Restricted Max-Min Fair Allocation∗†

Siu-Wing Cheng‡ Yuchen Mao‡

August 9, 2018

Abstract

The restricted max-min fair allocation problem seeks an allocation of resources to players that maximizes the minimum total value obtained by any player. It is NP-hard to approximate the problem to a ratio less than 2. Comparing the current best algorithm for estimating the optimal value with the current best for constructing an allocation, there is quite a gap between the ratios that can be achieved in polynomial time: roughly 4 for estimation and roughly $6+2\sqrt{10}$ for construction. We propose an algorithm that constructs an allocation with value within a factor of $6 + \delta$ from the optimum for any constant $\delta > 0$. The running time is polynomial in the input size for any constant $\delta$ chosen.

1 Introduction

Background. Let $P$ be a set of $m$ players. Let $R$ be a set of $n$ indivisible resources. Resource $r \in R$ is worth a non-negative integer value $v_{pr}$ for player $p \in P$. An allocation is a partition of $R$ into disjoint subsets $\{C_p : p \in P\}$ so that player $p$ is assigned the resources in $C_p$. The max-min fair allocation problem is to distribute resources to players so that the minimum total value of resources received by any player is maximized. We define the value of an allocation to be $\min_{p \in P} \sum_{r \in C_p} v_{pr}$. Equivalently, we want to find an allocation with maximum value.

Bezáková and Dani [5] attacked the problem using the techniques of Lenstra et al. [15] for the min-max version: the problem of scheduling on unrelated machine to minimize makespan. Bezáková and Dani proved that no polynomial-time algorithm can give an approximation ratio less than 2 unless $P = NP$. However, the assignment LP used in [15] cannot be rounded to give an approximation for the max-min allocation problem because the integrality gap is unbounded. Later, Bansal and Sviridenko [4] proposed a stronger LP relaxation, the configuration LP, for the max-min allocation problem. They showed that although the configuration LP has exponentially many constraints, it can be solved to any desired accuracy in polynomial time. They also showed that there is an integrality gap of $\Omega(\sqrt{m})$. Asadpour and Saberi [3] developed a polynomial-time rounding scheme for the configuration LP that gives an approximation ratio of $O(\sqrt{m \log^3 m})$. Saha and Srinivasan [17] improved it to $O(\sqrt{m \log m})$. Chakrabarty, Chuzhoy, and Khanna [6] showed that an $O(n^\delta \log n)$-approximate allocation can be computed in $n^{O(1/\delta)}$ time for any $\delta \geq \frac{9 \log \log n}{\log n}$.

In this paper, we focus on the restricted max-min fair allocation problem. In the restricted case, each resource is desired by some subset of players, and has the same value $v_r$ for those who desire it and value 0 for the rest. Even in this case, no approximation ratio better than 2 can be obtained unless $P = NP$ [5]. Bansal and Sviridenko [4] proposed a polynomial-time $O\left(\frac{\log \log m}{\log \log \log m}\right)$-approximation algorithm which is based on rounding the configuration LP.

∗A preliminary version appears in the Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP), 37:1-37:13.
†Research supported by the Research Grants Council, Hong Kong, China (project no. 16201116).
‡Department of Computer Science and Engineering, HKUST. \{scheng, ymaoad\}@cse.ust.hk
Feige \cite{Feige} proved that the integrality gap of the configuration LP is bounded by a constant (large and unspecified). His proof was made constructive by Haeteler et al. \cite{Haeteler}, and hence, a constant approximation can be found in polynomial time. Adapting Haxell’s techniques for hypergraph bipartite matching \cite{Haxell}, Asadpour et al. \cite{Asadpour} proved that the integrality gap of the configuration LP is at most 4. Therefore, by solving the configuration LP approximately, one can estimate the optimal solution value within a factor of $4 + \delta$ in polynomial time for any constant $\delta > 0$. However, it is not known how to construct a $(4 + \delta)$-approximate allocation in polynomial time. Inspired by the ideas in \cite{Asadpour} and \cite{Haxell}, Annamalai et al. \cite{Annamalai} developed a purely combinatorial algorithm that avoids solving the configuration LP. It runs in polynomial time and guarantees an approximation ratio $6 + 2\sqrt{10} + \delta$ for any constant $\delta > 0$. Nevertheless, the analysis still relies on the configuration LP. There is quite a gap between the current best estimation ratio\footnote{It was recently improved to $3 + \frac{\delta}{2}$ independently in \cite{Chan, Golovin}} $4 + \delta$ and the current best approximation ratio $6 + 2\sqrt{10} + \delta \approx 12.325 + \delta$. This is an interesting status that few problems have.

If one constrains the restricted case further by requiring $v_r \in \{1, \varepsilon\}$ for some fixed constant $\varepsilon \in (0, 1)$, then it becomes the $(1, \varepsilon)$-restricted case. Golovin proposed an $O(\sqrt{n})$-approximation algorithm for this case \cite{Golovin}. Chan et al. \cite{Chan} showed that it is still NP-hard to obtain an approximation ratio less than 2 and that the algorithm of Annamalai et al. \cite{Annamalai} achieves an approximation ratio of 9 in this case. The analysis in \cite{Chan} does not rely on the configuration LP.

**Our contributions.** We propose an algorithm for the restricted max-min fair allocation problem that achieves an approximation ratio of $6 + \delta$ for any constant $\delta > 0$. It runs in polynomial time for any constant $\delta$ chosen. Our algorithm uses the same framework of Annamalai et al. \cite{Annamalai}: we maintain a stack of layers to record the relation between players and resources, and use lazy update and a greedy strategy to achieve a polynomial running time.

Let $\tau^*$ be the optimal solution value. Let $\lambda > 2$ be the target approximation ratio. To obtain a $\lambda$-approximate solution, the value of resources a player need is $\tau^*/\lambda$. Our first contribution is a greedy strategy that is much more aggressive than that of Annamalai et al. \cite{Annamalai}. Their greedy strategy considers a player greedy if that player claims at least $\tau^*/2$ worth of resources, which is more than needed. In contrast, we consider a player greedy if it claims (nearly) the largest total value among all the candidates. When building the stack, as in \cite{Annamalai}, we add greedy players and the resources claimed by them to the stack. Intuitively, our more aggressive greedy strategy leads to a faster growth of the stack, and hence a significantly smaller approximation ratio can be achieved.

Our aggressive strategy brings challenge to the analysis that previous approaches \cite{Annamalai, Chan} cannot cope with. Our second contribution is a new analysis tool: an injection that maps a lot of players in the stack to their competing players who can access resources of large total value. Since players added to the stack must be greedy, they claim more than their competing players. Therefore, such an injection allows us to conclude that players in the stack claim large worth of resources. By incorporating competing players into the analysis framework of Chan et al. \cite{Chan}, we improve the approximation ratio to $6 + \delta$. Our analysis does not rely on the configuration LP, and it is purely combinatorial.

**2 Preliminaries**

Let $\tau^*$ be the optimal solution value. Let $\lambda > 2$ denote our target approximation ratio. Given any value $\tau \leq \tau^*$, our algorithm returns an allocation of value $\tau/\lambda$ in polynomial time. We will show how to combine this algorithm with binary search to obtain an allocation of value at least $\tau^*/\lambda$ in the end. We assume that $\tau$ is no more than $\tau^*$ in the rest of this section.
2.1 Fat edges, thin edges and partial allocations

A resource $r$ is fat if $v_r \geq \tau/\lambda$, and thin otherwise. For a set $B$ of thin resources, we define $\text{value}(B) = \sum_{r \in B} v_r$. For any player $p$, and any fat resource $r_f$ that is desired by $p$, $(p, r_f)$ is a fat edge. For any player $p$, and any set $B$ of thin resources, $(p, B)$ is a thin edge if $p$ desires all the resources in $B$ and $\text{value}(B) \geq \tau/\lambda$. For a thin edge $e = (p, B)$, we say player $p$ and the resources in $B$ are covered by $e$, and define $\text{value}(e) = \text{value}(B) = \sum_{r \in B} v_r$. We use uppercase calligraphic letters to denote sets of thin edges. Given a set $S$ of thin edges, we say $S$ covers a player or a thin resource if some edge in $S$ covers that player or resource, and define $\text{value}(S)$ to be the total value of the thin resources covered by $S$. That is, $\text{value}(S) = \text{value}(\bigcup_{(p, B) \in S} B)$.

Since our target approximation ratio is $\lambda$, a player will be satisfied if it receives either a single fat resource it desires, or at least $\tau/\lambda$ worth of thin resources it desires. Hence, it suffices to consider allocations that consist of two parts, one being a set of fat edges and the other being a set of thin edges.

Let $G$ be the bipartite graph formed by all the players, all the fat resources, and all the fat edges. We will start with an arbitrary maximum matching $M$ of $G$ (which is a set of fat edges) and an empty set $E$ of thin edges, and iteratively update and grow $M$ and $E$ into an allocation that satisfies all the players. We call the intermediate solutions partial allocations and formally define them as follows.

A partial allocation consists of a maximum matching $M$ of $G$ and a subset $E$ of thin edges such that (i) no two edges in $M$ and $E$ satisfy (i.e., cover) the same player, (ii) no two edges in $E$ share any resource, (iii) every edge $(p, B) \in E$ is minimal in the sense that every proper subset $B' \subset B$ has value less than $\tau/\lambda$.

In Section 3, we present an algorithm which, given a partial allocation and an unsatisfied player $p_0$, computes a new partial allocation that satisfies $p_0$ and all the players that used to be satisfied. Repeatedly invoking this algorithm returns an allocation that satisfies all the players.

2.2 A problem of finding node-disjoint paths

We define a family of networks and a problem of finding node-disjoint paths in these networks. These networks and the node-disjoint paths problem are used heavily in our algorithm and analysis.

2.2.1 The problem

Recall that $G$ is a bipartite graph formed by all the players, all the fat resources, and all the fat edges. With respect to any maximum matching $M$ of $G$, we define $G_M$ to be a directed bipartite graph obtained from $G$ by orienting edges of $G$ from $r$ to $p$ if the edge $(p, r)$ is in $M$, and from $p$ to $r$ if $(p, r)$ is not in $M$. See Figure 1(a) and (b) for an example.

Figure 1: A reduction from the node-disjoint path problem to max flow problem. The squares represent players and the circles represent fat resources.
We use $P_M$ and $\overline{P}_M$ to denote the subsets of players matched and unmatched in $M$, respectively. Given $S \subseteq \overline{P}_M$ and $T \subseteq P$, we use $G_M(S, T)$ to denote the problem of finding the maximum number of node-disjoint paths from $S$ to $T$ in $G_M$. This problem will arise in this paper for different choices of $S$ and $T$. A feasible solution of $G_M(S, T)$ is just any set of node-disjoint paths from $S$ to $T$ in $G_M$. An optimal solution maximizes the number of such paths. Let $f_M(S, T)$ denote the size of an optimal solution of $G_M(S, T)$. In the cases that $S \cap T \neq \emptyset$, a feasible solution may contain a path from a player $p \in S \cap T$ to itself, i.e., a path with no edge. We call such a path a trivial path. Any path with at least one edge is non-trivial.

Let $\Pi$ be any feasible solution of $G_M(S, T)$. The paths in $\Pi$ originate from a subset of $S$, which we call the sources, and terminate at a subset of $T$, which we call the sinks. We denote the sets of sources and sinks by $\text{source}(\Pi)$ and $\text{sink}(\Pi)$, respectively. A trivial path has only one node which is both its source and sink. From now on, we use $\Pi^+$ to denote the subset of non-trivial paths in $\Pi$.

### 2.2.2 Solving the problem

An optimal solution of $G_M(S, T)$ can be found by solving a maximum $s$-$t$ flow problem. Let $H$ be the $s$-$t$ flow network obtained from $G_M$ by adding a super source $s$ and directed edges from $s$ to all vertices in $S$, adding a super sink $t$ and directed edges from all vertices in $T$ to $t$, and setting the capacities of all edges to 1. It suffices to find an integral maximum flow in $H$. The paths in $G_M$ used by this maximum flow is an optimal solution of $G_M(S, T)$. Node-disjointness is guaranteed because, in $H$, every player has its in-degree at most one and every resource has its out-degree at most one.

Figure 1 gives an example. The squares represent players. The circles represent fat resources. In (a), the bold undirected edges form the maximum matching $M$. The two lower square nodes are unmatched and they form $\overline{P}_M$. The two upper nodes are matched and they form $P_M$. An optimal solution of $G_M(S, T)$ can be computed by finding an integral maximum $s$-$t$ flow in the network in (d). The shaded nodes and bold edges in (d) form a maximum $s$-$t$ flow. If you ignore $s$, $t$, and the edges incident to them, the remaining shaded nodes and bold edges form an optimal solution of $G_M(S, T)$, which contains one trivial path and one non-trivial path.

### 2.2.3 Non-trivial paths and the $\oplus$ operator

Let $\pi$ be a non-trivial path from $\overline{P}_M$ to $P$ in $G_M$. If we ignore the directions of edges in $\pi$, then $\pi$ is called an alternating path in the matching literature [16]: the first edge of $\pi$ does not belong to $M$, every other edge of $\pi$ belongs to $M$, and $\pi$ has an even number of edges. We use $M \oplus \pi$ to denote the result of flipping $\pi$, i.e., removing the edges in $\pi \cap M$ from the matching and adding the edges in $\pi \setminus M$ to the matching. $M \oplus \pi$ is a maximum matching of $G$. Moreover, $\text{source}(\pi)$ is unmatched in $M$ but it becomes matched in $M \oplus \pi$, and $\text{sink}(\pi)$ is matched in $M$ but it becomes unmatched in $M \oplus \pi$.

We can extend the above operation to any set $\Pi^+$ of node-disjoint non-trivial paths from $\overline{P}_M$ to $P$ in $G_M$. $\Pi^+$ can be regarded as a set of edges. We can form $M \oplus \Pi^+$ as in the previous paragraph, i.e., ignore the directions of edges in $\Pi^+$, remove the edges in $\Pi^+ \cap M$ from the matching, and add the edges in $\Pi^+ \setminus M$ to the matching. $M \oplus \Pi^+$ is a maximum matching of $G$. Players in $\text{source}(\Pi^+)$ are unmatched in $M$ but they become matched in $M \oplus \Pi^+$, and players in $\text{sink}(\Pi^+)$ are matched in $M$ but they become unmatched in $M \oplus \Pi^+$.

### 2.2.4 Feasible solutions of $G_M(S, T)$

The preliminary background above are sufficient for understanding how our algorithm works. However, in order to carry out a rigorous analysis, we have to delve into the feasible solutions of $G_M(S, T)$.
First let’s discuss the $\oplus$ operation further. Let $\Pi^+$ be any set of node-disjoint non-trivial paths from $\overline{P}_M$ to $P$ in $G_M$. $M \oplus \Pi^+$ is a maximum matching of $G$. Now consider $G_{M \oplus \Pi^+}$, the directed bipartite graph defined for the maximum matching $M \oplus \Pi^+$ as in section 2.2.2.

We claim that $G_{M \oplus \Pi^+}$ can be interpreted as a graph obtained from $G_M$ by reversing the edges used by $\Pi^+$: when edges in $\Pi^+ \cap M$ are removed from the matching and edges in $\Pi^+ \setminus M$ are added to the matching, their counterparts in $G_M$ are reversed. Figure 2 gives an example. In (a), the bold edges form a maximum matching $M$. (b) shows $G_M$, and $\Pi^+$ consists of the two bold paths, one from $p_3$ to $p_1$ and the other from $p_4$ to $p_2$. In (c), the bold edges form a maximum matching $M \oplus \Pi^+$ which is obtained from $M$ by flipping the edges in $\Pi^+$. $G_{M \oplus \Pi^+}$ is shown in (d). Comparing (b) and (d), it is easy to see that $G_{M \oplus \Pi^+}$ can be obtained from $G_M$ by reversing the edges in $\Pi^+$.

Now we are ready to establish a few properties for feasible solutions of $G_M(S,T)$ that will be used later in the analysis of our algorithm. As we explained in section 2.2.2 computing an optimal solution of $G_M(S,T)$ can be reduced to computing an integral maximum $s$-$t$ flow. Consequently, feasible solutions of $G_M(S,T)$ have some properties that are similar to those in the max-flow literature. Claims 2.1, 2.2 and 2.3 are very much like testing the optimality of a flow, augmenting a flow, and rerouting a flow, respectively. Recall that for a set $\Pi$ of node-disjoint paths, $\Pi^+$ is the subset of non-trivial paths in $\Pi$.

**Claim 2.1.** Let $\Pi$ be a feasible solution of $G_M(S,T)$. $\Pi$ is an optimal solution of $G_M(S,T)$ if and only if $G_{M \oplus \Pi^+}$ contains no path from $S \setminus \text{source}(\Pi)$ to $T \setminus \text{sink}(\Pi)$.

**Proof.** Let $H$ be the $s$-$t$ flow network constructed from $G_M$ as in section 2.2.2. One can extend $\Pi$ to a flow in $H$ with value $|\Pi|$ by pushing unit flows from $s$ to $\text{source}(\Pi)$, along $\Pi$, and then from $\text{sink}(\Pi)$ to $t$. Denote this flow by $F$. Let $H_F$ be the residual graph of $H$ with respect to $F$. $H_F$ can be obtained from $H$ by reversing the edges used by $F$. Recall that $G_{M \oplus \Pi^+}$ can be obtained from $G_M$ by reversing the edges in $\Pi^+$. Hence, if you ignore the $s$, $t$, and the edges incident to them, the remaining of the residual graph $H_F$ is exactly $G_{M \oplus \Pi^+}$. If there is a path $\pi$ in $G_{M \oplus \Pi^+}$ from $S \setminus \text{source}(\Pi)$ to $T \setminus \text{sink}(\Pi)$, then the concatenation $s \cdot \pi \cdot t$ is a path in the residual graph $H_F$. This means that we can augment $F$ using $s \cdot \pi \cdot t$ to increase the flow value and obtain more node-disjoints paths from $S$ to $T$ in $G_M$. (The augmentation may produce some unit-flow cycle(s) in $H$, and such cycles can be simply ignored when extracting the node-disjoint paths in $G_M$ from $S$ to $T$.) If such a path $\pi$ does not exist, then $F$ is a maximum flow which proves the optimality of $\Pi$.

The proof of Claim 2.1 immediately implies Claim 2.2.

**Claim 2.2.** Let $\Pi$ be a feasible solution of $G_M(S,T)$. Suppose that $G_{M \oplus \Pi^+}$ contains a path $\pi$ from $S \setminus \text{source}(\Pi)$ to $T \setminus \text{sink}(\Pi)$. We can use $\pi$ to augment $\Pi$ to a feasible solution $\Pi'$ of $G_M(S,T)$ such that $|\Pi'| = |\Pi| + 1$, the vertex set of $\Pi'$ is a subset of the vertices in $\Pi \cup \{\pi\}$, $\text{source}(\Pi') = \text{source}(\Pi) \cup \{\text{source}(\pi)\}$, and $\text{sink}(\Pi') = \text{sink}(\Pi) \cup \{\text{sink}(\pi)\}$.

---

Figure 2: An illustration of the $\oplus$ operation and the relation between $G_M$ and $G_{M \oplus \Pi^+}$.
are equal, implying that
\[ |G| = |H|. \]
Since we push a flow from sink \( s \) to a player in \( T \) along \( t \), before the update still receives a unit flow from \( s \), \( s \) cannot be the sink of a non-trivial path in \( \Pi \) set of \( \Pi \). Therefore, they cannot be the sink of a non-trivial path in \( \Pi \). Hence, \( t \cdot \pi \cdot t \) is a cycle in \( H \). We update \( F \) to another flow \( F' \) by sending a unit flow along \( t \cdot \pi \cdot t \). After removing all cycle(s) of flows in \( F' \) and removing all edges in \( F' \) incident to \( s \) and \( t \), we obtain a set \( \Pi' \) of node-disjoint paths from \( S \) to \( T \) in \( G_M \).

Since sending flow around a cycle does not change the total flow, the values of \( F \) and \( F' \) are equal, implying that \( |\Pi| = |\Pi'| \). By sending the unit flow around \( t \cdot \pi \cdot t \), we do not update the flow on directed edges incident to \( s \) in \( H \). Thus, every player who received a unit flow from \( s \) before the update still receives a unit flow from \( s \) afterwards, so \( source' = source \) in \( T \). Since \( sink' = sink \) in \( \Pi \), \( sink' \) no longer sends a unit flow to \( t \), \( sink' \) becomes a new sink. All the other sinks are not affected. We conclude that \( sink' = (sink \setminus \{source\}) \cup \{sink\} \).

Figure \( \ref{fig:example} \) gives an example of the proof of Claim 2.3. In (a), \( M \) consists of the bold edges. In (b), the bold edges form a flow \( F \). The bold edges other than those incident to \( s \) and \( t \) form a feasible solution \( \Pi \) of \( G_M(S, T) \) consisting of two paths: one from \( p_1 \) to \( p_2 \) and the other from

\begin{figure}[h]
\centering
\begin{subfigure}{0.24\textwidth}
\centering
\includegraphics[width=\textwidth]{a.png}
\caption{\( M \)}
\end{subfigure}
\begin{subfigure}{0.24\textwidth}
\centering
\includegraphics[width=\textwidth]{b.png}
\caption{The \( s-t \) flow network obtained from \( G_M \)}
\end{subfigure}
\begin{subfigure}{0.24\textwidth}
\centering
\includegraphics[width=\textwidth]{c.png}
\caption{Residual graph}
\end{subfigure}
\begin{subfigure}{0.24\textwidth}
\centering
\includegraphics[width=\textwidth]{d.png}
\caption{Augmented flow}
\end{subfigure}
\caption{An example of the proof of Claim 2.1 with \( S = \{p_1, p_5\} \) and \( T = \{p_2, p_5\} \)}
\end{figure}
For any maximum matchings

Claim 2.4. We derive some relations between one from $p$ of two node-disjoint paths from $S$ to $T$ in $G_M$: a trivial one from $p_5$ to itself and a non-trivial one from $p_1$ to $p_2$. The cycle $p_3p_5p_5p_2$ in $F'$ is ignored. $\Pi$ and $\Pi'$ satisfy Claim 2.3.

2.2.5 More properties

We derive some relations between $f_M(S, T)$'s for different choices of $M$, $S$, and $T$.

Claim 2.4. For any maximum matchings $M$ and $M'$ of $G$,

(i) $f_M(\overline{P}_M, \overline{P}_{M'}) = |\overline{P}_M| = |\overline{P}_{M'}|$, and

(ii) for every subset $T$ of players, $f_M(\overline{P}_M, T) = f_{M'}(\overline{P}_{M'}, T)$.

Proof. We first prove (i). Consider the symmetric difference $M \oplus M'$. It consists of cycles and alternating paths of even lengths. All these alternating paths are node-disjoint and appear as directed paths in $G_M$. Since these paths have even lengths, they are either from players to players or from resources to resources. Any node in $\overline{P}_M \setminus \overline{P}_{M'}$ (i.e., matched by $M'$ but not by $M$) must be an endpoint of some alternating path, and the other endpoint of the path must be a node in $\overline{P}_{M'} \setminus \overline{P}_M$ (i.e., matched by $M$ but not by $M'$). Any node in $\overline{P}_M \cap \overline{P}_{M'}$ has no incident edge in $M \oplus M'$, so it is a trivial path. Putting things together, there are node-disjoint paths (trivial or non-trivial) in $G_M$ from all nodes in $\overline{P}_M$ to $\overline{P}_{M'}$. So $f_M(\overline{P}_M, \overline{P}_{M'}) = |\overline{P}_M|.

Next we prove (ii). Let $\Pi$ be an optimal solution of $G_M(\overline{P}_M, T)$. Let $M''$ be the maximum matching obtained from $M$ by flipping the alternating paths in $\Pi^+$, i.e., $M'' = M \ominus \Pi^+$. After flipping the alternating paths, players in $\text{source}(\Pi^+)$ become matched and players in $\text{sink}(\Pi^+)$ become unmatched. Thus, $\overline{P}_{M''} = (\overline{P}_M \setminus \text{source}(\Pi^+)) \cup \text{sink}(\Pi^+) = (\overline{P}_M \setminus \text{source}(\Pi)) \cup \text{sink}(\Pi)$. The last step is due to the fact that each trivial path is a single vertex in $\overline{P}_M$ which serves as a source and a sink simultaneously. So $\text{sink}(\Pi) \subseteq \overline{P}_{M''}$. By (i), $f_M(\overline{P}_M, \overline{P}_{M''}) = |\overline{P}_M| = |\overline{P}_{M''}|$, which implies that $f_M(\overline{P}_M', \text{sink}(\Pi)) = |\Pi|$ (Just take an optimal solution of $G_M(\overline{P}_M', \overline{P}_{M''})$ and delete the paths ending at $\overline{P}_{M''} \setminus \text{sink}(\Pi)$). As $\text{sink}(\Pi) \subseteq T$ by definition, $f_M(\overline{P}_M', T) \geq f_M(\overline{P}_M', \text{sink}(\Pi)) = |\Pi|$. Recall that $\Pi$ is an optimal solution of $G_M(\overline{P}_M, T)$, so $f_M(\overline{P}_M, T) = |\Pi| \leq f_M(\overline{P}_M', T)$. We can similarly prove the other direction that $f_M(\overline{P}_M, T) \geq f_M(\overline{P}_M', T)$.

Claim 2.3 below states that if adding a player $p$ to $T$ increases $f_M(S, T)$, adding $p$ to any subset $T' \subset T$ increases $f_M(S, T')$ too.
Claim 2.5. Let $M$ be a maximum matching of $G$. Let $S$ be any subset of $\mathcal{P}_M$. Let $T$ be any subset of $P$. Let $p$ be an arbitrary player in $P$. If $f_M(S, T \cup \{p\}) = f_M(S, T) + 1$, then for every $T' \subseteq T$, $f_M(S, T' \cup \{p\}) = f_M(S, T') + 1$.

Proof. Let $\Pi_1$ be an optimal solution of $G_M(S, T)$. Note that $\Pi_1$ is also a feasible solution of $G_M(S, T \cup \{p\})$. Let $\Pi_2$ be an optimal solution of $G_M(S, T \cup \{p\})$ obtained by augmenting $\Pi_1$ (using Claim 2.2). Then, $\operatorname{sink}(\Pi_1) \subseteq \operatorname{sink}(\Pi_2)$. If $p \in \operatorname{sink}(\Pi_2)$, then $\operatorname{sink}(\Pi_1) \cup \{p\} \subseteq \operatorname{sink}(\Pi_2)$, implying that there are $|\Pi_1| + 1 = f_M(S, T') + 1$ node-disjoint paths from $S$ to $T' \cup \{p\}$, and thus establishing the claim. If $p \notin \operatorname{sink}(\Pi_2)$, then $\Pi_2$ is a feasible solution of $G_M(S, T)$. But then $f_M(S, T \cup \{p\}) = |\Pi_2| \leq f_M(S, T)$, a contradiction to the assumption. \hfill$\square$

3 The Algorithm

In this section, we present an algorithm which, given a partial allocation and an unsatisfied player $p_0$, computes a new partial allocation that satisfies $p_0$ and all the players that used to be satisfied. Recall that a partial allocation consists of a maximum matching of $G$ and a subset $\mathcal{E}$ of thin edges such that (i) no two edges in $M$ and $\mathcal{E}$ satisfy (i.e., cover) the same player, (ii) no two edges in $\mathcal{E}$ share any resource, (iii) every edge $(p, B) \in \mathcal{E}$ is minimal in the sense that every proper subset $B' \subset B$ has value less than $\tau/\lambda$.

Let $M$ and $\mathcal{E}$ be the maximum matching of $G$ and the set of thin edges in the current partial allocation, respectively. Let $p_0$ be an arbitrary player who is not yet satisfied.

3.1 Overview

To satisfy $p_0$, the simplest case is that we can find a minimal thin edge $(p_0, B_0)$ such that $B_0$ excludes all the resources covered by $\mathcal{E}$. (Recall that by definition of thin edges, $\operatorname{value}(B_0) \geq \tau/\lambda$.) We can extend the partial allocation by adding $(p_0, B_0)$ to $\mathcal{E}$.

More generally, we can use any thin edge $(q_0, B_0)$ such that $B_0$ meets the above requirements even if $q_0 \neq p_0$, provided that there is a path from $p_0$ to $q_0$ in $G_M$. If $q_0 \neq p_0$, such a path is an alternating path in $G$ with respect to $M$, and $q_0$ is matched by $M$. We can flip this path to match $p_0$ with a fat resource and then include $(q_0, B_0)$ in $\mathcal{E}$ to satisfy $q_0$.

The thin edge $(q_0, B_0)$ mentioned above may not always exist. In other words, some edges in $\mathcal{E}$ may share resources with $(q_0, B_0)$. Let $\{(p_1, B'_1), \ldots, (p_k, B'_k)\}$ be such thin edges in $\mathcal{E}$. In this situation, we say $(q_0, B_0)$ is blocked by $\{(p_1, B'_1), \ldots, (p_k, B'_k)\}$. In order to free up the resources held by $p_1, \ldots, p_k$, we need to satisfy each player $p_i$ with resources other than those in $B_0, B'_1, \ldots, B'_k$. Afterwards, we can satisfy $p_0$ as before. To record the different states of the algorithm, we initialize a stack to contain $(p_0, \emptyset)$ as the first layer and then create another layer on top that stores the sets $X_2 = \{(q_0, B_0)\}$ and $\mathcal{Y}_2 = \{(p_1, B'_1), \ldots, (p_k, B'_k)\}$ among other things for bookkeeping. We change our focus to satisfy the set of players $Y_2 = \{p_1, \ldots, p_k\}$.

To satisfy a player in $Y_2$ (by a new edge), we need to identify a minimal thin edge $(q_1, B_1)$ such that $B_1$ excludes the resources already covered by thin edges in the current stack because we don’t want $(q_1, B_1)$ to block or be blocked by any edges in the current stack. As to $q_1$, we require that $G_M$ contains two node-disjoint paths from $\{p_0\} \cup Y_2$ to $\{q_0, q_1\}$. If $(q_1, B_1)$ is blocked by some thin edges in $\mathcal{E}$, we initialize a set $X_3 = \{(q_1, B_1)\}$; otherwise, we initialize a set $\mathcal{I} = \{(q_1, B_1)\}$. Ideally, if $(q_1, B_1)$ is unblocked, we could immediately make some progress. Since there are two node-disjoint paths from $\{p_0\} \cup Y_2$ to $\{q_0, q_1\}$, $q_1$ is reachable from either $p_0$ or a player in $Y_2$. In the former case, we can satisfy $p_0$. In the latter case, the path from $Y_2$ to $q_1$ must be node-disjoint from the path from $p_0$ to $q_0$, so we can satisfy some player $p_i$ without affecting the alternating path from $p_0$ to $q_0$, and free up the resources previously held by that player. But we would not do so because, as argued in [1], in order to achieve a polynomial running time, we should let $\mathcal{I}$ grow bigger so that a larger progress can be made at once.
Since there are multiple players in $Y_2$ to be satisfied, we continue to look for another minimal thin edge $(q_i, B_i)$. We require that $B_i$ excludes the resources covered by thin edges in the current stack (including $X_3$ and $I$), and that $G_M$ contains one more node-disjoint paths from $\{p_0\} \cup Y_2$ after adding $q_i$ to the destination set. If $(q_i, B_i)$ is blocked by some thin edges in $E$, we add $(q_i, B_i)$ to $X_3$; otherwise, we add it to $I$. After collecting all such thin edges in $X_3$ and $I$, we construct the set $Y_3$ of thin edges in the current partial allocation that block $X_3$. Then, we add a new top layer to the stack that stores $X_3$ and $Y_3$ among other things for bookkeeping. Then, in order to free up the resources held by edges in $Y_3$ and to make edges in $X_3$ unblocked, we turn our attention to satisfying the players covered by $Y_3$ with new edges and so on. These repeated additions of layers to the stack constitute the build phase of the algorithm.

The build phase stops when we have enough thin edges in $I$ to satisfy a predetermined fraction of players covered by $Y_l$ for some $l$. We shrink the $l$-th layer and delete all layers above it. The above is repeated until $I$ is not large enough to satisfy the predetermined fraction of players covered by any $Y_l$ in the stack. These repeated removal of layers constitute the collapse phase of the algorithm. At the end of the collapse phase, we switch back to the build phase.

The alternation of build and collapse phases continues until we succeed in satisfying player $p_0$, our original goal, which is stored in the bottommost layer in the stack.

The lazy update (i.e., wait until $I$ is large enough before switching to the collapse phase) is not sufficient for achieving a polynomial running time. A greedy strategy is also needed. In [1], when a blocked thin edge $(q, B)$ is picked and added to $X_3$ for some $l$, $B$ is required to be a minimal set of value at least $\tau/2$, which is more than $\tau/\lambda$. Intuitively, if such an edge is blocked, it must be blocked by many edges. Hence, the strategy leads to a fast growth of the stack. We use a more aggressive strategy: we allow the value of $B$ to be as large as $\tau + \tau/\lambda$, and among all candidates, we pick the thin edge $(q, B)$ with (nearly) the largest value. Our strategy leads to a faster growth of the stack, and hence, a polynomial running time can be achieved for a smaller $\lambda$.

3.2 Notation and definitions

Let $M$ and $E$ denote the maximum matching of $G$ and the set of thin edges, respectively, that are used in the current partial allocation. Let $p_0$ denote the next player we want to satisfy.

A state of the algorithm consists of several components, namely, $M$, $E$, a stack of layers, and a global variable $I$ that stores a set of unblocked thin edge. The layers in the stack are indexed starting from 1 at the bottom. For $i \geq 1$, the $i$-th layer $L_i$ is a 4-tuple $(X_i, Y_i, d_i, z_i)$, where $X_i$ and $Y_i$ are sets of thin edges, and $d_i$ and $z_i$ are two numeric values that we will explain later. We use $I$, $X_i$ and $Y_i$ to denote the sets of players covered by edges in $I$, $X_i$ and $Y_i$, respectively. The set $I$ grows during the build phase and shrinks during the collapse phase, and $I$ changes correspondingly. The same is true for $X_i$, $X_i$, $Y_i$, and $Y_i$. For any $k \geq 1$, let $X_{<k}$ denote $\bigcup_{i=1}^{k} X_i$, $Y_{<k}$, $X_{<k}$, and $Y_{<k}$ are similarly defined.

The sets $X_i$ and $Y_i$ are defined inductively. At the beginning of the algorithm, $X_1 = \emptyset$, $Y_1 = \{(p_0, \emptyset)\}$, $d_1 = z_1 = 0$, and $I = \emptyset$. The first layer in the stack is thus $(\emptyset, \{(p_0, \emptyset)\}, 0, 0)$.

Let $\ell$ be the index of the topmost layer in the stack. Consider the construction of the $(\ell + 1)$-th layer in an execution of the build phase. When it first starts, $X_{<\ell + 1}$ is initialized to be empty. We say that a player $p$ is addable if

$$f_M(Y_{<\ell}, X_{<\ell + 1} \cup I \cup \{p\}) = f_M(Y_{<\ell}, X_{<\ell + 1} \cup I) + 1.$$ 

Note that this definition depends on $X_{<\ell + 1} \cup I$, so adding edges to $X_{<\ell + 1}$ and $I$ may affect the addability of players.

Given an addable player $p$, we say that a thin edge $(p, B)$ is addable if

$$\text{value}(B) \in [\tau/\lambda, \tau + \tau/\lambda] \text{ and } B \text{ excludes resources covered by } X_{<\ell + 1} \cup Y_{<\ell} \cup I.$$
An addable thin edge \((p, B)\) is \textbf{unblocked} if there exists a subset \(B' \subseteq B\) such that \(\text{value}(B') \geq \tau/\lambda\) and \(B'\) excludes resources used in \(E\). Otherwise, \((p, B)\) is \textbf{blocked}. During the construction of the \((\ell + 1)\)-th layer, the algorithm adds some blocked addable thin edges to \(X_{\ell + 1}\) and some unblocked addable thin edges to \(I\). When the growth of \(X_{\ell + 1}\) stops, the algorithm constructs \(Y_{\ell + 1}\) as the set of thin edges in \(E\) that share resource(s) with some edge(s) in \(X_{\ell + 1}\).

After constructing \(X_{\ell + 1}\) and \(Y_{\ell + 1}\) and growing \(I\), the values \(d_{\ell + 1}\) and \(z_{\ell + 1}\) are defined as

\[d_{\ell + 1} := f_M(Y_{\leq \ell}, X_{\leq \ell + 1} \cup I), \quad z_{\ell + 1} := |X_{\ell + 1}|.\]

The values \(d_{\ell + 1}\) and \(z_{\ell + 1}\) do not change once computed unless the layer \(L_{\ell + 1}\) is destructed in the collapse phase. That is, \(d_{\ell + 1}\) and \(z_{\ell + 1}\) record the values \(f_M(Y_{\leq \ell}, X_{\leq \ell + 1} \cup I)\) and \(|X_{\ell + 1}|\) at the time of construction. (Note that \(f_M(Y_{\leq \ell}, X_{\leq \ell + 1} \cup I)\) and \(|X_{\ell + 1}|\) may change subsequently.) The values \(d_{\ell + 1}\) and \(z_{\ell + 1}\) are introduced only for the analysis. They are not used by the algorithm.

Whenever we complete the construction of a new layer in the stack, we enter the collapse phase to check whether any existing layer is collapsible. If so, shrink the stack and update the current partial allocation \((M, E)\). We stay in the collapse phase until no layer is collapsible. If the stack has become empty, we are done as the player \(p_0\) has been satisfied. Otherwise, we reenter the build phase. We give the detailed specification of the build and collapse phases in the following subsections.

### 3.3 Build phase

Let \(\ell\) be the index of the topmost layer in the stack. Let \(M\) and \(E\) denote the maximum matching in \(G\) and the set of thin edges in the current partial allocation, respectively. We call the following routine \textsc{Build} to construct the next layer \(L_{k + 1}\).

\[
\textsc{Build}(M, E, \mathcal{I}, (L_1, \ldots, L_{\ell}))
\]

1. Initialize \(X_{\ell + 1}\) to be the empty set.
2. If there is an addable player \(p\) and an unblocked addable edge \((p, B)\), then:
   
   (a) take a minimal subset \(B' \subseteq B\) such that \(\text{value}(B') \geq \tau/\lambda\) and \(B'\) excludes the resources covered by \(E\) (we call \((p, B')\) a minimal unblocked addable edge),
   
   (b) add \((p, B')\) to \(I\),
   
   (c) repeat step 2
3. When we come to step 3 no unblocked addable edge is left. If there is no (blocked) addable edge, go to step 4. For each addable player \(p\) who is incident to at least one addable edge, identify one maximal blocked addable edge \((p, B)\) such that \(B \not\subset B'\) for any blocked addable edge \((p, B')\). Among the maximal blocked addable edges identified, pick the one with the largest value, and add it to \(X_{\ell + 1}\). Then repeat step 3.
4. At this point, the construction of \(X_{\ell + 1}\) is complete. Let \(Y_{\ell + 1}\) be the set of the thin edges in \(E\) that share resource(s) with some thin edge(s) in \(X_{\ell + 1}\).
5. Compute \(d_{\ell + 1} := f_M(Y_{\leq \ell}, X_{\leq \ell + 1} \cup I)\) and \(z_{\ell + 1} := |X_{\ell + 1}|\).
6. Push the new layer \(L_{\ell + 1} = (X_{\ell + 1}, Y_{\ell + 1}, d_{\ell + 1}, z_{\ell + 1})\) onto the stack. \(\ell := \ell + 1\).

\textsc{Build} differs from its counterpart in \[1\] in several places, particularly in step 3. First, we require blocked addable edges to be maximal while only minimal addable edges of value at least \(\tau/2\) are considered in \[1\]. Second, when adding addable edges to \(X_{\ell + 1}\), we pick the one with (nearly) the largest value. In contrast, one arbitrary addable edge is picked in \[1\].
Lemma 3.2. Build maintains invariants 1–7 in Table 1.

Proof. Suppose that the invariants hold before Build constructs the new topmost layer $L_{\ell+1}$. It suffices to check the invariants after the construction of $L_{\ell+1}$. Invariants 1 and 2 are clearly preserved by the working of Build.

One may wonder, instead of identifying a maximal blocked addable edge for each player, whether it is better to identify a maximum blocked addable edge for the algorithm, i.e., the blocked addable edge with the largest value. However, finding the blocked addable edge with the largest value for $p$ is an instance of the NP-hard knapsack problem. Maximal blocked addable edges are sufficient for our purposes.

Is it possible that $X_{\ell+1} = \emptyset$ and so $Y_{\ell+1} = \emptyset$? We will establish the result, Lemma 4.1 in Section 5.2, that if $|Y_{\ell+1}| < \sqrt{\mu} |Y_{\ell}|$ for some $i$, then some layer below $L_{\ell+1}$ is collapsible. Therefore, if $Y_{\ell+1}$ is empty, then some layer below $L_{\ell+1}$ must be collapsible, the algorithm will enter the collapse phase next, and $L_{\ell+1}$ will be removed.

**Lemma 3.1.** Build runs in $\text{poly}(m, n)$ time.

Proof. It suffices to show that steps 2 and 3 run in polynomial time. Two maximum flow computations tell us whether a player $p$ is addable. Suppose so. We start with the thin edge $(p, B)$ where $B = \emptyset$. Let $R_p$ denote the set of thin resources that are desired by $p$. First, we incrementally insert to $B$ thin resources from $R_p$ that appear in neither the current partial allocation nor $X_{\ell+1} \cup Y_{\ell} \cup \mathcal{I}$. If $\text{value}(B)$ becomes greater than or equal to $\tau/\lambda$, then we must be in step 2 and $(p, B)$ is a minimal unblocked addable edge that can be added to $\mathcal{I}$. Suppose that $\text{value}(B) < \tau/\lambda$ after the incremental insertion stops. Then, $p$ has no unblocked addable edge. If we are in step 3 we continue to add to $B$ thin resources from $R_p$ that appear in the current partial allocation but not in $X_{\ell+1} \cup Y_{\ell} \cup \mathcal{I}$. If $\text{value}(B) < \tau/\lambda$ when the incremental insertion stops, then $p$ has no addable edge. Otherwise, we continue until $\text{value}(B)$ is about to exceed $\tau + \tau/\lambda$ or we have examined all thin resources in $R_p$, whichever happens earlier. In either case, the final $\text{value}(B)$ is in the range $[\tau/\lambda, \tau + \tau/\lambda]$ and $(p, B)$ is a maximal blocked addable edge.

Table 1 shows the invariants that will be used in the analysis of the algorithm. Clearly, they all hold at the start of the algorithm, i.e., $\ell = 1$, $X_1 = \emptyset$, $\mathcal{I} = \emptyset$, $Y_1 = \{(p_0, \emptyset)\}$, and $d_1 = z_1 = 0$. We show that they are maintained by Build.
Consider invariant 3. It holds for \( i \in [1, \ell] \) because none of \( E, X_i \), and \( Y_i \) is changed. Since all edges in \( E \) that share some resource(s) with some edge(s) in \( X_{i+1} \) are added to \( Y_{i+1} \), invariant 3 also holds for \( i = \ell + 1 \).

Consider invariant 4. By induction assumption, \( Y_2, \ldots, Y_\ell \) are disjoint subsets of \( E \). By construction, \( Y_{i+1} \subseteq E \). If an edge \( e \in Y_{i+1} \) belongs to \( Y_\omega \), then the resources covered by \( e \) must be excluded by all edges in \( X_{i+1} \) by the definition of addable edges. But \( e \) must share resource(s) with some edge(s) in \( X_{i+1} \) in order that \( e \in Y_{i+1} \), a contradiction. So \( Y_2, \ldots, Y_{\ell+1} \) are disjoint subsets of \( E \).

Invariant 5 follows from invariants 3 and 4 and the fact that edges in \( E \) do not share any resource.

Let \( I' \) denote the version of \( I \) immediately before the execution of BUILD to construct \( L_{\ell+1} \). Let \( I'' \) denote the set of unblocked addable edges inserted into \( I' \) for constructing \( L_{\ell+1} \). Correspondingly, \( I' \) and \( I'' \) denote the set of players covered by \( I' \) and \( I'' \), respectively.

Consider invariant 6. We have \( f_M(Y_{\ell-1}, I') = |I'| \) as invariant 6 is assumed to hold before BUILD executes. Each player in \( X_{i+1} \) and \( I' \) is determined to be addable, i.e., adding such a player to \( X_{i+1} \cup I \) increases the value of \( f_M(Y_{\ell-1}, X_{i+1} \cup I) \) by one. Then, Claim 2.4 implies that \( f_M(Y_{\ell}, I' \cup I'') = f_M(Y_{\ell}, I') + |I''| \). Recall that \( f_M(Y_{\ell-1}, I') = |I'| \), which implies \( f_M(Y_{\ell}, I') = |I'| \). Thus \( f_M(Y_{\ell}, I' \cup I'') = |I'| + |I''| = |I' \cup I''| \), preserving invariant 6.

Consider invariant 7. BUILD does not change \( Y_{\leq i} \) and \( X_{i+1} \) for any \( i \in [1, \ell \! - \! 1] \), and BUILD does not delete any edge from \( I \). Therefore, for all \( i \in [1, \ell - 1] \), \( f_M(Y_{\leq i}, X_{i+1} \cup I) \) cannot decrease and so \( f_M(Y_{\leq i}, X_{i+1} \cup I) \) remains larger than or equal to \( d_{i+1} \). By construction, BUILD sets \( d_{\ell+1} \) := \( f_M(Y_{\ell}, X_{\ell+1} \cup I) \). Invariant 7 is preserved.

### 3.4 Collapse phase

Let \( M \) be the maximum matching in \( G \) in the current partial allocation. Let \( L_1, L_2, \ldots, L_\ell \) be the layers currently in the stack from bottom to top. We need to tell whether a layer can be collapsed, and this requires a certain decomposition of \( I \).

**Collapsibility.** Let \( I_1 \cup I_2 \cup \cdots \cup I_\ell \) be some partition of \( I \). Let \( I_i \) denote the set of players covered by \( I_i \). We use \( I_{\leq j} \) and \( I_{\geq j} \) to denote \( \bigcup_{i=1}^{j} I_i \) and \( \bigcup_{i=j}^{\ell} I_i \), respectively. Note that \( |I_i| = |I_i| \) by invariant 1 in Table 1. The set \( I_1 \cup I_2 \cup \cdots \cup I_\ell \) is a canonical decomposition of \( I \) if

\[
\forall i \in [1, \ell], \quad f_M(Y_{\leq i}, I_{\leq i}) = f_M(Y_{\leq i}, I) = |I_{\leq i}| = |I_{\leq i}|.
\]

**Lemma 3.3.** In \( \text{poly}(\ell, m, n) \) time, one can compute a canonical decomposition \( I_1 \cup I_2 \cup \cdots \cup I_\ell \) of \( I \) and a canonical solution of \( G_M(Y_{\leq i}, I) \) that can be partitioned into a disjoint union \( \Gamma_1 \cup \Gamma_2 \cup \cdots \cup \Gamma_\ell \) such that for every \( i \in [1, \ell] \), \( \Gamma_i \) is a set of \( |I_i| \) paths from \( Y_i \) to \( I_i \).

**Proof.** We first compute an optimal solution \( \Pi_1 \) of \( G_M(Y_1, I) \) by successive augmentations (using Claim 2.3). So \( |\Pi_1| = f_M(Y_1, I) \). For \( j = 2, \ldots, \ell \), we compute an optimal solution \( \Pi_j \) of \( G_M(Y_{\leq j}, I) \) by successively augmenting \( \Pi_{j-1} \). By Claim 2.3, \( \text{source}(\Pi_{j-1}) \subseteq \text{source}(\Pi_j) \). Therefore, we inductively maintain the property that for all \( i \in [1, j] \), \( \Pi_j \) contains \( |\Pi_i| = f_M(Y_{\leq i}, I) \) node-disjoint paths from \( Y_{\leq i} \) to \( I \). In the end, \( I = \text{sink}(\Pi_\ell) \) by invariant 6 in Table 1. We obtain the canonical decomposition and canonical solution as follows: for every \( i \in [1, \ell] \), let \( \Gamma_i \) be the subset of paths in \( \Pi_\ell \) from \( Y_i \) to \( I_i \), let \( I_i = \text{sink}(\Gamma_i) \), and let \( I_i \) be the subset of edges in \( I \) that cover the players in \( I_i \). The \( \Gamma_i \)'s are disjoint because the \( Y_i \)'s are disjoint by invariant 4 in Table 1.

Note that \( I_\ell \) and \( \Gamma_\ell \) are actually empty because the invariant 6 in Table 1 ensures that \( f_M(Y_{\leq \ell-1}, I) = |I| \) when we enter the collapse phase.

Consider \( \Gamma_i \). The sources (which are also sinks) of the trivial paths in \( \Gamma_i \) are players covered by unblocked addable thin edges in \( I_i \), and can be satisfied by these thin edges. Recall that
the non-trivial paths in \( \Gamma_i \) are alternating paths with respect to \( M \). If we flip these alternating paths, their sources can be satisfied by fat resources. Their sinks can be satisfied by thin edges in \( \mathcal{I}_i \). The subset of \( \mathcal{Y}_i \) that cover \( \text{source}(\Gamma_i) \) should then be removed from \( \mathcal{E} \) because the players in \( \text{source}(\Gamma_i) \) are now satisfied either by a fat edge or a thin edge from \( \mathcal{I}_i \). This subset of \( \mathcal{Y}_i \) should also be removed from \( \mathcal{X}_i \) because they no longer block edges in \( \mathcal{X}_i \). A layer is collapsible if a certain portion of its blocking edges can be removed. More precisely, for any \( i \in [1, \ell] \), \( L_i \) is collapsible if:

\[ \exists \text{ a canonical decomposition } \mathcal{I}_1 \cup \mathcal{I}_2 \cup \ldots \mathcal{I}_\ell \text{ of } \mathcal{I} \text{ such that } |I_i| \geq \mu|Y_i|, \text{where } \mu \text{ is a constant that will be defined later.} \]

By Lemma 3.3, the collapsibility of a layer can be checked in \( \text{poly}(\ell, m, n) \) time.

**Collapse layers.** Let \((L_1, \ldots, L_\ell)\) denote the current layers in the stack. The routine \text{Collapse} below checks whether some layer is collapsible, and if yes, it collapses layers in the stack until no collapsible layer is left. The execution of \text{Collapse} may update the stack, \( \mathcal{I} \), and the current partial allocation, including both the maximum matching \( M \) in \( G \) and the set of thin edges \( \mathcal{E} \) in the partial allocation. \text{Collapse} works in the same manner as its counterpart in [1], but there are small differences in the presentation.

\text{Collapse}(\mathcal{M}, \mathcal{E}, \mathcal{I}, (L_1, \cdots, L_\ell))

1. Compute a canonical decomposition \( \mathcal{I}_1 \cup \mathcal{I}_2 \cup \cdots \mathcal{I}_\ell \) of \( \mathcal{I} \) and a canonical solution \( \Gamma_1 \cup \Gamma_2 \cup \cdots \Gamma_\ell \) of \( G_M(\mathcal{Y}_{\leq \ell}, I) \). If no layer is collapsible, go to build phase. Otherwise, let \( L_1 \) be the collapsible layer with the smallest index \( t \).

2. Remove all layers above \( L_t \) from the stack. Set \( \mathcal{I} := \mathcal{I}_{< t-1} \).

3. Recall that \( \text{source}(\Gamma_t) \subseteq \mathcal{Y}_t \) by Lemma 3.3. Let \( \mathcal{V} \) denote the subset of edges in \( \mathcal{Y}_t \) that cover players in \( \text{source}(\Gamma_t) \).
   
   (a) Update the maximum matching \( M \) by flipping the non-trivial paths in \( \Gamma_t \), i.e., set \( M := M \oplus \Gamma_t^+ \) where \( \Gamma_t^- \) is the set of non-trivial paths in \( \Gamma_t \). This update matches the sources of non-trivial paths in \( \Gamma_t \), and makes their sinks unmatched.
   
   (b) Add to \( \mathcal{E} \) the edges in \( \mathcal{I}_t \), i.e., set \( \mathcal{E} := \mathcal{E} \cup \mathcal{I}_t \). The sinks of non-trivial paths in \( \Gamma_t \) and the sources (which are also sinks) of trivial paths \( \Gamma_t \) are now satisfied by edges in \( \mathcal{I}_t \).
   
   (c) Each player in \( \text{source}(\Gamma_t) \) is now satisfied by either a fat resource or a thin edge from \( \mathcal{I}_t \). If \( t = 1 \), then \( p_0 \) is already satisfied, and the algorithm terminates. Assume that \( t \geq 2 \). Then \( \mathcal{V} \subseteq \mathcal{Y}_t \subseteq \mathcal{E} \). Edges in \( \mathcal{V} \) can be removed from \( \mathcal{E} \), so set \( \mathcal{E} := \mathcal{E} \setminus \mathcal{V} \). Consequently, edges in \( \mathcal{V} \) no longer block edges in \( \mathcal{X}_t \), so set \( \mathcal{Y}_t := \mathcal{Y}_t \setminus \mathcal{V} \).

4. If \( t \geq 2 \), we need to update \( \mathcal{X}_t \) because the removal of \( \mathcal{V} \) from \( \mathcal{E} \) (and hence \( \mathcal{Y}_t \)) may make some edges in \( \mathcal{X}_t \) unblocked. For each edge \((p, B) \in \mathcal{X}_t \) that becomes unblocked, perform the following operations:
   
   (a) Remove \((p, B) \) from \( \mathcal{X}_t \).
   
   (b) If \( f_M(\mathcal{Y}_{< t-1}, I \cup \{p\}) = f_M(\mathcal{Y}_{< t-1}, I) + 1 \), then add \((p, B') \) to \( \mathcal{I} \), where \( B' \) is an arbitrary minimal subset of \( B \) such that \( \text{value}(B') \geq \tau/\lambda \) and \( B' \) excludes the resources used in \( \mathcal{E} \).

5. If \( t = 1 \), step 3 already satisfied the player \( p_0 \) in the bottommost layer in the stack, so the algorithm terminates. Otherwise, update \( t := t + 1 \) and go back to step 1.
When we shrink a layer in the stack, the basis for generating the layers above it is no longer valid, and therefore, the easiest handling is to remove all such layers. In turn, it means that we should shrink the lowest collapsible layer $L_t$ in the stack, which explains the choice of $t$ in step 3. In one iteration, the layers $L_1, \ldots, L_{t-1}$ are preserved, $L_t$ is updated, and the updated $L_t$ becomes the topmost layer in the stack. One may wonder if it is possible that $Y_t$ becomes empty after step 3 and so $X_t$ also becomes empty after step 4. We will establish the result, Lemma 3.4 in Section 5.2, that if $|Y_{i+1}| < \sqrt{\mu} Y_{i+1}[i]$ for some $i$, then some layer below the $(i+1)$-th layer is collapsible. Therefore, if $Y_t$ becomes empty, some layer below $L_t$ is collapsible and so $L_t$ will be removed in the next iteration of steps 2-4 of COLLAPSE.

We need to show that invariants 1–7 in Table 7 are satisfied after COLLAPSE terminates so that we are ready to enter the build phase.

**Lemma 3.4.** COLLAPSE maintains invariants 1–7 in Table 7.

**Proof.** It suffices to show that invariants 1–7 are preserved after collapsing the lowest collapsible layer $L_t$ in steps 2-4. Clearly, invariant 1 is preserved by the working of COLLAPSE. Consider invariant 2. Since $T$ is updated to $T_{\leq t-1}$, the inclusion of $T_t$ in $E$ in step 3(b) does not break invariant 2. Step 3(c) deletes edges from $E$, which clearly preserves invariant 2. In step 3(b), all edges added to $T$ exclude the resources used by $E$, so invariant 2 is preserved. Let $M$ denote the maximum matching in the current partial allocation. Let $(L_1, L_2, \ldots, L_t)$ be the layers in the stack before step 2. It suffices to consider $(L_1, L_2, \ldots, L_t)$ for invariants 3–7 as $L_t$ will become stack top after one iteration of steps 2-4.

Consider invariant 3. Only steps 3(b) and 3(c) may affect it. In step 3(b), the edges that are added to $E$ are from $T$. By invariant 1, no edge from $T$ shares any resource with any edge in $X_{\leq t}$, so invariant 3 is preserved. In step 3(c), we may delete edges from $Y_t$, but these edges are also deleted from $E$, so invariant 3 still holds.

Invariant 4 holds because COLLAPSE never adds any new edge to any $Y_t$, and when some edges are removed from $E$ by COLLAPSE, they are also removed from $Y_t$.

Invariant 5 follows from invariants 3 and 4 and the fact that edges in $E$ do not share any resource.

Consider invariant 6. Since the topmost layer in the stack is going to be $L_t$, invariant 6 is concerned with $f_M(Y_{\leq t-1}, I)$. By the definition of a canonical solution, $T_{t-1}$ certifies that $f_M(Y_{\leq t-1}, I_{\leq t-1}) = |I_{\leq t-1}| \iff f_M(Y_{\leq t-1}, I) = |I|$ as $T := T_{\leq t}$ in step 2. In step 3 only step 3(a) may affect the equality $f_M(Y_{\leq t-1}, I) = |I|$ because $M$ may be changed by flipping the alternating paths in $T_t^+$. Let $M'$ denote the updated matching. $T_t^+$ is node-disjoint from $T_{t-1}$, so flipping the alternating paths in $T_t^+$ does not affect $T_{t-1}$. Therefore, $T_{t-1}$ still certifies that $f_{M'}(Y_{\leq t-1}, I_{\leq t-1}) = |I_{\leq t-1}| \iff f_{M'}(Y_{\leq t-1}, I) = |I|$, so step 3 preserves invariant 6.

In step 3 a new edge $(p, B)$ is inserted into $T$ only if $f_M(Y_{\leq t-1}, I \cup \{p\}) > f_M(Y_{\leq t-1}, I)$. Thus, when the size of $T$ increases by one, $f_M(Y_{\leq t-1}, I)$ also increases by one. So step 3 preserves invariant 6.

Consider invariant 7. Since it holds before COLLAPSE, we conclude that for all $i \in [1, t-1]$, $f_M(Y_{\leq i}, X_{\leq i+1} \cup I) \geq d_{i+1}$ before step 2. Since $L_t$ is going to be the topmost layer, we only need to show that these inequalities hold after steps 2-4. Take any index $i \in [1, t-1]$. We claim that, before the execution of steps 2-4, $G_M(Y_{\leq i}, X_{\leq i+1} \cup I) \geq G_M(Y_{\leq i}, X_{\leq i+1} \cup I)$ and $G_M(Y_{\leq i}, X_{\leq i+1} \cup I) \geq |I_{\leq i}| \geq d_{i+1}$ before the execution of step 2. Step 2 sets $T := T_{\leq t-1}$, and so after that, $\Pi_t$ remains an optimal solution of $G_M(Y_{\leq i}, X_{\leq i+1} \cup I)$, and hence $f_M(Y_{\leq i}, X_{\leq i+1} \cup I) \geq |\Pi_t| \geq d_{i+1}$. Step 3 changes the matching $M$ by flipping the alternating paths in $T_t^+$. Let $M'$ denote the updated matching. Since $T_t^+$ is node-disjoint from $\Pi_t$ by the claim, flipping the paths in $T_t^+$ does not affect $\Pi_t$, meaning that $\Pi_t$ is still a feasible solution of $G_{M'}(Y_{\leq i}, X_{\leq i+1} \cup I)$. Thus, $f_{M'}(Y_{\leq i}, X_{\leq i+1} \cup I) \geq |\Pi_t| = f_M(Y_{\leq i}, X_{\leq i+1} \cup I) \geq d_{i+1}$ after step 3. In step 4 the removal of edges from
\(X_t\) can only affect \(f_M'(Y_{<t-1}, X_{<t} \cup I)\) and so for \(i \in [1, t - 2]\), \(f_M'(Y_{<i}, X_{<i+1} \cup I) \geq d_{i+1}\) after step \([3]\). If \(f_M'(Y_{<t-1}, X_{<t} \cup I)\) decreases after removing an edge \((p, B)\) from \(X_t\), that is, \(f_M'(Y_{<t-1}, X_{<t} \cup I) = f_M'(Y_{<t-1}, (X_{<t} \cup I) \setminus \{p\}) + 1\), then when we reach step \([3]\) Claim \(2.5\) will imply that \(f_M'(Y_{<t-1}, J \cup \{p\}) = f_M'(Y_{<t-1}, J) + 1\), and so step \([3]\) will add \(p\) to \(I\). Afterwards, \(f_M'(Y_{<t-1}, X_{<t} \cup I)\) returns to its value prior to the removal of \((p, B)\) from \(X_t\). As a result, invariant \(7\) holds after step \([3]\).

It remains to prove the claim: before executing steps \(2-4\), for \(i \in [1, t-1]\), \(G_M(Y_{<i}, X_{<i+1} \cup I_{<i})\) and \(G_M(Y_{<i}, X_{<i+1} \cup I)\) have a common optimal solution \(\Pi_i\) that is node-disjoint from \(\Gamma_i\). For convenience, define \(\Gamma_{<i} = \Gamma_1 \cup \Gamma_2 \ldots \cup \Gamma_i\). By the definition of a canonical solution, \(\Gamma_{<i}\) is an optimal solution of \(G_M(Y_{<i}, I)\). We augment \(\Gamma_{<i}\) successively (using Claim \(2.2\)) to obtain \(\Pi_i\) as an optimal solution of \(G_M(Y_{<i}, X_{<i+1} \cup I)\). It remains to show that \(\Pi_i\) is an optimal solution of \(G_M(Y_{<i}, X_{<i+1} \cup I_{<i})\) and that \(\Pi_i\) is node-disjoint from \(\Gamma_i\). By the definition of canonical solution, \(I_{<i} = \text{sink}(\Pi_{<i})\). Next we prove that \(\Pi_i\) is node-disjoint from \(\Gamma_i\). Recall that \(\Pi_i\) is obtained from \(\Gamma_{<i}\) by successive augmentations using Claim \(2.2\). Let \(\{\Gamma_{<i} = \Phi_1, \Phi_2, \ldots, \Phi_k = \Pi_i\}\) be the successive sets of node-disjoint paths that are obtained during the successive augmentations. By Claim \(2.2\) \(I_{<i} = \text{sink}(\Gamma_{<i}) \subseteq \text{sink}(\Phi_j)\) for all \(j \in [1, k]\). Note that \(\Phi_1 = \Gamma_{<i}\) is node-disjoint from \(\Gamma_i\) and \(\Phi_{j-1}\) is node-disjoint from \(\Gamma_{j-1}\) for some \(j \in [2, k]\). We argue that \(\Phi_j\) is node-disjoint from \(\Gamma_i\), and then so is \(\Phi_{j-1}\) for some \(j \in [2, k]\). We argue that \(\Phi_j\) is node-disjoint from \(\Gamma_i\), and then so is \(\Phi_k = \Pi_i\) by induction. Recall that \(\Phi_j^+\) is the set of non-trivial paths in \(\Phi_{j-1}\). Consider \(G_M(\Phi_{j-1}^+)\). Let \(\gamma\) be the path in \(G_M(\Phi_{j-1}^+)\) that we use to augment \(\Phi_{j-1}\) to produce \(\Phi_j\). Since \(\Phi_{j-1}\) is node-disjoint from \(\Gamma_{j-1}\), \(G_M(\Phi_{j-1}^+)\) contains \(\Gamma_i\). If \(\gamma\) is node-disjoint from \(\Gamma_i\), then by Claim \(2.2\) \(\Phi_j\) must be node-disjoint from \(\Gamma_i\) because the vertex set of \(\Phi_j\) is a subset of the vertices of \(\Phi_{j-1}\) and \(\gamma\). If \(\gamma\) shares a node with some path in \(\Gamma_i\), then by switching at that shared node, we have a path \(\gamma'\) in \(G_M(\Phi_{j-1}^+)\) that originates from \(Y_{<i}\) and ends at a sink of \(\Gamma_i\) which is in \(I_i\). It means that if we augment \(\Phi_{j-1}\) using \(\gamma'\), we would obtain a feasible solution of \(G_M(Y_{<i}, X_{<i+1} \cup I)\) whose sinks contains all players of \(I_{<i}\) and some player in \(I_i\). This allows us to extract a feasible solution of \(G_M(Y_{<i}, I)\) whose sinks contains all players of \(I_{<i}\) and some player in \(I_i\). But then \(f_M(Y_{<i}, I) > |I_{<i}|\), a contradiction to the definition of canonical decomposition.

In the following, we prove two more properties that result from the invariants and the working of the algorithm. They will be used later in the analysis of the approximation ratio.

**Lemma 3.5.** Let \((L_1, \ldots, L_\ell)\) be the stack for an arbitrary state of the algorithm.

(i) For every \(j \in [1, \ell]\), \(|X_j| \leq z_j\).

(ii) For every \(j \in [1, \ell]\), \(d_j \geq \sum_{i=1}^j z_i\).

*Proof.* Recall that \(z_j := |X_j|\) after BUILD completes the construction of \(L_j\) for all \(j \in [1, \ell]\). Afterwards, \(z_j\) remains unchanged until \(L_j\) is removed from the stack. \(X_j\) may shrink in step \([4]\) of COLLAPSE, but it never grows. As a result, \(z_j \geq |X_j|\) at all times.

We prove (ii) by induction on the chronological order of executing BUILD and COLLAPSE. Initially, \(L_1\) is the only layer in the stack and \(d_1 = z_1 = 0\) because both \(\mathcal{I}\) and \(X_1\) are empty sets. Thus, (ii) holds at the beginning. Suppose that we build a new layer \(L_{k+1}\) in the build phase. Let \(\mathcal{I}'\) be the set of unblocked addable edges newly added to \(\mathcal{I}\) during the construction of \(L_{k+1}\), and let \(\mathcal{I}'\) be the set of players covered by \(\mathcal{I}'\). Then we have \(d_{k+1} = f_M(Y_{<k}, X_{<k+1} \cup I \cup \mathcal{I}')\).
Thus, \textsc{Build} preserves (ii). \textsc{Collapse} has no effect on (ii) because the values \(d_i\)’s and \(z_i\)’s will not change once they are computed until layer \(L_i\) is removed.

\textbf{Lemma 3.6.} Let \((L_1, \ldots, L_\ell)\) be the stack for an arbitrary state of the algorithm. If \(L_i\) is not collapsible for all \(i \in [1, \ell - 1]\), the following properties are satisfied.

\begin{enumerate}
\item \(|I| < \mu |Y_{\ell \leq \ell - 1}|.\)
\item For every \(j \in [1, \ell - 1]\), \(|X_{\ell \leq j + 1}| > \sum_{i=1}^{j+1} z_i - \mu |Y_{\ell \leq j}|.\)
\end{enumerate}

\textit{Proof.} By invariant 6 in Table 1 \(f_M(Y_{\ell \leq \ell - 1}, I) = |I|\). Hence, in the canonical decomposition of \(I\), we have \(I_{\ell \leq \ell - 1} = I\). If \(|I_{\ell \leq \ell - 1}| = |I| \geq \mu |Y_{\ell \leq \ell - 1}|, \) by the pigeonhole principle, there exists an index \(j \in [1, \ell - 1]\) such that \(|I_j| \geq \mu |Y_j|\). But then layer \(L_j\) is collapsible, a contradiction.

Consider (ii). Assume to the contrary that there exists \(k \in [1, \ell - 1]\) such that \(|X_{\ell \leq k+1}| \leq \sum_{i=1}^{k+1} z_i - \mu |Y_k|\). Equivalently, \(\sum_{i=1}^{k+1} z_i \geq |X_{\ell \leq k+1}| + \mu |Y_k|\). By invariant 7 in Table 1 and Lemma 3.5(ii), \(f_M(Y_{\ell \leq k}, X_{\ell \leq k+1} \cup I) \geq d_{k+1} \geq \sum_{i=1}^{k+1} z_i \geq |X_{\ell \leq k+1}| + \mu |Y_{\ell \leq k}|.\) So any optimal solution of \(G_M(Y_{\ell \leq k}, X_{\ell \leq k+1} \cup I)\) contains at least \(\mu |Y_{\ell \leq k}|\) node-disjoint paths from \(Y_{\ell \leq k}\) to \(I\).

It follows that \(f_M(Y_{\ell \leq k}, I) \geq \mu |Y_{\ell \leq k}|.\) By the definition of canonical decomposition, \(|I_{\ell \leq k}| = f_M(Y_{\ell \leq k}, I) \geq \mu |Y_{\ell \leq k}|.\) By the pigeonhole principle, there exists some \(j \in [1, k]\) such that \(|I_j| \geq \mu |Y_j|\). But then layer \(L_j\) is collapsible, a contradiction.

\section{Polynomial running time and binary search}

It is clear that each call of \textsc{Build} and \textsc{Collapse} runs in time polynomial in \(\ell, m, n\) and \(\ell\). So we need to give a bound on \(\ell\) (the number of layers in the stack) and the total number of calls of \textsc{Build} and \textsc{Collapse}.

Lemma 4.1 below is the key to obtaining such bounds. Recall that \(\mu\) is the constant we use to determine the collapsibility of layers, i.e., \(L_i\) is collapsible if \(|I_i| \geq \mu |Y_i|\). By Lemma 4.1 if no layer is collapsible in the current stack, then the size of the each layer is at least a constant fraction of the total size of all layers below it. This guarantees that the algorithm never gets stuck: it can either build a new non-empty layer or collapse some layer. It will also allow us to obtain a logarithmic bound on the maximum number of layers. The proof of Lemma 4.1 is quite involved and it requires establishing several technical results and the use of competing players. We defer the proof to Section 5.

\textbf{Lemma 4.1.} Assume that the values \(\tau\) and \(\lambda\) used by the algorithm satisfy the relations \(\tau \leq \tau^*\) and \(\lambda = 6 + \delta\) for an arbitrary constant \(\delta \in (0, 1)\). There exists a constant \(\mu \in (0, 1)\) dependent on \(\delta\) such that for any state \((M, E, I, (L_1, \ldots, L_\ell))\) of the algorithm, if \(|Y_{i+1}| < \sqrt{\mu |Y_{\ell \leq i}|}\) for some \(i \in [1, \ell - 1]\), then some layer below \(L_{i+1}\) must be collapsible.

With Lemma 4.1, an argument similar to that in \[1, \text{Lemmas 4.10 and 4.11}\] can show that given a partial allocation, our algorithm can extend it to satisfy one more player in polynomial
time. By repeating the algorithm at most \( n \) times, we can extend a maximum matching of \( G \) to an allocation that satisfies all the players.

We sketch the argument in [I] in the following. (The counterpart of Lemma [I.1] in [I] uses \( \lambda \approx 12.325 + \delta \).) Recall that \( P \) is the set of all players. As \( |Y_{\leq i}| \leq |P| \) and \( |Y_{\leq i}| \) grows by a factor \( 1 + \frac{1}{\mu} \) layer by layer when no layer is collapsible, the number of layers in the stack is at most \( \log_{1+\frac{1}{\mu}}|P| \). Define a signature vector \( (s_1, s_2, \ldots, s_i, \infty) \) such that \( s_i := \lfloor \log_{1/(1-\mu)}|Y_i|^{1/2} \rfloor \).

First, it is shown that the signature vector decreases lexicographically after one call of \( \text{BUILD} \) and one call of \( \text{COLLAPSE} \), and the coordinates in the signature vector are non-decreasing from left to right [I] Lemma 4.10. When a new layer is added, the vector gains a new second to rightmost coordinate and so the lexicographical order decreases. After collapsing the last layer \( L_t \) in the collapse phase, \( |Y_t| \) drops to \((1-\mu)|Y_t| \) or less. One can then verify that \( s_t \) drops by one or more. So the signature vector decreases lexicographically. After \( \text{COLLAPSE} \), no layer is collapsible. Lemma [I.1] implies immediately that \( s_t \geq s_{t-1} \).

Second, one can verify from the definition that each \( s_i \) is bounded by an integer \( U \) of value at most \( \log |P| \cdot \Omega(\mu^{-3/2} \log \frac{1}{\mu}) \). The number of layers is at most \( \log_{1+\sqrt{\mu}}|P| \leq U \). Thus, the sum of coordinates in any signature vector is at most \( U^2 \). The number of distinct partitions of the integer \( U^2 \) is \( \Omega(|P| \Omega(\mu^{-3/2} \log \frac{1}{\mu})) \), which is an upper bound on the number of distinct signature vectors. This bounds the number of calls to \( \text{BUILD} \) and \( \text{COLLAPSE} \). Details can be found in [I] Lemma 4.11. Since each call of \( \text{BUILD} \) and \( \text{COLLAPSE} \) runs in \( \text{poly}(\ell, m, n) \) time, we conclude that the algorithm runs in \( \text{poly}(m, n) \) time.

The remaining task is to binary search for \( \tau^* \). If we use a value \( \tau \) that is at most \( \tau^* \), the algorithm terminates in polynomial time with an allocation. If we use a value \( \tau > \tau^* \), there are two possible outcomes. We may be lucky and always have some collapsible layer below \( L_{i+1} \) whenever \( |Y_{i+1}| < \sqrt{\mu}|Y_{\leq i}| \) for some \( i \in [1, \ell-1] \). In this case, the algorithm returns in polynomial time an allocation of value at least \( \tau/\lambda \geq \tau^*/\lambda \). The second outcome is that no layer is collapsible at some point, but \( |Y_{i+1}| < \sqrt{\mu}|Y_{\leq i}| \) for some \( i \in [1, \ell-1] \). This can be detected in \( O(1) \) time by maintaining \( |Y_{i+1}| \) and \( |Y_{\leq i}| \), which allows us to detect that \( \tau > \tau^* \) and halt the algorithm. Since this is the first violation of the property that some layer below \( L_{i+1} \) is collapsible if \( |Y_{i+1}| < \sqrt{\mu}|Y_{\leq i}| \) for some \( i \in [1, \ell-1] \), the running time before halting is polynomial in \( m \) and \( n \). The last allocation returned by the algorithm during the binary search has value at least \( \tau^*/\lambda = \tau^*/(6 + \delta) \). We will see in Section 5.2 that a smaller \( \delta \) requires a smaller \( \mu \) and hence a higher running time.

In summary, the initial range for \( \tau \) for the binary search process is \([a, b] \), where \( a \) is initialized to zero, and \( b \) is initialized to be the sum of values of all resources divided by the number of players (rounded down). In each binary search probe (i.e., \( \tau = \lfloor (a + b)/2 \rfloor \)), if the algorithm returns an allocation of value at least \( \tau/\lambda \), we update \( a := \tau+1 \) and recurse on \([a, b] \). Otherwise, we update \( b := \tau-1 \) and recurse on \([a, b] \). When the range becomes empty, we stop. The last allocation returned by the algorithm during the binary search has value at least \( \tau^*/\lambda = \tau^*/(6 + \delta) \). This gives the main theorem of this paper as stated below.

**Theorem 4.2.** For any fixed constant \( \delta \in (0, 1) \), there is an algorithm for the restricted max-min fair allocation problem that returns a \((6 + \delta)\)-approximate solution in time polynomial in the number of players and the number of resources.

5 Analysis

We will develop lower and upper bounds for the total value of the thin resources in the stack and show that if Lemma 4.1 does not hold, the lower bound would exceed the upper bound. To do this, we need a tool to analyze our aggressive greedy strategy for picking blocked addable thin edges. We introduce this tool in section 5.1 and then prove Lemma 4.1 in section 5.2. Recall that we assume \( \tau \leq \tau^* \).
5.1 Competing players

In this section, we will show that there is an injective map \( \varphi \) from players covered by blocked addable thin edges to players who can access thin resources of large total value. We call the image of \( \varphi \) the competing players. The next result shows that the target players can be identified via a special maximum matching with respect to an optimal allocation. Recall that for a maximum matching \( M \) of \( G, \overline{P}_M \) is the set of players not matched by \( M \).

**Lemma 5.1.** Let OPT be an arbitrary optimal allocation, i.e., of value \( \tau^* \). There exists a maximum matching \( M^* \) of \( G \) induced by OPT such that \( M^* \) matches every player who is assigned at least one fat resource in OPT. Hence, every player in \( \overline{P}_{M^*} \) is assigned only thin resources in OPT which are worth a total value of \( \tau^* \) or more.

*Proof.* We construct a maximum matching of \( G \) induced by OPT as follows. OPT induces a matching \( M \) of \( G \) that matches all the players who receive at least one fat resource in OPT. \( M \) may not be a maximum matching though. We augment \( M \) to a maximum matching \( M^* \) using augmenting paths as in basic matching theory [13]. Augmentation ensures that \( M^* \) matches all the players who are matched by \( M \). Hence, \( M^* \) is the desired maximum matching. \( \square \)

Lemma 5.2 below establishes the existence of the injective map \( \varphi \) from \( X_{\leq \ell} \) to \( \overline{P}_{M^*} \), where \( M^* \) is a maximum matching induced by some optimal allocation OPT. Let \( D_\varphi \) and \( \text{Im}_\varphi \) be the domain and image of \( \varphi \), respectively. Consider Lemma 5.2 (i) and (iii). It would be ideal if \( D_\varphi \) covers the entire \( X_{\leq \ell} \). However, for technical reasons, when COLLAPSE removes a player from \( X_{\leq \ell} \), we may have to remove two players from \( D_\varphi \) in order to maintain other properties of \( \varphi \). Lemma 5.2 (iii) puts a lower bound on the size of the domain of \( \varphi \).

As stated in Lemma 5.1, players in \( \text{Im}_\varphi \subseteq \overline{P}_{M^*} \) are assigned at least \( \tau^*|\text{Im}_\varphi| \geq \tau|\text{Im}_\varphi| \) worth of thin resources in total in OPT. We argue that Lemma 5.2 (ii) implies that a large subset of these thin resources are in the stack. Consider the time when a player \( p \) was added to \( X_k \) (and its corresponding addable edge was added to \( X_k \)). The player \( p \) was addable at that time, and so was \( \varphi(p) \). Since the algorithm preferred \( p \) to \( \varphi(p) \), either no addable edge was incident to \( \varphi(p) \) or the maximal addable edge identified for \( \varphi(p) \) had value no more than the maximal addable edge \( e_p \) identified for \( p \). In both cases, at least \( \tau - \text{value}(e_p) \) worth of thin resources assigned to \( \varphi(p) \) in OPT were already in the stack.

To have a good lower bound for the total value of the thin resources in the stack, we also need to look at the players in \( \overline{P}_{M^*} \setminus \text{Im}_\varphi \). Let \( \ell \) be the index of the topmost layer in the stack. Lemma 5.2 (iv) will allow us to prove that roughly \( |Y_{\leq \ell-1} - |X_{\leq \ell} \cup I| \) of players in \( \overline{P}_{M^*} \setminus \text{Im}_\varphi \) are still addable after we finish adding edges to \( X_k \) during the construction of layer \( L_\ell \). However, there are no more addable edges (to be added to \( X_k \)). Therefore, each of these addable players can access no more than \( \tau/\lambda \) worth of thin resources that are not in the stack. In other words, at least \( \tau - \tau/\lambda \) worth of the thin resources that are assigned to each of them in OPT are already in the stack.

**Lemma 5.2.** Let \( M^* \) be a maximum matching of \( G \) induced by some optimal allocation. For any state \( (M, \mathcal{E}, \mathcal{I}, \{L_1, \ldots, L_\ell\}) \) of the algorithm, there exists an injection \( \varphi \) such that:

(i) The domain \( D_\varphi \) and image \( \text{Im}_\varphi \) of \( \varphi \) are subsets of \( X_{\leq \ell} \) and \( \overline{P}_{M^*} \), respectively.

(ii) For every player \( p \in D_\varphi \), when \( p \) was added to \( X_k \) for some \( k \in [1, \ell] \), \( \varphi(p) \) was also an addable player at that time.

(iii) \( |D_\varphi| \geq 2|X_{\leq \ell}| - \sum_{i=1}^\ell z_i \).

(iv) \( f_M(\overline{P}_M, (\overline{P}_{M^*} \setminus \text{Im}_\varphi) \cup X_{\leq \ell}) = |\overline{P}_M| \).

18
Proof. Our proof is by induction on the chronological order of the build and collapse phases. In the base case, \( \ell = 1 \), \( X_1 = \emptyset \), and \( z_1 = 0 \). The existence of \( \varphi \) is trivial as its domain \( D_\varphi \subseteq X_1 = \emptyset \) and image \( \text{Im}_\varphi = \emptyset \). Then, (i), (ii), and (iii) are satisfied trivially. As both \( \text{Im}_\varphi \) and \( X_{\leq 1} \) are empty, the left hand side of (iv) becomes \( f_M(\overline{P}_M, \overline{P}_{M^*}) \) which is equal to \( |\overline{P}_M| \) by Claim 2.3(i). We analyze how to update the injection \( \varphi \) during the build and collapse phases in order to preserve (i)–(iv).

**Build phase.** Suppose that BUILD begins to construct a new layer \( L_\ell \). \( X_\ell \) is initialized to be empty. The value \( z_\ell \) is computed only at the completion of \( L_\ell \). However, in this proof, we initialize \( z_\ell = 0 \), increment \( z_\ell \) whenever we add an edge to \( X_\ell \) (and the corresponding player to \( X_\ell \)), and show the validity of (i)–(iv) inductively. This will then imply the validity of (i)–(iv) at the completion of \( L_\ell \).

Since \( X_\ell = \emptyset \) and \( z_\ell = 0 \) initially, properties (i)–(iv) are satisfied by the current \( \varphi \) by inductive assumption.

Step 2 of BUILD does not change \( X_\ell \), and so \( \varphi \) needs no update.

In step 2 of BUILD, when we add an edge to \( X_\ell \), we need to update \( z_\ell, \varphi \) and \( D_\varphi \). Suppose that a thin edge incident to player \( q_1 \) is added to \( X_\ell \). So \( q_1 \) is an addable player. For clarity, we use \( X_\ell', z_\ell', \varphi', D_{\varphi'} \) and \( \text{Im}_{\varphi'} \) to denote the updated \( X_\ell, z_\ell, \varphi, D_{\varphi}, \) and \( \text{Im}_{\varphi} \), respectively. Clearly, \( X_\ell' = X_\ell \cup \{q_1\} \) and \( z_\ell' = z_\ell + 1 \). We set \( D_{\varphi'} := D_{\varphi} \cup \{q_1\} \). For every player \( p \in D_{\varphi'} \setminus \{q_1\} \), we set \( \varphi'(p) := \varphi(p) \). We determine \( \varphi'(q_1) \) as follows. Recall that for a set \( \Pi \) of non-trivial paths in \( G_M \), we use \( \Pi^+ \) to denote the set of non-trivial paths in \( \Pi \).

Let \( \Pi_1 \) be an optimal solution of \( G_M(Y_{\leq \ell-1}, X_{\leq \ell} \cup I \cup \{q_1\}) \). Player \( q_1 \) must be a sink of \( \Pi_1 \) since otherwise we would have \( f_M(Y_{\leq \ell-1}, X_{\leq \ell} \cup I \cup \{q_1\}) = f_M(Y_{\leq \ell-1}, X_{\leq \ell} \cup I) \), contradicting the addability of \( q_1 \). For the same reason, we have \( q_1 \notin X_{< \ell} \). As \( q_1 \in \text{sink}(\Pi_1) \), \( q_1 \) must be unmatched in the maximum matching \( M^+ \Pi_1 \), i.e., \( q_1 \in \overline{P}_M \setminus \Pi_1^+ \). Let \( \Pi_2 \) be an optimal solution of \( G_{M \oplus \Pi_1^+}((\overline{P}_M \setminus \Pi_1^+) \setminus \text{Im}_\varphi) \cup X_{\leq \ell}) \).

\[
|\Pi_2| = f_{M \oplus \Pi_1^+}((\overline{P}_M \setminus \Pi_1^+) \setminus \text{Im}_\varphi) \cup X_{\leq \ell}) = f_M(\overline{P}_M, (\overline{P}_{M^*} \setminus \text{Im}_\varphi) \cup X_{\leq \ell}) \\
= |\overline{P}_M| \\
= |\overline{P}_M| \quad \text{(by induction assumption)}
\]

It follows that every player in \( \overline{P}_M \setminus \Pi_1^+ \) is a source in \( \Pi_2 \). So there exists a path \( \pi \in \Pi_2 \) that originates from \( q_1 \). Let \( q_2 = \text{sink}(\pi) \).

We claim that \( q_2 \notin X_{< \ell} \). If \( \pi \) is a trivial path, then the claim holds trivially since \( q_2 = q_1 \notin X_{< \ell} \). Suppose that \( \pi \) is a non-trivial path. Assume, for the sake of contradiction, that \( q_2 \in X_{< \ell} \). This allows us to apply Claim 2 to \( G_M(Y_{\leq \ell-1}, X_{\leq \ell} \cup I \cup \{q_1\}) \), its optimal solution \( \Pi_1 \), and the path \( \pi \) (Recall that \( q_1 = \text{sink}(\pi) \in \text{sink}(\Pi_1) \)). By Claim 2.3, we can use \( \pi \) to convert \( \Pi_1 \) to an equal-sized set of node-disjoint paths from \( Y_{< \ell-1} \) to \( X_{< \ell} \cup I \). But then \( f_M(Y_{\leq \ell-1}, X_{< \ell} \cup I) \geq |\Pi_1| = f_M(Y_{\leq \ell-1}, X_{< \ell} \cup I \cup \{q_1\}) \), a contradiction to the addability of \( q_1 \). This proves our claim that \( q_2 \notin X_{< \ell} \).

Observe that \( q_2 \in \overline{P}_{M^*} \setminus \text{Im}_\varphi \) because \( q_2 \in \text{sink}(\Pi_2) \subseteq (\overline{P}_{M^*} \setminus \text{Im}_\varphi) \cup X_{\leq \ell} \) and \( q_2 \notin X_{\leq \ell} \). This allows us to set \( \varphi'(q_1) := q_2 \) and keep \( \varphi' \) injective.

Now we show that properties (i)–(iv) are satisfied by \( \varphi', z_\ell', D_{\varphi'} \), and \( X_\ell' \). Properties (i) and (iii) are straightforwardly satisfied.

By induction assumption, (ii) holds for players in \( D_{\varphi'} \setminus \{q_1\} = D_{\varphi} \). It remains to check the validity of (ii) for \( \varphi'(q_1) = q_2 \). Recall that \( \pi \) is a path from \( q_1 \) to \( q_2 \) in \( G_{M \oplus \Pi_1^+} \). If \( \pi \) is a trivial path, then (ii) holds because \( q_2 = q_1 \) and \( q_1 \) is addable. Assume that \( \pi \) is non-trivial. By Claim 2.3, we can use \( \pi \) to convert \( \Pi_1 \) to an equal-sized set of node-disjoint paths in \( G_M \) from

\[
19
\]
Since property (iv) is affected by replacing $q_1$ at the end of collapsing $Y$, we need to update $\varphi$.

Consider (iv). If $\pi$ is a trivial path, i.e., $q_1 = q_2$, then (iv) holds because 
\[
(\bar{P}_M \setminus \operatorname{Im}_\varphi) \cup X_{<\ell} \subseteq (\bar{P}_M \setminus (\operatorname{Im}_\varphi \cup \{q_2\})) \cup X_{<\ell} \cup \{q_1\} = (\bar{P}_M \setminus \operatorname{Im}_\varphi) \cup X_{<\ell}.
\]

Suppose that $\pi$ is non-trivial. Recall that $\Pi_2$ is an optimal solution of $G_{M\oplus \Pi_1^+}((\bar{P}_M \setminus \operatorname{Im}_\varphi) \cup X_{<\ell})$, and that $|\Pi_2| = |\bar{P}_M \setminus \Pi_1^+|$. Take the maximum matching $M \oplus \Pi_1^+$ of $G$ and flip the paths in $\Pi_2^+ \setminus \{\pi\}$ in $G$. This produces another maximum matching $M' = (M \oplus \Pi_1^+) \oplus (\Pi_2^+ \setminus \{\pi\})$. All $|\bar{P}_M \setminus \Pi_1^+|$ sinks of $\Pi_2$, except for $q_2$, are unmatched in $M'$. Player $q_1$ is also unmatched in $M'$. There are equally many unmatched players in $M'$, and $M \oplus \Pi_1^+$ as both are maximum matchings of $G$. This implies that $(\operatorname{sink}(\Pi_2) \setminus \{q_2\}) \cup \{q_1\}$ is exactly $\bar{P}_M$. Since $\operatorname{sink}(\Pi_2) \subseteq (\bar{P}_M \setminus \operatorname{Im}_\varphi) \cup X_{<\ell}$, we conclude that 
\[
\bar{P}_M' \subseteq ((\bar{P}_M \setminus \operatorname{Im}_\varphi) \cup X_{<\ell}) \cup \{q_1\} \\
\subseteq (\bar{P}_M \setminus (\operatorname{Im}_\varphi \cup \{q_2\})) \cup X_{<\ell} \cup \{q_1\} \\
= (\bar{P}_M \setminus \operatorname{Im}_\varphi) \cup X_{<\ell}.
\]

By the above subset relation and Claim 2.3(i), 
\[
|\bar{P}_M| \geq f_M(\bar{P}_M, (\bar{P}_M \setminus \operatorname{Im}_\varphi) \cup X_{<\ell}) \geq f_M(\bar{P}_M, \bar{P}_M) = |\bar{P}_M|.
\]

Hence, (iv) holds.

Clearly, steps [1],[2] of BUILD do not affect $\varphi$.

**Collapse phase.** Suppose that we are going to collapse the layer $L_t$ at the end of collapsing $L_t$, we only need to prove (i)–(iv) with $\ell$ substituted by $t$.

Clearly, step [1] of COLLAPSE has no effect on $\varphi$.

Consider step [2] of COLLAPSE. Go back to the last time when $L_t$ was either created by BUILD as the topmost layer or made by COLLAPSE as the topmost layer. By the inductive assumption, there was an injection $\varphi''$ at that time that satisfies (i)–(iv). We set $\varphi := \varphi''$, $D_\varphi := D_{\varphi''}$, and $\operatorname{Im}_\varphi := \operatorname{Im}_{\varphi''}$.

In step [3] of COLLAPSE, the maximum matching $M$ may change, so only (iv) is affected. Nonetheless, by Claim 2.3(ii), the value of $f_M(\bar{P}_M, (\bar{P}_M \setminus \operatorname{Im}_\varphi) \cup X_{<\ell})$ remains the same after updating $M$. So (iv) is satisfied afterwards.

In step 4 of COLLAPSE, we may remove some edges from $X_t$ and add some edges to $I$. Adding edges to $I$ does not affect $\varphi$. We need to update $\varphi$ when an edge is removed from $X_t$.

Suppose that we are going to remove from $X_t$ an edge that is incident to a player $q_1$. Let $X'_{<\ell}$, $\varphi'$, $D_\varphi'$, and $\operatorname{Im}_\varphi'$ denote the updated $X_{<\ell}$, $\varphi$, $D_\varphi$, and $\operatorname{Im}_\varphi$, respectively. Note that $X'_{<\ell} = X_{<\ell} \setminus \{q_1\}$. We show how to define $\varphi'$, $D_\varphi'$, and $\operatorname{Im}_\varphi'$ appropriately. Recall that $z_t$ was defined in the last construction of the layer $L_t$ during the build phase, and it has remained fixed despite possible changes to $X_t$ since then.

Consider (iv). If (iv) is not affected by replacing $X_{<\ell}$ with $X'_{<\ell}$, that is, $f_M(\bar{P}_M, (\bar{P}_M \setminus \operatorname{Im}_\varphi) \cup X'_{<\ell}) = |\bar{P}_M|$, then we simply set $D_\varphi := D_\varphi \setminus \{q_1\}$ and $\varphi'(p) := \varphi(p)$ for all $p \in D_\varphi$. Since $\operatorname{Im}_\varphi \subseteq \operatorname{Im}_\varphi'$, $\varphi'$ satisfies (iv), i.e., $f_M(\bar{P}_M, (\bar{P}_M \setminus \operatorname{Im}_\varphi') \cup X'_{<\ell}) = |\bar{P}_M|$. Suppose that property (iv) is affected by replacing $X_{<\ell}$ with $X'_{<\ell}$, and as a consequence, 
\[
f_M(\bar{P}_M, (\bar{P}_M \setminus \operatorname{Im}_\varphi) \cup X'_{<\ell}) = |\bar{P}_M| - 1. \tag{1}
\]

Since $f_M(\bar{P}_M, \bar{P}_M) = |\bar{P}_M|$, we have 
\[
f_M(\bar{P}_M, \bar{P}_M' \cup X'_{<\ell}) = |\bar{P}_M|. \tag{2}
\]
Comparing equations (1) and (2), we conclude that there exists a player \( q_2 \in D_{\varphi} \) such that

\[
 f_M(\overline{P}_M, (\overline{P}_M \setminus (\text{Im}_{\varphi} \setminus \{\varphi(q_2)\})) \cup X_{\leq t}') = |\overline{P}_M|.
\]

We set \( D_{\varphi'} := D_{\varphi} \setminus \{q_1, q_2\} \), and \( \varphi'(p) := \varphi(p) \) for all \( p \in D_{\varphi'} \). Then (iv) is satisfied by \( \varphi' \).

Irrespective of which definition of \( \varphi' \) above is used, properties (i) and (ii) trivially hold. Property (iii) holds because the left hand side decreases by at most 2 and the right hand side decreases by exactly 2.

### 5.2 Proof of Lemma 4.1

Suppose, for the sake of contradiction, that there exists an index \( k \in [1, \ell - 1] \) such that \( |Y_{k+1}| < \sqrt{\mu} |Y_{\leq k}| \) but no layer below \( L_{k+1} \) is collapsible. Without loss of generality, we assume that \( k \) is the smallest such index. Therefore, \( |Y_{i+1}| \geq \sqrt{\mu} |Y_{\leq i}| \) for every \( i \in [1, k-1] \).

Consider the moment immediately after the last construction of the \((k+1)\)-th layer in the build phase. Let \((M', \mathcal{E}', \mathcal{T}', (L'_1, \ldots, L'_{k+1}))\) be the state of the algorithm at that moment, where \( L'_i = (\mathcal{X}'_i, \mathcal{Y}'_i, d_{k+1}, z'_i) \).

We will derive a few inequalities that hold given the existence of \( \mathcal{S} \) of thin edges, \( \text{value} \). Recall that given a set \( \mathcal{S} \) of thin resources, \( \text{value}(\mathcal{S}) \) is the total value of the thin resources covered by \( \mathcal{S} \).

We first define some notation that will be used in the proof. Let \( M^* \) be a maximum matching induced by an optimal allocation \( \text{OPT} \). Let \( \varphi' \) and \( D_{\varphi'} \) be the injection and its domain as defined in Lemma 5.2 with respect to \( M^* \) and the state \((M', \mathcal{E}', \mathcal{T}', (L'_1, \ldots, L'_{k+1}))\). Define

\[
\forall p \in D_{\varphi'}, w_p := \text{value}(B), \text{ where } (p, B) \text{ is the thin edge in } \mathcal{X}'_{\leq k+1} \text{ that is incident to } p.
\]

It is well defined because \( D_{\varphi'} \subseteq \mathcal{X}'_{\leq k+1} \) by Lemma 5.2(i) and no player is incident to two edges in \( \mathcal{X}'_{\leq k+1} \) by invariant 1 in Table 1. Also by invariant 1 in Table 1

\[
\forall p \in D_{\varphi'}, w_p \in [\tau/\lambda, \tau + \tau/\lambda].
\]

Given the above definition, we already have two easy inequalities. Recall that given a set \( \mathcal{S} \) of thin edges, \( \text{value}(\mathcal{S}) \) is the total value of the thin resources covered by \( \mathcal{S} \).

\[
\text{value}(\mathcal{X}'_{\leq k+1} \cup \mathcal{Y}'_{\leq k}) \geq \text{value}(\mathcal{X}'_{\leq k+1}) \geq \sum_{p \in D_{\varphi'}} w_p \geq \frac{\tau}{\lambda} |D_{\varphi'}| \leq \sum_{p \in D_{\varphi'}} w_p \leq \left(\tau + \frac{\tau}{\lambda}\right) |D_{\varphi'}|
\]

We list three more inequalities in the claims below. Their proofs are deferred to Sections 5.3.1, 5.3.2 and 5.3.3, respectively.

**Claim 5.3.** \( |D_{\varphi'}| \leq |Y_{\leq k}| \).

**Claim 5.4.** \( \text{value}(\mathcal{X}'_{\leq k+1} \cup \mathcal{Y}'_{\leq k}) \leq \frac{\tau}{\lambda} |D_{\varphi'}| + \frac{2\tau}{\lambda} |Y_{\leq k}'| + \frac{\delta_1}{\delta_2} |Y_{\leq k}'| \), where \( \delta_1 = \lambda \mu + 2 \mu + 2 \sqrt{\mu} \).

**Claim 5.5.** \( \text{value}(\mathcal{X}'_{\leq k+1} \cup \mathcal{Y}'_{\leq k}) \geq (\tau - \frac{\tau}{\lambda}) (|Y_{\leq k}'| - |D_{\varphi'}|) + \sum_{p \in D_{\varphi'}} (\tau - w_p) - \frac{\delta_1}{\delta_2} |Y_{\leq k}'| \), where \( \delta_2 = 2 \lambda \mu + 2 \lambda \sqrt{\mu} + 6 \sqrt{\mu} \).

Putting all the five inequalities together gives the following system:

\[
\text{value}(\mathcal{X}'_{\leq k+1} \cup \mathcal{Y}'_{\leq k}) \geq \sum_{p \in D_{\varphi'}} w_p,
\]

\[
\frac{\tau}{\lambda} |D_{\varphi'}| \leq \sum_{p \in D_{\varphi'}} w_p \leq (\tau + \tau/\lambda) |D_{\varphi'}|,
\]

21
Recall a few things. This is impossible because (3) and (4) contradict each other. Hence, Lemma 4.1 is true.

$$|D_{\varphi'}| \leq |Y'_{\leq k}|,$$

$$\text{value}(\mathcal{A}'_{\leq k+1} \cup \mathcal{Y}'_{\leq k}) \leq \frac{\tau}{\lambda}|D_{\varphi'}| + \frac{2\tau}{\lambda}|Y'_{\leq k}| + \frac{\delta_1 \tau}{\lambda}|Y'_{\leq k}|,$$

$$\text{value}(\mathcal{A}'_{\leq k+1} \cup \mathcal{Y}'_{\leq k}) \geq (\tau - \tau/\lambda)(|Y'_{\leq k}| - |D_{\varphi'}|) + \sum_{p \in D_{\varphi'}} (\tau - w_p) - \frac{\delta_2 \tau}{\lambda}|Y'_{\leq k}|.$$

Divide the above system by $\frac{\tau}{\lambda}|D_{\varphi'}|$. To simplify the system, define the variables $B_1 := \text{value}(\mathcal{A}'_{\leq k+1} \cup \mathcal{Y}'_{\leq k})/(\frac{\tau}{\lambda}|D_{\varphi'}|)$, $B_2 := |Y'_{\leq k}|/|D_{\varphi'}|$, and $B_3 := (\sum_{p \in D_{\varphi'}} w_p)/(\frac{\tau}{\lambda}|D_{\varphi'}|)$. Then we can write the above system equivalently as follows:

$$B_1 \geq B_3,$$

$$1 \leq B_3 \leq \lambda + 1,$$

$$1 \leq B_2,$$

$$B_1 \geq 1 + 2B_2 + \delta_1 B_2,$$

$$B_1 \geq (\lambda - 1)(B_2 - 1) + \lambda - B_3 - \delta_2 B_2.$$

The first, fourth, and fifth inequalities give

$$2(1 + 2B_2 + \delta_1 B_2) \geq B_1 + B_3 \geq (\lambda - 1)(B_2 - 1) + \lambda - \delta_2 B_2$$

$$\Rightarrow 2 + (4 + 2\delta_1)B_2 \geq (\lambda - 1 - \delta_2)B_2 + 1$$

$$\Rightarrow (\lambda - 5 - 2\delta_1 - \delta_2)B_2 \leq 1 \quad (3)$$

When $\mu$ tends to zero, both $\delta_1$ and $\delta_2$ tend to 0. Given that $\lambda = 6 + \delta$ for an arbitrary constant $\delta \in (0, 1)$, when $\mu$ is sufficiently small, $\lambda - 5 - 2\delta_1 - \delta_2 > 1$. Since $B_2 \geq 1$ by the third inequality, we obtain

$$\lambda - 5 - 2\delta_1 - \delta_2 B_2 > 1. \quad (4)$$

This is impossible because (3) and (4) contradict each other. Hence, Lemma 4.1 is true.

### 5.3 Proofs of Claims 5.3-5.5

Recall a few things. $(M, E, I, (L_1, \ldots, L_i))$ is the current state of the algorithm. The index $k$ is the smallest index such that $|Y'_{k+1}| < \sqrt{\mu}|Y'_{\leq k}|$ and no layer below $L_{k+1}$ is collapsible. Hence, $|Y'_{k+1}| \geq \sqrt{\mu}|Y'_{\leq k}|$ for every $i \in [1, k]$. $(M', E', I', (L'_1, \cdots, L'_{k+1}))$ is the state of the algorithm immediately after the last construction of the $(k+1)$-th layer in the build phase. $\varphi'$, $D_{\varphi'}$ and $\text{Im}_{\varphi'}$ are the injection, its domain, and its image defined in Lemma 5.2 with respect to a maximum matching $M^*$ induced by an optimal allocation OPT and the state $(M', E', I', (L'_1, \cdots, L'_{k+1}))$.

Before proving Claims 5.3-5.5, we derive two more claims.

**Claim 5.6.** For $i \in [1, k]$, $L'_i = L_i$. That is, none of layers $L'_1, \ldots, L'_k$ have ever been collapsed since the last construction of the $(k+1)$-th layer in the build phase. For layers $L'_{k+1} = (\mathcal{X}'_{k+1}, \mathcal{Y}'_{k+1}, d'_{k+1}, z'_{k+1})$ and $L_{k+1} = (\mathcal{X}_{k+1}, \mathcal{Y}_{k+1}, d_{k+1}, z_{k+1})$, we have $\mathcal{X}_{k+1} \subseteq \mathcal{X}'_{k+1}$, $\mathcal{Y}_{k+1} \subseteq \mathcal{Y}'_{k+1}$, $d_{k+1} = d'_{k+1}$, and $z_{k+1} = z'_{k+1}$.

If some of $L'_1, \ldots, L'_k$ has ever been collapsed, then $L'_{k+1}$ would be removed by Collapsed and it could not be the last $(k+1)$-th layer constructed so far. As to layer $L'_{k+1}$, since its construction so far, it has never been destructed, but it may have been shrunk by Collapsed, and the resulting layer is $L_{k+1}$.

**Claim 5.7.** $|D_{\varphi'}| > |X'_{\leq k+1}| - \mu|Y'_{\leq k}|$.

By Claim 5.6, none of layers $L'_1, \ldots, L'_k$ is collapsible with respect to $(M', E', I', (L'_1, \cdots, L'_{k+1}))$. By Lemma 5.2(ii), $|X'_{\leq k+1}| > \sum_{i=1}^{k+1} z_i - \mu|Y'_{\leq k}|$. By Lemma 5.2(iii), $|D_{\varphi'}| \geq 2|X'_{\leq k+1} - \sum_{i=1}^{k+1} z_i$.

Combining the two inequalities proves Claim 5.7.
5.3.1 Proof of Claim 5.3

By invariant 7 in Table 1, \(|Y_{\leq k}'| \geq f_M(Y'_{\leq k}, X'_{\leq k} \cup I') \geq d_{k+1} - \sum_{i=1}^{k+1} z_{k+1} \geq |X'_{\leq k+1}|\) which is at least \(|D_{\varphi'}|\) as \(D_{\varphi'} \subseteq X'_{\leq k+1}\). Putting the above together shows that \(|D_{\varphi'}| \leq |Y'_{\leq k}|\) as stated in Claim 5.3.

5.3.2 Proof of Claim 5.4

We derive upper bounds for \(\text{value}(X'_{\leq k} \cup Y'_{\leq k})\) and \(\text{value}(X'_{k+1})\) separately. Combining them proves the claim.

Every edge \(e \in Y'_{\leq k}\) belongs to some partial allocation. By the definition of a partial allocation, every thin edge in it is minimal, i.e., taking away a thin resource from \(e\) puts its value below \(\tau/\lambda\), which implies that \(\text{value}(e) \leq 2\tau/\lambda\). Every edge in \(X'_{\leq k}\) is blocked, so it has less than \(\tau/\lambda\) worth of resources that are not covered by edges in \(Y'_{\leq k}\). We conclude that

\[
\text{value}(X'_{\leq k} \cup Y'_{\leq k}) \leq \frac{\tau}{\lambda} |X'_{\leq k}| + \frac{2\tau}{\lambda} |Y'_{\leq k}|. 
\]

(5)

We bound \(\text{value}(X'_{k+1})\) next. By Claim 5.6,

\[
|X'_{k+1} \setminus X_{k+1}| = |X'_{\leq k+1}| - |X_{\leq k+1}|
\leq \sum_{i=1}^{k+1} z_i - |X_{\leq k+1}|
= \mu|Y_{\leq k}|
= \mu|Y'_{\leq k}|
\]

Every edge in \(X'_{k+1}\) has value at most \(\tau + \tau/\lambda\) by invariant 1 in Table 1. Therefore,

\[
\text{value}(X'_{k+1}) \leq \text{value}(X_{k+1}) + (\tau + \tau/\lambda)(|X'_{k+1} \setminus X_{k+1}|)
\leq \text{value}(X_{k+1}) + \mu(\tau + \tau/\lambda)|Y'_{\leq k}|.
\]

(6)

A reasoning analogous to that behind (5) gives

\[
\text{value}(X_{k+1}) \leq \frac{\tau}{\lambda} |X_{k+1}| + \frac{2\tau}{\lambda} |Y_{k+1}|
\leq \frac{\tau}{\lambda} |X_{k+1}| + \frac{2\sqrt{\mu}}{\lambda} |Y_{\leq k}|
= \frac{\tau}{\lambda} |X_{k+1}| + \frac{2\sqrt{\mu}}{\lambda} |Y'_{\leq k}|.
\]

(7)

Combining inequalities (6) and (7) yields

\[
\text{value}(X'_{k+1}) \leq \frac{\tau}{\lambda} |X'_{k+1}| + \frac{2\sqrt{\mu}}{\lambda} |Y'_{\leq k}| + \mu(\tau + \tau/\lambda)|Y'_{\leq k}|
\leq \frac{\tau}{\lambda} |X'_{k+1}| + (\lambda \mu + \mu + 2\sqrt{\mu}) \frac{\tau}{\lambda} |Y'_{\leq k}|.
\]

(8)

The last inequality is due to \(X_{k+1} \subseteq X'_{k+1}\).

Putting (5) and (8) together, we obtain

\[
\text{value}(X'_{\leq k+1} \cup Y'_{\leq k}) \leq \frac{\tau}{\lambda} |X'_{\leq k+1}| + \frac{2\tau}{\lambda} |Y'_{\leq k}| + (\lambda \mu + \mu + 2\sqrt{\mu}) \frac{\tau}{\lambda} |Y'_{\leq k}|.
\]
By Claim 5.7, $|D_{\varphi'}| > |X'_{<k+1}| - \mu|Y'_{<k}|$. Substituting it into the above inequality gives the following inequality as stated in Claim 5.4

$$\text{value}(X'_{<k+1} \cup Y'_{<k}) < \frac{\tau}{\lambda}|D_{\varphi'}| + \frac{2\tau}{\lambda}|Y'_{<k}| + (\lambda\mu + 2\mu + 2\sqrt{\mu})\frac{\tau}{\lambda}|Y'_{<k}|.$$ 

### 5.3.3 Proof of Claim 5.5

Recall the things stated at the beginning of Section 5.

For each player $p$, let $A_p$ denote the set of resources assigned to $p$ in OPT. By Lemma 5.1 for any $p \in \overline{P}_{M'}$, $A_p$ consists of thin resources only and $\text{value}(A_p) \geq \tau$. Since OPT is an allocation, $A_p \cap A_q = \emptyset$ for any distinct players $p$ and $q$. For any subset $S$ of players, we define $A_S := \bigcup_{p \in S} A_p$.

Let $B$ denote the set of resources covered by the edges in $X'_{<k+1} \cup Y'_{<k}$. To derive a lower bound of $\text{value}(B)$, it suffices to focus on the value of a subset of $B$. In particular, we are interested in resources in $B$ that are allocated to $\overline{P}_{M'}$ in OPT, i.e., $B \cap A_{\overline{P}_{M'}}$. $B \cap A_{\overline{P}_{M'}}$ can be divided into two disjoint subsets: those in $A_{\overline{M}\varphi'}$ and those in $A_{\overline{P}_{M'} \setminus \overline{M}\varphi'}$. We consider them separately in the following analysis.

**Resources in $B \cap A_{\overline{P}_{M'} \setminus \overline{M}\varphi'}$:** Lemma 5.2(iv) implies that

$$f_{\overline{M}'}(\overline{P}_{M'} \setminus \overline{M}\varphi') \cup X'_{<k+1} \cup I') = |\overline{P}_{M'}|.$$ 

We have $Y'_{i} \subseteq S'$ for $i \in [2, k]$ by invariant 4 in Table 1 and $Y_1 = \{(p_0, 0)\}$ where $p_0$ is the player we want to satisfy. Therefore, the players in $Y'_{<k}$ are not matched by $M'$, i.e., $Y'_{<k} \subseteq \overline{P}_{M'}$. It follows from (9) that

$$f_{\overline{M}'}(Y'_{<k}, (\overline{P}_{M'} \setminus \overline{M}\varphi') \cup X'_{<k+1} \cup I') = |Y'_{<k}|.$$ 

We also have

$$f_{\overline{M}'}(Y'_{<k}, X'_{<k+1} \cup I') \leq |X'_{<k+1}| + |I'|.$$ 

Comparing (10) and (11), we conclude that, during the construction of $L'_{k+1}$, after we finish adding edges to $I'$ and $X'_{<k+1}$ in steps 2 and 3 of BUILD, at least $|Y'_{<k}| - |X'_{<k+1}| - |I'|$ players in $\overline{P}_{M'} \setminus \overline{M}\varphi'$ are still addable. Let $S$ denote this subset of addable players in $\overline{P}_{M'} \setminus \overline{M}\varphi'$. Still, no more edge is added to $X'_{<k+1} \cup I'$ in steps 2 and 3 of BUILD. The reason must be that the players in $S$ do not have addable edges. In other words, for each player $p \in S$, since $\text{value}(A_p) \geq \tau$, at least $\tau - \tau/\lambda$ worth of the thin resources in $A_p$ must already be covered by $X'_{<k+1} \cup Y'_{<k} \cup I'$. Therefore, the value of resources in $A_S$ that are covered by $X'_{<k+1} \cup Y'_{<k} \cup I'$ is at least

$$(\tau - \tau/\lambda)(|Y'_{<k}| - |X'_{<k+1}| - |I'|).$$

Subtracting the contribution of $I'$, we obtain

$$\text{value}(B \cap A_{\overline{P}_{M'} \setminus \overline{M}\varphi'}) \geq \text{value}(B \cap A_S)$$

$$\geq (\tau - \tau/\lambda)(|Y'_{<k}| - |X'_{<k+1}| - |I'|) - \text{value}(I').$$ 

By Claim 5.6, $L_i'$ is not collapsible for $i \in [1, k]$. Then by Lemma 3.6(i), $|I'| < \mu|Y'_{<k}|$. By invariant 1 in Table 1, edges in $I'$ have values at most $2\tau/\lambda$, so $\text{value}(I') \leq \frac{2\tau}{\lambda}|I'|$. This allow us to modify (12) and get

$$\text{value}(B \cap A_{\overline{P}_{M'} \setminus \overline{M}\varphi'}) \geq (\tau - \tau/\lambda)(|Y'_{<k}| - |X'_{<k+1}| - (\tau + \tau/\lambda)|I'|)$$

$$> (\tau - \tau/\lambda)(|Y'_{<k}| - |X'_{<k+1}|) - (\tau + \tau/\lambda)|Y'_{<k}|.$$ 

By Claim 5.7, $|D_{\varphi'}| > |X'_{<k+1}| - \mu|Y'_{<k}|$. Substituting this inequality into (13) gives the final inequality that we want in this case.

$$\text{value}(B \cap A_{\overline{P}_{M'} \setminus \overline{M}\varphi'}) > (\tau - \tau/\lambda)(|Y'_{<k}| - |D_{\varphi'}|) - 2\mu\tau|Y'_{<k}|.$$ 

(14)
Resources in $B \cap A_{\text{Im}e}$. As $D_{\varphi} \subseteq X'_{c,k+1}$ by Lemma 5.2(i), every player in $D_{\varphi}$ belongs to $X'_j$ for some $j \in [1,k+1]$. For every $j \in [1,k]$, let $t_j$ denote the time immediately after the last construction of the $j$-th layer in the building phase prior to the creation of $L_{k+1}$. Similarly let $t_{k+1}$ be the time immediately after the construction of $L_{k+1}$. One can see that $t_1 < t_2 < \cdots < t_{k+1}$. For each $t_j$ and $c \in [1,j]$, we use $X^c_j$ and $Y^c_j$ to denote the set of addable edges and the set of blocking edges in the $c$-th layer at time $t_j$. Similarly, we define $T^c_j$ to be the set of unblocked addable edges at time $t_j$.

For reasons similar to those underlying Claim 5.6 from time $t_j$ to time $t_{k+1}$, none of the layers below the $j$-th layer have ever been collapsed. As a result, we have $X^c_j = X'_c$ and $Y^c_j = Y'_c$ for $c \in [1,j-1]$. The $j$-th layer may have been collapsed several times, so $X''_j \subseteq X^c_j$. We also have $z'_j = |X''_j|$ because the $z'_j$ is unchanged from time $t_j$ to time $t_{k+1}$.

For every $q \in D_{\varphi}$, let $e_q$ denote the edge in $X'_{c,k+1}$ that is incident $q$. Recall that $w_q = \text{value}(e_q)$ by definition. Let $q^* = \varphi'(q)$.

**Claim 5.8.** Suppose that $q \in D_{\varphi} \cap X'_j$ for some $j \in [2,k+1]$. The total value of resources shared by $X^c_{j\leq j} \cup Y^c_{j\leq j-1} \cup T^c_j$ and $A_{\varphi}$ is at least $\tau - w_q$.

**Proof.** If $w_q \geq \tau$, the correctness is trivial. Assume that $w_q < \tau$. Recall that $e_q \in X'_j \subseteq X^c_j$. Consider the moment when $q$ is picked by the algorithm and $e_q$ is added to $X^c_j$. By Lemma 5.2(ii), $q^*$ is also an addable player. Since we do not choose $q^*$, either $q^*$ is not incident to any addable edge or the maximal blocked addable thin edge $e_{q^*}$ identified for $q^*$ has value no greater than $\text{value}(e_q)$. In the former case, $A_{\varphi}$ has at most $\tau/\lambda$ worth of its resources not covered by $X^c_{j\leq j} \cup Y^c_{j\leq j-1} \cup T^c_j$. Since $\text{value}(A_{\varphi^*}) \geq \tau$ by Lemma 5.1, we conclude that more than $\tau - \tau/\lambda$ worth of thin resources in $A_{\varphi}$ are already covered by $X^c_{j\leq j} \cup Y^c_{j\leq j-1} \cup T^c_j$. Note that $\tau - \tau/\lambda \geq \tau - w_q$ as $w_q = \text{value}(e_q) \geq \tau/\lambda$ by invariant 1 in Table 1. In the latter case, it must be that $\text{value}(e_{q^*}) \leq \text{value}(e_q) = w_q < \tau$. Recall that a maximal blocked addable edge has value in $[\tau/\lambda, \tau + \tau/\lambda]$. Therefore, given that $\text{value}(e_{q^*}) < \tau$ and yet $e_{q^*}$ is maximal, $e_{q^*}$ must contain all the thin resources desired by $q^*$ but not yet covered by $X^c_{j\leq j} \cup Y^c_{j\leq j-1} \cup T^c_j$. Therefore, $A_{\varphi}$ has at least $\tau - \text{value}(e_{q^*}) \geq \tau - w_q$ worth of thin resources covered by $X^c_{j\leq j} \cup Y^c_{j\leq j-1} \cup T^c_j$. 

The total value of resources shared by $X^c_{j\leq j} \cup Y^c_{j\leq j-1}$ and $A_{\varphi}(D_{\varphi} \cap X'_j)$ is at least the total value of those shared by $X^c_{j\leq j} \cup Y^c_{j\leq j-1} \cup T^c_j$ and $A_{\varphi}(D_{\varphi} \cap X'_j)$ minus $\text{value}(T^c_j)$. Since none of the layers below the $j$-th layer is collapsible, by Lemma 3.6(i), $|T^c_j| \leq \mu|Y^c_{j\leq j-1}|$. By invariant 1 in Table 1, edges in $T^c_j$ have values at most $2\tau/\lambda$. Therefore, by applying Claim 5.8 to all players in $D_{\varphi} \cap X'_j$, the total value of resources shared by $X^c_{j\leq j} \cup Y^c_{j\leq j-1}$ and $A_{\varphi}(D_{\varphi} \cap X'_j)$ is at least

$$\sum_{q \in D_{\varphi} \cap X'_j} (\tau - w_q) - \text{value}(T^c_j) \geq \sum_{q \in D_{\varphi} \cap X'_j} (\tau - w_q) - \frac{2\mu\tau}{\lambda}|Y^c_{j\leq j-1}|.$$ 

Comparing $X^c_{j\leq j} \cup Y^c_{j\leq j-1}$ and $X^c_{j\leq j} \cup Y^c_{j\leq j-1}$, the only difference is $X^c_j \setminus X'_j$ because $X^c_j = X'_c$, $Y^c_j = Y'_c$ for all $c \in [1,j-1]$, and $X'_j \subseteq X^c_j$. Since none of the layers below the $j$-th layer is collapsible, Lemmas 3.5 and 3.6 imply that $|X^c_j| - |X'_j| = z'_j - |X'_j| \leq \sum_{i=1} z'_i - |X'_j| \leq \mu|Y^c_{j\leq j-1}|$. Also each blocked addable edge has value at most $\tau + \tau/\lambda$. Therefore, $\text{value}(X^c_j \setminus X'_j) \leq (\tau + \tau/\lambda)\mu|Y^c_{j\leq j-1}|$. 

25
As a result, the total value of resources shared by $X_{j+1}' \cup \mathcal{Y}_{j+1}'$ and $\mathcal{A}_{i'}(D_{i'} \cap X'_j)$ is at least
\[
\sum_{q \in D_{i'} \cap X'_j} (\tau - w_q) - \frac{2\mu\tau}{\lambda} |Y_{j+1}'| - (\tau + \tau/\lambda)\mu |Y_{j+1}'| = \sum_{q \in D_{i'} \cap X'_j} (\tau - w_q) - (\tau + 3\tau/\lambda)\mu |Y_{j+1}'|.
\]

Recall that $X'_1 = \emptyset$ by the initialization of the building phase. Summing the right hand side of the above inequality over all $j \in [2, k+1]$, we conclude that the total value of resources shared by $X_{k+1}' \cup \mathcal{Y}_{k+1}'$ and $\mathcal{A}_{i'm}$ is at least
\[
\sum_{q \in D_{i'm}} (\tau - w_q) - (\tau + 3\tau/\lambda)\mu \sum_{j=2}^{k+1} |Y_{j-1}'|.
\]

By the definition of $k$, for all $j \in [2, k]$, $|Y'_j| \geq \sqrt{\mu} |Y_{j-1}'|$. By invariant 4 in Table 1, $\sum_{j=2}^{k} |Y'_j| \leq |Y_{k}'|$. Therefore,
\[
\sum_{j=2}^{k+1} |Y_{j-1}'| \leq \frac{1}{\sqrt{\mu}} \sum_{j=2}^{k} |Y'_j| + |Y_{k}'| \leq \left(\frac{1}{\sqrt{\mu}} + 1\right) |Y_{k}'| \leq \frac{2}{\sqrt{\mu}} |Y_{k}'|.
\]

Recall that $B$ is the set of resources covered by edges in $X_{k+1}' \cup \mathcal{Y}_{k}'$. Substituting (16) into (15) gives
\[
\text{value}(\mathcal{B} \cap A_{i'm}) \geq \sum_{q \in D_{i'm}} (\tau - w_q) - (2\tau + 6\tau/\lambda)\sqrt{\mu} |Y_{k}'|.
\]

Combining (17) and (14) completes the proof:
\[
\text{value}(X_{k+1}' \cup \mathcal{Y}_{k}') \geq \text{value}(\mathcal{B} \cap A_{F_{i'm} \setminus A_{i'm}}) + \text{value}(\mathcal{B} \cap A_{i'm})
\geq (\tau - \tau/\lambda)(|Y_{k}'| - |D_{i'm}|) + \sum_{q \in D_{i'm}} (\tau - w_q) - (2\lambda\mu + 2\lambda\sqrt{\mu} + 6\sqrt{\mu})\frac{\tau}{\lambda} |Y_{k}'|.
\]

6 Conclusion

We show that for any constant $\delta \in (0, 1)$, a $(6 + \delta)$-approximate solution can be computed for the restricted fair allocation problem in polynomial time. There is still a gap between the current best estimation ratio and the approximation ratio $6 + \delta$ achieved by this paper. Whether the approximation ratio can match the estimation ratio remains an open problem.

The problem can be generalized slightly to the RAM model where the values of resources are non-integers. Only the binary search scheme needs to be adjusted. When we work on an interval $[a, b]$ for $\tau$, we will recurse on either $[a, (a + b)/2]$ or $[(a + b)/2, b]$ depending on the outcome of solving the reduced problem. We terminate the binary search when $b \leq (1 + \delta)a$. This gives an approximation ratio of $6 + \delta'$ for a constant $\delta' \in (0, 1)$ depending on $\delta$.

References

[1] C. Annamalai, C. Kalaitzis, and O. Svensson. Combinatorial algorithm for restricted max-min fair allocation. ACM Transactions on Algorithms, 13(3):37:1–37:28, 2017.

[2] A. Asadpour, U. Feige, and A. Saberi. Santa Claus meets hypergraph matchings. ACM Transactions on Algorithms, 8(3):24:1–24:9, 2012.
[3] A. Asadpour and A. Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. In Proceedings of the 39th ACM Symposium on Theory of Computing, pages 114–121, 2007.

[4] N. Bansal and M. Sviridenko. The Santa Claus problem. In Proceedings of the 38th ACM Symposium on Theory of Computing, pages 31–40, 2006.

[5] I. Bezáková and Varsha Dani. Allocating indivisible goods. SIGecom Exchanges, 5(3):11–18, 2005.

[6] D. Chakrabarty, J. Chuzhoy, and S. Khanna. On allocating goods to maximize fairness. In Proceedings of the 50th IEEE Symposium on Foundations of Computer Science, pages 107–116, 2009.

[7] T.-H.H. Chan, Z.G. Tang, and X. Wu. On (1, ε)-restricted max-min fair allocation problem. In Proceedings of the 27th International Symposium on Algorithms and Computation, pages 23:1–23:13, 2016.

[8] S.-W. Cheng and Y. Mao. Integrality gap of the configuration LP for the restricted max-min fair allocation. CoRR, abs/1807.04152, 2018.

[9] U. Feige. On allocations that maximize fairness. In Proceedings of the 19th ACM-SIAM Symposium on Discrete Algorithms, pages 287–293, 2008.

[10] D. Golovin. Max-min fair allocation of indivisible good. Technical report, Carnegie Mellon University, 2005.

[11] B. Haeupler, B. Saha, and A. Srinivasan. New constructive aspects of the Lovász local lemma. Journal of the ACM, 58(6):28:1–28:28, 2011.

[12] P.E. Haxell. A condition for matchability in hypergraphs. Graphs and Combinatorics, 11(3):245–248, 1995.

[13] J.E. Hopcroft and R.M. Karp. An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. SIAM Journal on Computing, 2:225–231, 1973.

[14] K. Jansen and L. Rohwedder. A note on the integrality gap of the configuration LP for restricted santa claus. CoRR, abs/1807.03626, 2018.

[15] J.K. Lenstra, D.B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. In Proceedings of the 28th IEEE Symposium on Foundations of Computer Science, pages 217–224, 1987.

[16] L. Lovász and M. D. Plummer. Matching Theory. American mathematical society, 2009.

[17] B. Saha and A. Srinivasan. A new approximation technique for resource-allocation problems. In Proceedings of the 1st Symposium on Innovations in Computer Science, pages 342–357, 2010.