to prove that our algorithm terminates in polynomial time.

Theorem 4.15 (Running time)**.** *The combinatorial algorithm runs in time polynomial in* $|\mathcal{R}|$ *and* $|\mathcal{P}|^{O(1/\varepsilon^2 \log(1/\varepsilon))}$.

Proof. To show the claim we define the signature vector $s := (s_0, \ldots, s_\ell, \infty)$, where

$$s_i := \lfloor \log_{1/(1-\mu)} \frac{|P_i|}{\delta^{i+1}} \rfloor$$

corresponding to the state $(\ell, \{L_0, \ldots, L_\ell\}, I, M)$ of the algorithm. The signature vector changes as the algorithm executes.

Claim 4.16. Across each iterative step, the lexicographic value of the signature vector decreases. Further, the coordinates of the signature vector are always non-decreasing.

Proof. We show this by induction as usual on the variable r that counts the number of times the iterative step has been executed. The statement for $r = 0$ is immediate. Suppose it is true for $r \geqslant 0$. Let $s = (s_0, \ldots, s_\ell, \infty)$ and $s' = (s'_0, \ldots, s'_{\ell'}, \infty)$ denote the signature vector at the beginning and at the end of the $(r+1)$-th iterative step. We consider two cases:

No layer was collapsed. Let $L_{\ell+1}$ be the newly constructed layer. In this case, $\ell' = \ell + 1$. By Lemma 4.14, $|P_{\ell+1}| \geqslant \delta |P_{\leqslant \ell}| > \delta |P_\ell|$. Clearly, $s' = (s_0, \ldots, s_\ell, s'_{\ell+1}, \infty)$ where $\infty > s'_{\ell+1} \geqslant s'_\ell = s_\ell$. Thus, the signature vector s' also has increasing coordinates and smaller lexicographic value compared to s.

At least one layer was collapsed. Let $0 \leqslant t \leqslant \ell$ be the index of the last layer that was collapsed during the r-th iterative step. As a result of the collapse operation suppose the layer P_t changed to P'_t. Then we know that $|P'_t| < (1-\mu)|P_t|$. Since none of the layers with indices less than t were affected during this procedure, $s' = (s_0, \ldots, s_{t-1}, s'_t, \infty)$ where $s'_t = \lfloor \log_{1/(1-\mu)} \frac{|P'_t|}{\delta^{t+1}} \rfloor \leqslant \lfloor \log_{1/(1-\mu)} \frac{(1-\mu)|P_t|}{\delta^{t+1}} \rfloor \leqslant \lfloor \log_{1/(1-\mu)} \frac{|P_t|}{\delta^{t+1}} \rfloor - 1 = s_t - 1$. This shows that the lexicographic value of the signature vector decreases. That the coordinates of s' are non-decreasing follows from Lemma 4.14. □

The proof of the theorem now follows from two facts: a) a single execution of the iterative step takes only time that is polynomial in $|\mathcal{P}|$ and $|\mathcal{R}|$; and b) the number of signature vectors is bounded by a polynomial in $|\mathcal{P}|$ for any fixed $\varepsilon > 0$. We prove the second of these claims now.

Claim 4.17. The number of signature vectors is at most $|\mathcal{P}|^{O(1/\mu \cdot 1/\delta \cdot \log(1/\delta))}$.

Proof. By Lemma 4.14, $|\mathcal{P}| \geqslant P_{\leqslant \ell} \geqslant (1+\delta) P_{\leqslant \ell-1} \geqslant \ldots \geqslant (1+\delta)^\ell |P_0|$. This implies that $\ell \leqslant \log_{1+\delta} |\mathcal{P}| \leqslant \frac{1}{\delta} \log |\mathcal{P}|$, where the last inequality is obtained by using Taylor series and that $\delta \in [0, 1/100]$.

Now consider the i-th coordinate of the signature vector s_i. It can be no larger than $\log_{1/(1-\mu)} \frac{|\mathcal{P}|}{\delta^{i+1}}$. Using the bound on the index i and after some manipulations, we get

$$\begin{aligned} s_i &\leqslant \left(\log |\mathcal{P}| + (i+1) \log \frac{1}{\delta} \right) \frac{1}{\log \frac{1}{1-\mu}} \\ &\leqslant \left(\log |\mathcal{P}| + (\frac{1}{\delta} \log |\mathcal{P}| + 1) \log \frac{1}{\delta} \right) \frac{1}{\log \frac{1}{1-\mu}} \\ &= \log |\mathcal{P}| \cdot O\left(\frac{1}{\mu \delta} \log \frac{1}{\delta} \right), \end{aligned}$$

where the final bound is obtained by again expanding using Taylor series around 0. Thus, if we let $U = \log|\mathcal{P}| \cdot O\left(\frac{1}{\mu\delta}\log\frac{1}{\delta}\right)$ be an upper bound on the number of layers and the value of each coordinate of the signature vector, then the sum of coordinates of the signature vector is always upper bounded by U^2.

Now, as in the simpler algorithm, we apply the bound on the number of partitions of an integer. Recall that the number of partitions of an integer N can be upper bounded by $e^{O(\sqrt{N})}$ [HR18]. Since each signature vector corresponds to some partition of an integer at most U^2, we can upper bound the total number of signature vectors by $\sum_{i \leqslant U^2} e^{O(\sqrt{i})}$.

Now using the bound of U, we have that the number of signatures is at most $|\mathcal{P}|^{O(1/\mu \cdot 1/\delta \cdot \log(1/\delta))}$. □

The theorem now follows from the claims by substituting in the values of $\mu := \varepsilon/100$ and $\delta := \varepsilon/100$

□

5 Conclusion

In this paper we have presented new ideas for local search algorithms based on alternating trees. This led to an improved approximation algorithm for the RESTRICTED MAX-MIN FAIR ALLOCATION problem. The obtained algorithm is also combinatorial and therefore bypasses the need of solving the exponentially large configuration-LP.

Apart from further improving the approximation guarantee, we believe that an interesting future direction is to consider our techniques in the more abstract setting of matchings in hypergraphs. For example, Haxell [Hax95] proved, using an alternating tree algorithm, a sufficient condition for a bipartite hypergraph to admit a perfect matching.

Theorem 5.1 (Haxell's condition)**.** *Consider an* $(r+1)$*-uniform bipartite hypergraph* $\mathcal{H} = (\mathcal{P} \cup \mathcal{R}, E)$ *such that for every edge* $e \in E$, $|e \cap \mathcal{P}| = 1$ *and* $|e \cap \mathcal{R}| = r$*. For* $C \subseteq \mathcal{P}$ *let* $H(E_C)$ *denote the size of the smallest set* $R \subseteq \mathcal{R}$ *that hits all the edges in* $\mathcal{H}$ *that are incident to some vertex in* C*. If for every* $C \subseteq \mathcal{P}$, $H(E_C) > (2r-1)(|C|-1)$ *then there exists a perfect matching in* $\mathcal{H}$*.*

Note that Theorem 5.1 generalizes Hall's theorem for graphs. However, the proof of the statement does not lead to a polynomial time algorithm. It is an open question if a constructive analog of Theorem 5.1 can be proved. For instance, if we strengthen the sufficient condition to $H(E_C) > 100r(|C|-1)\ \forall C \subseteq \mathcal{P}$ can we also find a perfect matching in polynomial time?

We note that, with the techniques presented in this paper, we can prove the following weaker statement: given some $0 < \varepsilon \leqslant 1$ and assuming that $H(E_C) \geqslant \Omega(1/\varepsilon)r(|C|-1)$, then there is a polynomial time algorithm that assigns one edge $e_p \in E$ for every player $p \in \mathcal{P}$ such that it is possible to choose disjoint subsets $\{S_p \subseteq e_p \cap \mathcal{R}\}_{p \in \mathcal{P}}$ of size at least $(1-\varepsilon)r$.

References

[AFS12] Arash Asadpour, Uriel Feige, and Amin Saberi. Santa claus meets hypergraph matchings. *ACM Transactions on Algorithms (TALG)*, 8(3):24, 2012. 1, 2, 4

[AS07] Arash Asadpour and Amin Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, STOC '07, pages 114–121, New York, NY, USA, 2007. ACM. 3

[BCG09] MohammadHossein Bateni, Moses Charikar, and Venkatesan Guruswami. Maxmin allocation via degree lower-bounded arborescences. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, STOC '09, pages 543–552, 2009. 3

[BD05] Ivona Bezáková and Varsha Dani. Allocating indivisible goods. *ACM SIGecom Exchanges*, 5(3):11–18, 2005. 1, 3

[BS06] Nikhil Bansal and Maxim Sviridenko. The santa claus problem. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 31–40. ACM, 2006. 1, 3, 6

[CCK09] Deeparnab Chakrabarty, Julia Chuzhoy, and Sanjeev Khanna. On allocating goods to maximize fairness. In *Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '09, pages 107–116, 2009. 3

[Fei08a] Uriel Feige. On allocations that maximize fairness. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 287–293. Society for Industrial and Applied Mathematics, 2008. 1, 2

[Fei08b] Uriel Feige. On estimation algorithms vs approximation algorithms. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 2. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2008. 2

[Hax95] Penny E Haxell. A condition for matchability in hypergraphs. *Graphs and Combinatorics*, 11(3):245–248, 1995. 1, 2, 21

[HR18] G. H. Hardy and S. Ramanujan. Asymptotic formulaæ in combinatory analysis. *Proceedings of the London Mathematical Society*, s2-17(1):75–115, 1918. 9, 21

[HSS11] Bernhard Haeupler, Barna Saha, and Aravind Srinivasan. New constructive aspects of the lovasz local lemma. *Journal of the ACM (JACM)*, 58(6):28, 2011. 2

[LST90] Jan Karel Lenstra, David B Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46(1-3):259–271, 1990. 1, 2

[PS12] Lukas Polacek and Ola Svensson. Quasi-polynomial local search for restricted max-min fair allocation. In *Automata, Languages, and Programming*, pages 726–737. Springer, 2012. 2, 4

[SS10] Barna Saha and Aravind Srinivasan. A new approximation technique for resource-allocation problems. In *ICS*, pages 342–357, 2010. 3

[Sve12] Ola Svensson. Santa claus schedules jobs on unrelated machines. *SIAM Journal on Computing*, 41(5):1318–1341, 2012. 2