Restricted Max-Min Fair Allocation

Siu-Wing Cheng

Department of Computer Science and Engineering, HKUST, Hong Kong
scheng@cse.ust.hk
https://orcid.org/0000-0002-3557-9935

Yuchen Mao

Department of Computer Science and Engineering, HKUST, Hong Kong
ymaoad@cse.ust.hk
https://orcid.org/0000-0002-1075-344X

⎯⎯ 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: $4 + \delta$ for estimation and $6 + 2\sqrt{10} + \delta \approx 12.325 + \delta$ for construction, where δ is an arbitrarily small constant greater than 0. We propose an algorithm that constructs an allocation with value within a factor $6 + \delta$ from the optimum for any constant $\delta > 0$. The running time is polynomial in the input size for any constant δ 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. The *value of an allocation* is $\min_{p \in P} \sum_{r \in C_p} v_{pr}$. So we want to find an allocation with maximum value.

No algorithm can achieve an approximation ratio less than 2 unless P = NP [5]. Bansal and Sviridenko [4] proposed the configuration LP and showed that it can be solve to any desired accuracy in polynomial time. The configuration LP turns out to be a useful tool for this problem. Using it, approximation ratios of $O(\sqrt{m \log m})$ and $O(n^\delta \log n)$ for any $\delta \geqslant \frac{9 \log \log n}{\log n}$ have been attained [3, 4, 6, 12]. In this paper, we focus on the *restricted case* in which each resource r 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] designed a $O\left(\frac{\log \log m}{\log \log \log m}\right)$-approximation algorithm which is based on rounding the configuration LP. Feige [8] proved that the integrality gap of the configuration LP is bounded by a constant
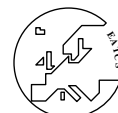
(large and unspecified). His proof was made constructive by Haeupler et al. [10], and hence, results in a constant-approximation algorithm. Asadpour et al. [2] proved that the integrality gap of the configuration LP is at most 4. As a consequence, by solving the configuration LP approximately, one can estimate the optimal solution value within a factor of $4 + \delta$ for any constant $\delta > 0$. However, it is not known how to construct a $(4 + \delta)$-approximate allocation in polynomial time. Annamalai et al. [1] developed a $(6 + 2\sqrt{10} + \delta)$-approximation algorithm. Their algorithm is purely combinatorial, but the analysis still relies on the configuration LP. There is quite a gap between the current best estimation ratio $4 + \delta$ and the current best approximation ratio $6 + 2\sqrt{10} + \delta \approx 12.325 + \delta$.

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 [9]. Chan et al. [7] showed that it is still NP-hard to obtain an approximation ratio less than 2 and that the algorithm in [1] achieves an approximation ratio of 9 in this case. The analysis in [7] 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 δ chosen. Our algorithm uses the same framework as [1]: 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.

Our first contribution is a greedy strategy that is much more aggressive than that in [1]. Let τ^* be the optimal solution value. Let $\lambda > 2$ be the target approximation ratio. To obtain a λ-approximate solution, the value of resources a player need is τ^*/λ. The greedy strategy in [1] 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, both [1] and we add greedy players and the resources claimed by them to the stack. Intuitively, our more aggressive definition of greedy leads to faster growth of the stack, and hence a significantly smaller approximation ratio can be achieved.

Our aggressive strategy brings challenge to the analysis that approaches in [1, 7] 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 in [7], we improve the approximation ratio to $6 + \delta$. Our analysis does not rely on the configuration LP, and is purely combinatorial.

2   Preliminaries

Let τ^* be the optimal solution value. Let λ denote our target approximation ratio. Given any value $\tau \leqslant \tau^*$, our algorithm returns an allocation of value τ/λ in polynomial time. We will show how to combine this algorithm with binary search to obtain an allocation of value at least τ^*/λ in the end. We assume that τ is no more than τ^* in the rest of this section.

Bipartite graph and thin edges.   A resource r is *fat* if $v_r \geqslant \tau/\lambda$; otherwise, r is *thin*. Let G be the bipartite graph formed by representing the players in P and the fat resources in R as vertices, and connecting a player p and a fat resource r by an edge if p desires r. Similarly,

players and thin resources form a hypergraph, namely, there are vertices representing players in P and thin resources in R, and a player p and a subset B of thin resources form an edge (p, B) if p desires all resources in B. (Note that $(p, \emptyset)$ is included.) We call the edges of this hypergraph *thin edges*. For a subset B of thin resources, we define the value of B as $value(B) = \sum_{r \in B} v_r$. For a thin edge $e = (p, B)$, its value $value(e)$ is defined to be the total value of thin resources covered by it, i.e., $value(e) = value(B)$. We use uppercase calligraphic letters to denote subsets of thin edges. Given a set $\mathcal{S}$ of thin edges, define $value(\mathcal{S})$ to be the total value of the thin resources covered by $\mathcal{S}$.

Partial allocation.   Since our target approximation ratio is λ, it suffices to assign each player p either a single fat resource r such that $\{p, r\}$ is an edge of G, or a subset B of thin resources such that (p, B) is a thin edge and $value(B) \geqslant \tau/\lambda$. Hence, it suffices to consider allocations that consist of two parts, one being a maximum matching M of G and the other being a subset $\mathcal{E}$ of thin edges such that every player is covered by either M or $\mathcal{E}$, no two edges in $\mathcal{E}$ share any resource, and every edge in $\mathcal{E}$ has value at least τ/λ.

Our algorithm will start with an arbitrary maximum matching of G alone, grow and update the set $\mathcal{E}$ of thin edges, and whenever necessary, update the maximum matching as well. We call the intermediate solutions *partial allocations*. A **partial allocation** consists of a maximum matching M of G and a set $\mathcal{E}$ of thin edges such that: (i) no player is covered by both M and $\mathcal{E}$; (ii) no two edges in $\mathcal{E}$ share any resource; (iii) every edge $(p, B) \in \mathcal{E}$ is minimal in the sense that $value(B) \geqslant \tau/\lambda$ and every proper subset $B' \subset B$ has value less than τ/λ. We say a player p is *satisfied* by a partial allocation if p is covered by M or $\mathcal{E}$. A partial allocation is an allocation if it satisfies every player.

Node-disjoint paths.   We define a family of networks which are heavily used in both our algorithm and its analysis. With respect to any arbitrary maximum matching M of G, define G_M to be a directed bipartite graph such that G_M has the same vertex set as G (i.e., players and fat resources), there is a directed edge from player p to resource r if $\{p, r\}$ is an edge of G that is not used in M, and there is a directed edge from resource r to player p if $\{p, r\}$ is an edge of G that is used in M.

We use P_M and $\bar{P}_M$ to denote the subsets of players matched and unmatched in M, respectively. Given $S \subseteq \bar{P}_M$ 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 paths. Let $f_M(S, T)$ denote the size of an optimal solution of $G_M(S, T)$. In case that $S \cap T \neq \emptyset$, a feasible solution may allow 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*. Other paths are *non-trivial*.

Let Π be any feasible solution of $G_M(S, T)$. The paths in Π 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 $source(\Pi)$ and $sink(\Pi)$, respectively. A trivial path has only one node which is both its source and sink. We use Π^0 and Π^+ to denote the sets of the trivial paths and the non-trivial paths in Π, respectively.

An optimal solution of $G_M(S, T)$ can be found by solving a maximum s-t flow problem as follows. Add a super source s and directed edges from s to all vertices in S. Add a super sink t and directed edges from all vertices in T to t. Set the capacities of all edges to 1. Find an integral maximum flow in the resulting network. The paths in G_M used by this maximum flow is an optimal solution of $G_M(S, T)$. Node-disjointness is ensured because, in the s-t flow network, each player has in-degree at most one and each resource has out-degree at most one.

Let π be a non-trivial path from $\bar{P}_M$ to P in G_M. If we ignore the directions of edges in π, then π is called an *alternating path* in the matching literature [11]. We use $M \oplus \pi$ to denote the result of flipping π, 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 also a maximum matching of G. We can extend the above operation and form $M \oplus \Pi^+$ for any feasible solution Π of $G_M(S, T)$ for any $S \subseteq \bar{P}_M$ and any $T \subseteq P$. $M \oplus \Pi^+$ is a maximum matching of G. The following results follow from basic theories of matching and network flow.

▶ **Claim 2.1.** *For any maximum matchings M and M' of G, (i) $f_M(\bar{P}_M, \bar{P}_{M'}) = |\bar{P}_M|$, and (ii) for every subset T of players, $f_M(\bar{P}_M, T) = f_{M'}(\bar{P}_{M'}, T)$.*

If one treats Π like a flow in the network G_M, then $G_{M \oplus \Pi^+}$ behaves like the residual graph with respect to Π. Claims 2.2 and 2.3 below concerns with augmentation using $G_{M \oplus \Pi^+}$.

▶ **Claim 2.2.** *Let Π be a feasible solution of $G_M(S, T)$. Then, $M \oplus \Pi^+$ is a maximum matching of G, so the directed bipartite graph $G_{M \oplus \Pi^+}$ is well defined. Also, Π is an optimal solution of $G_M(S, T)$ if and only if $G_{M \oplus \Pi^+}$ contains no path from $S \setminus source(\Pi)$ to $T \setminus sink(\Pi)$.*

▶ **Claim 2.3.** *Let Π be a feasible solution of $G_M(S, T)$. Suppose that $G_{M \oplus \Pi^+}$ contains a path π from $S \setminus source(\Pi)$ to $T \setminus sink(\Pi)$. We can use π to augment Π to a feasible solution Π' of $G_M(S, T)$ such that $|\Pi'| = |\Pi| + 1$, the vertex set of Π' is a subset of the vertices in $\Pi \cup \{\pi\}$, $source(\Pi') = source(\Pi) \cup \{source(\pi)\}$, and $sink(\Pi') = sink(\Pi) \cup \{sink(\pi)\}$.*

We can also push flow along a path in $G_{M \oplus \Pi^+}$ from $sink(\Pi)$ to T. This reroutes the flow in G_M without changing its flow value. Claim 2.4 below gives a precise statement.

▶ **Claim 2.4.** *Let Π be a feasible solution of $G_M(S, T)$. Suppose that there is a non-trivial path π in $G_{M \oplus \Pi^+}$ from $sink(\Pi)$ to T. Clearly, $sink(\pi) \notin sink(\Pi)$ because every node in $sink(\Pi)$ has zero in-degree in $G_{M \oplus \Pi^+}$. We can use π to convert Π to a feasible solution Π' of $G_M(S, T)$ such that $|\Pi'| = |\Pi|$, the vertex set of Π' is a subset of the vertices in $\Pi \cup \{\pi\}$, $source(\Pi') = source(\Pi)$, and $sink(\Pi') = (sink(\Pi) \setminus \{source(\pi)\}) \cup \{sink(\pi)\}$.*

3 The Algorithm

3.1 Overview

We give an overview of the common framework that our algorithm shares with that in [1]. Let M and $\mathcal{E}$ denote 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.

To satisfy p_0, the simplest case is that we can find a minimal thin edge (p_0, B) such that $value(B)$ is at least τ/λ and B excludes the resources covered by edges in $\mathcal{E}$, i.e., not blocked by any thin edge in $\mathcal{E}$. We can extend the partial allocation by adding (p_0, B) to $\mathcal{E}$.

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

We may have the situation that the thin edge (q, B) mentioned above is blocked by some thin edges in $\mathcal{E}$. Pick such a (q, B) arbitrarily, and call it (q_0, B_0). Let $\{(p_1, B_1'), \ldots, (p_k, B_k')\}$ be the thin edges in $\mathcal{E}$ that block (q_0, B_0), i.e., $B_0 \cap B_i' \neq \emptyset$ for $i \in [1, k]$. To make (q_0, B_0) unblocked, we need to satisfy each player p_i, $i \in [1, k]$, with a fat resource or another thin edge. 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 $\mathcal{X}_2 = \{(q_0, B_0)\}$ 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 $value(B_1)$ is at least τ/λ and G_M contains two node-disjoint paths from $\{p_0\} \cup Y_2$ to $\{q_0, q_1\}$, and we also require B_1 to exclude the resources already covered by thin edges in the current stack (i.e., $\mathcal{X}_2$ and $\mathcal{Y}_2$) because the current plan to satisfy q_0 in the future involves some of these thin resources. If (q_1, B_1) is blocked by thin edges in $\mathcal{E}$, we initialize a set $\mathcal{X}_3 = \{(q_1, B_1)\}$; 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 either reachable from 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. We can remove a blocking edge from $\mathcal{Y}_2$ without affecting the alternating path from p_0 to q_0. 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 large progress can be made.

Since there are multiple players in Y_2 to be satisfied, we continue to look for another minimal thin edge (q_2, B_2) such that G_M contains three node-disjoint paths from $\{p_0\} \cup Y_2$ to $\{q_0, q_1, q_2\}$, $value(B_2) \geqslant \tau/\lambda$, and B_2 excludes the resources covered by thin edges in the current stack (i.e., $\mathcal{X}_2 \cup \mathcal{Y}_2 \cup \mathcal{X}_3$) and $\mathcal{I}$. If (q_2, B_2) is blocked by thin edges in $\mathcal{E}$, we add (q_2, B_2) to $\mathcal{X}_3$; otherwise, we add it to $\mathcal{I}$. After collecting all such thin edges in $\mathcal{X}_3$ and $\mathcal{I}$, we construct the set $\mathcal{Y}_3$ of thin edges in the current partial allocation that block $\mathcal{X}_3$. Then, we add a new top layer to the stack that stores $\mathcal{X}_3$ and $\mathcal{Y}_3$ among other things for bookkeeping. Then, we turn our attention to satisfying the players in $\mathcal{Y}_3$ 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 $\mathcal{I}$ to satisfy a predetermined fraction of players in $\mathcal{Y}_l$ for some l, and then we shrink this layer and delete all layers above it. The above is repeated until $\mathcal{I}$ is not large enough to satisfy the predetermined fraction of players in any $\mathcal{Y}_l$ 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, that is stored in the bottommost layer in the stack.

A greedy strategy is also used for achieving a polynomial running time. In [1], when a blocked thin edge (q, B) is picked and added to $\mathcal{X}_l$ for some l, B is required to be a minimal set of value at least $\tau/2$, which is more than τ/λ. Intuitively, if such an edge is blocked, it must be blocked by many edges. Hence, the strategy leads to fast growth of 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 (q, B) with (nearly) the largest value. Our strategy leads to faster growth of the stack, and hence, a polynomial running time can be achieved for smaller λ.

3.2 Notation and definitions

A *state of the algorithm* consists of several components, namely, M, $\mathcal{E}$, a stack of layers, and a global variable $\mathcal{I}$ that stores a set of thin edge. The layers in the stack are indexed starting from 1 at the bottom. For $i \geqslant 1$, the i-th layer is a 4-tuple $(\mathcal{X}_i, \mathcal{Y}_i, d_i, z_i)$, where $\mathcal{X}_i$ and $\mathcal{Y}_i$ 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 set of players covered by edges in $\mathcal{I}$, $\mathcal{X}_i$ and $\mathcal{Y}_i$, respectively. For any $k \geqslant 1$, let $\mathcal{X}_{\leqslant k}$ denote $\bigcup_{i=1}^{k} \mathcal{X}_i$, and $\mathcal{Y}_{\leqslant k}$, $X_{\leqslant k}$, and $Y_{\leqslant k}$ are similarly defined.

The sets $\mathcal{X}_i$ and $\mathcal{Y}_i$ are defined inductively. At the beginning of the algorithm, $\mathcal{X}_1 = \emptyset$, $\mathcal{Y}_1 = \{(p_0, \emptyset)\}$, $d_1 = z_1 = 0$, and $\mathcal{I} = \emptyset$. The first layer in the stack is thus $(\emptyset, \{(p_0, \emptyset)\}, 0, 0)$.

Consider the construction of the $(k+1)$-th layer in an execution of the build phase. When it first starts, $\mathcal{X}_{k+1}$ is initialized to be empty. We say that p is **addable** if $f_M(Y_{\leqslant k}, X_{\leqslant k+1} \cup I \cup \{p\}) > f_M(Y_{\leqslant k}, X_{\leqslant k+1} \cup I)$. Note that this definition depends on $X_{\leqslant k+1} \cup I$, so adding edges to $\mathcal{X}_{k+1}$ and $\mathcal{I}$ may affect the addability of players. Given an addable player p, we say that a thin edge (p, B) is **addable** if $value(B) \in [\tau/\lambda, \tau + \tau/\lambda]$ and B excludes resources currently in $\mathcal{X}_{\leqslant k+1} \cup \mathcal{Y}_{\leqslant k} \cup \mathcal{I}$. An addable thin edge (p, B) is **unblocked** if there exists a subset $B' \subseteq B$ such that $value(B') \geqslant \tau/\lambda$ and B' excludes resources used in $\mathcal{E}$. Otherwise, (p, B) is **blocked**. During the construction of the $(k+1)$th layer, the algorithm adds some blocked addable thin edges to $\mathcal{X}_{k+1}$ and some unblocked addable thin edges to $\mathcal{I}$. When the growth of $\mathcal{X}_{k+1}$ stops, the algorithm constructs $\mathcal{Y}_{k+1}$ as the set of the thin edges in $\mathcal{E}$ that share resource(s) with some edge(s) in $\mathcal{X}_{k+1}$. Edges in $\mathcal{Y}_{k+1}$ are called **blocking** edges.

After constructing $\mathcal{X}_{k+1}$ and $\mathcal{Y}_{k+1}$ and growing $\mathcal{I}$, we define $d_{k+1} := f_M(Y_{\leqslant k}, X_{\leqslant k+1} \cup I)$ and $z_{k+1} := |X_{k+1}|$. The values d_{k+1} and z_{k+1} do not change once computed unless the layer L_{k+1} is destructed in the collapse phase, although $f_M(Y_{\leqslant k}, X_{\leqslant k+1} \cup I)$ and $|X_{k+1}|$ may change subsequently. The values d_{k+1} and z_{k+1} are introduced only for the analysis.

Whenever we complete the construction of a new layer in the stack, we check whether any existing layer is collapsible. If so, we leave the build phase and enter the collapse phase, during which the stack is shrunk and the current partial allocation is updated. 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.

3.3   Build phase

Assume that the stack currently contains layers $L_1, \ldots, L_k$ with L_k at the top. Let M and $\mathcal{E}$ denote the maximum matching in G and the set of thin edges in the current partial allocation, respectively. The following routine BUILD constructs the next layer L_{k+1}.

BUILD$(M, \mathcal{E}, \mathcal{I}, (L_1, \cdots, L_k))$
1. Initialize $\mathcal{X}_{k+1}$ 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 $value(B') \geqslant \tau/\lambda$ and B' excludes the resources used in $\mathcal{E}$ (we call (p, B') a *minimal unblocked addable edge*),
    b. add (p, B') to $\mathcal{I}$,
    c. go back to step 2.
3. When we come to step 3, no unblocked addable edge is left. If there are no (blocked) addable edges, 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'). Pick the edge with the largest value among those identified, add it to $\mathcal{X}_{k+1}$, and repeat step 3.
4. At this point, the construction of $\mathcal{X}_{k+1}$ is complete. Let $\mathcal{Y}_{k+1}$ be the set of the thin edges in $\mathcal{E}$ that share resource(s) with some thin edge(s) in $\mathcal{X}_{k+1}$.
5. Compute $d_{k+1} := f_M(Y_{\leqslant k}, X_{\leqslant k+1} \cup I)$ and $z_{k+1} := |X_{k+1}|$.
6. Push the new layer $L_{k+1} = (\mathcal{X}_{k+1}, \mathcal{Y}_{k+1}, d_{k+1}, z_{k+1})$ onto the stack.

BUILD differs from its counterpart in [1] in several places, particularly in step 3. First, we requires blocked addable edges to be maximal while [1] only considers minimal addable edges of value at least $\tau/2$. Second, when adding addable edges to $\mathcal{X}_{k+1}$, we pick the one with (nearly) the largest value. In contrast, [1] arbitrarily picks one addable edge.

■ **Table 1** Let ℓ denote the highest layer index in the current stack. Let M and $\mathcal{E}$ be the maximum matching and the set of thin edges in the current partial allocation.

Invariant 1	Every edge in $\mathcal{I}$ has value in $[\tau/\lambda, 2\tau/\lambda]$. Every edge in $\mathcal{X}_{\leqslant\ell}$ has value in $[\tau/\lambda, \tau + \tau/\lambda]$. No two edges from $\mathcal{X}_{\leqslant\ell}$ and $\mathcal{I}$ (both edges from either set or one edge from each set) cover the same player or share any resource.
Invariant 2	No edge in $\mathcal{E}$ shares any resource with any edge in $\mathcal{I}$.
Invariant 3	For all $i \in [1, \ell]$, every edge in $\mathcal{X}_i$ shares some resource(s) with some edge(s) in $\mathcal{Y}_i$ but not with any edge in $\mathcal{E} \setminus \mathcal{Y}_i$.
Invariant 4	$\mathcal{Y}_2, \ldots, \mathcal{Y}_\ell$ are disjoint subsets of $\mathcal{E}$. ($\mathcal{Y}_1 = \{(p_0, \emptyset)\}$ is not.)
Invariant 5	For all $i \in [1, \ell]$, no edge in $\mathcal{Y}_i$ shares any resource with any edge in $\mathcal{X}_j$ for any $j \neq i$.
Invariant 6	$f_M(Y_{\leqslant\ell-1}, I) = |I|$.
Invariant 7	For all $i \in [1, \ell-1]$, $f_M(Y_{\leqslant i}, X_{\leqslant i+1} \cup I) \geqslant d_{i+1}$.

Is it possible that $\mathcal{X}_{k+1} = \emptyset$ and $\mathcal{Y}_{k+1} = \emptyset$? We will establish Lemma 4.1 in Section 5.2, which implies that if $\mathcal{Y}_{k+1}$ is empty, then some layer below L_{k+1} is collapsible. As a result, the algorithm will enter the collapse phase next and L_{k+1} will be removed.

▶ **Lemma 3.1.** BUILD *runs in* $poly(m, n)$ *time.*

▶ **Lemma 3.2.** BUILD *maintains invariants 1–7 in Table 1.*

3.4 Collapse phase

Let M be the maximum matching in the current partial allocation. Let $(L_1, L_2, \ldots, L_\ell)$ be the current stack. Deciding whether a layer can be collapsed requires a decomposition of $\mathcal{I}$.

Collapsibility. Let $\mathcal{I}_1 \cup \mathcal{I}_2 \cup \cdots \mathcal{I}_\ell$ be some partition of $\mathcal{I}$. Let I_i denote the set of players covered by $\mathcal{I}_i$. We use $\mathcal{I}_{\leqslant j}$ and $I_{\leqslant j}$ to denote $\bigcup_{i=1}^{j} \mathcal{I}_i$ and $\bigcup_{i=1}^{j} I_i$, respectively. Note that $|I_i| = |\mathcal{I}_i|$ by invariant 1. The partition $\mathcal{I}_1 \cup \mathcal{I}_2 \cup \cdots \mathcal{I}_\ell$ is a **canonical decomposition** of $\mathcal{I}$ if for all $i \in [1, \ell]$, $f_M(Y_{\leqslant i}, I_{\leqslant i}) = f_M(Y_{\leqslant i}, I) = |I_{\leqslant i}| = |\mathcal{I}_{\leqslant i}|$. [1]

▶ **Lemma 3.3** ([1]). *In* $poly(\ell, m, n)$ *time, one can compute a canonical decomposition* $\mathcal{I}_1 \cup \mathcal{I}_2 \cup \ldots \mathcal{I}_\ell$ *of* $\mathcal{I}$ *and a canonical solution of* $G_M(Y_{\leqslant\ell}, I)$ *which can be partitioned into a disjoint union* $\Gamma_1 \cup \Gamma_2 \cup \cdots \Gamma_\ell$ *such that for every* $i \in [1, \ell]$, Γ_i *is a set of* $|I_i|$ *paths from* Y_i *to* I_i.

The canonical decomposition and solution can be obtained by starting with an optimal solution of $G_M(Y_1, I)$ and successively augment it (using Claim 2.3) to optimal solutions of $G_M(Y_{\leqslant 2}, I), \ldots, G_M(Y_{\leqslant\ell}, I)$. The resulting optimal solution of $G_M(Y_{\leqslant\ell}, I)$ is an canonical solution, and also induces a canonical decomposition of $\mathcal{I}$.

Consider Γ_i. The sources (which are also sinks) of the trivial paths in Γ_i can be satisfied by a new thin edge from $\mathcal{I}_i$. Recall that the non-trivial paths in Γ_i are alternating paths of M. Their sources can be satisfied by fat resources if we flip these alternating paths and satisfy the sinks with thin edges from $\mathcal{I}_i$. If we do so, then edges in $\mathcal{Y}_i$ that cover $source(\Gamma_i)$ can be safely removed from $\mathcal{E}$ as the players in $source(\Gamma_i)$ are satisfied by new edges, and from $\mathcal{Y}_i$ since 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 $i \in [0, \ell]$, L_i is **collapsible** if there is a canonical decomposition $\mathcal{I}_0 \cup \mathcal{I}_1 \cup \ldots \mathcal{I}_\ell$ of $\mathcal{I}$ such that $|I_i| \geqslant \mu|Y_i|$, where μ is a constant that will be determined later.

Collapse layers. When we find that some layer is collapsible, we run the routine Collapse below, which collapses layers in the stack until no layer is collapsible. Collapse works in the same manner as its counterpart in [1], but there are small differences in the presentation.

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

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

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

3. Recall that $source(\Gamma_t) \subseteq Y_t$ by Lemma 3.3. Let $\mathcal{V}$ denote the set of the thin edges in $\mathcal{Y}_t$ that cover $source(\Gamma_t)$. Recall that $\mathcal{Y}_t \subseteq \mathcal{E}$.

    a. Update the maximum matching M by flipping the non-trivial paths in Γ_t, i.e., set $M := M \oplus \Gamma_t^+$. This matches the sources of non-trivial paths in Γ_t while leave their sinks unmatched.

    b. Add to $\mathcal{E}$ edges in I_t, i.e., set $\mathcal{E} := \mathcal{E} \cup \mathcal{I}_t$. Now the sinks of non-trivial paths are satisfied. Also the sources of trivial paths are satisfied by new thin edges.

    c. Now each player in $source(\Gamma_t)$ is satisfied either by a fat resource or a thin edge from $\mathcal{I}_t$. Edges in $\mathcal{V}$ can be safely removed from $\mathcal{E}$. Set $\mathcal{E} := \mathcal{E} \setminus \mathcal{V}$. Consequently, edges in $\mathcal{V}$ no longer block edges in $\mathcal{X}_t$. Set $\mathcal{Y}_t := \mathcal{Y}_t \setminus \mathcal{V}$.

4. If $t \geqslant 2$, 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:

    a. Remove (p, B) from $\mathcal{X}_t$.

    b. If $f_M(Y_{\leqslant t-1}, I \cup \{p\}) > f_M(Y_{\leqslant t-1}, I)$, then add (p, B') to $\mathcal{I}$, where B' is an arbitrary minimal subset of B such that $value(B') \geqslant \tau/\lambda$ and B' excludes the resources covered by $\mathcal{E}$.

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 $\ell := t$ and go back to step 1.

▶ **Lemma 3.4.** Collapse *maintains invariants 1–7 in Table 1.*

<h1>4   Polynomial running time and binary search</h1>

Each call of Build and Collapse runs in time polynomial in ℓ, m and n. Lemma 4.1 below is the key to obtaining a bound on ℓ and the total number of calls of Build and Collapse. The proof of Lemma 4.1 is deferred to Section 5.

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

Lemma 4.1, immediately implies a logarithmic bound on the maximum number ℓ of layers. Using argument similar to that in [1, Lemmas 4.10 and 4.11], we 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 of value at least τ/λ.

The remaining task is to binary search for τ^*. If we use a value τ that is at most τ^*, 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 $|Y_{i+1}| < \sqrt{\mu}|Y_{\leqslant i}|$ for some $i \in [1, \ell - 1]$. In this case, the algorithm returns in polynomial time an allocation of value at least $\tau/\lambda \geqslant \tau^*/\lambda$. The second outcome is that no layer is collapsible at some point, but $|Y_{i+1}| < \sqrt{\mu}|Y_{\leqslant i}|$ for some $i \in [1, \ell - 1]$. This can be detected in $O(1)$ time by maintaining $|Y_{i+1}|$ and $|Y_{\leqslant i}|$, which allows us to detect that $\tau > \tau^*$ and halt the algorithm. Since this is the first violation of this property, 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 δ requires a smaller μ and hence a higher running time.

▶ **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 $(6 + \delta)$-approximation in time polynomial in the number of players and the number of resources.*

5   Analysis

We will derive 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.

5.1   Competing players

To analyze our aggressive greedy strategy for selecting blocked addable thin edges, we need an injective map φ from the players covered by them to players who can access thin resources of high total value. The next result shows that these target players exist.

▶ **Lemma 5.1.** *Let* OPT *be an arbitrary optimal allocation. 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 $\bar{P}_{M^*}$ is assigned only thin resources in* OPT *that are worth a total value of τ or more, assuming that $\tau \leqslant \tau^*$.*

The domain of the injection φ is a subset of $X_{\leqslant \ell}$ and its image is a subset of $\bar{P}_{M^*}$. We call the image of φ the competing players. For any player $q \in X_{\leqslant \ell}$, $\varphi(q)$ has access to thin resources that are worth a total value of τ or more. Our goal is to prove that $\varphi(q)$ is also an addable player when q is added to $\mathcal{X}_{\leqslant \ell}$. Since the algorithm prefers q to $\varphi(q)$, either no addable edge is incident to $\varphi(q)$ or the maximal addable edge identified for $\varphi(q)$ has less value than the edge $e_q \in \mathcal{X}_{\leqslant \ell}$ that covers q. In both cases, more than $\tau - value(e_q)$ worth of thin resources assigned to $\varphi(q)$ in OPT are already in the stack. This will allow us to prove a good lower bound for the total value of the thin resources in the stack.

Lemma 5.2 below states the properties of competing players. We already discussed the usage of Lemma 5.2(i) and (ii). It would be ideal if the domain of φ could cover the entire $X_{\leqslant \ell}$. However, for technical reasons, when Collapse removes a player from $X_{\leqslant \ell}$, we may have to remove two players from the domain of φ in order to maintain the properties of φ. Lemma 5.2(iii) puts a lower bound on the size of the domain of φ. When deriving lower bound for the total value of the thin resources in the stack, the players in $\bar{P}_{M^*}$ that are not competing players and outside $X_{\leqslant \ell} \cup I$ also play a role. Lemma 5.2(iv) will allow us to prove that a large subset of such players are still addable after we finish adding edges to $\mathcal{X}_\ell$ during the construction of layer L_ℓ. Each of these addable players contributes a large worth of thin resources to 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 φ such that*

(i) *The domain D_φ and image Im_φ of φ are subsets of $X_{\leqslant \ell}$ and $\bar{P}_{M^*}$, respectively.*

(ii) *For every player $p \in D_\varphi$, when p was added to X_k for some $k \in [2, \ell]$, $\varphi(p)$ was also an addable player at that time.*

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

(iv) $f_M(\bar{P}_M, (\bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X_{\leqslant \ell}) = |\bar{P}_M|$.

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 φ is trivial as its domain $D_\varphi \subseteq X_1 = \emptyset$. So $\mathrm{Im}_\varphi = \emptyset$. Then, (i), (ii) and (iii) are satisfied trivially, and (iv) follows from Claim 2.1(i). We discuss how to update φ during the build and collapse phases.

Build phase. Suppose that BUILD begins to construct a new layer L_ℓ. X_ℓ is initialized to be empty. The value z_ℓ is computed only at the completion of L_ℓ. However, in this proof, we initialize $z_\ell = 0$, increment z_ℓ whenever we add an edge to X_ℓ, and show the validity of (i)–(iv) inductively.

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

Step 2 of BUILD does not change X_ℓ, and so φ needs no update.

Consider step 3 of BUILD. Suppose that a thin edge incident to player q_1 is added to $\mathcal{X}_\ell$. So q_1 is addable. For clarity, we use X'_ℓ, z'_ℓ, φ', $D_{\varphi'}$, and $\mathrm{Im}_{\varphi'}$ to denote the updated X_ℓ, z_ℓ, φ, D_φ, and Im_φ, 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 $p \in D_{\varphi'} \setminus \{q_1\}$, we set $\varphi'(p) := \varphi(p)$. We set $\varphi'(q_1)$ as follows.

Let Π_1 be an optimal solution of $G_M(Y_{\leqslant \ell - 1}, X_{\leqslant \ell} \cup I \cup \{q_1\})$. We have $q_1 \in sink(\Pi_1)$ since otherwise we would have $f_M(Y_{\leqslant \ell - 1}, X_{\leqslant \ell} \cup I \cup \{q_1\}) = f_M(Y_{\leqslant \ell - 1}, X_{\leqslant \ell} \cup I)$, contradicting the addability of q_1. Similarly, $q_1 \notin X_{\leqslant \ell}$. As $q_1 \in sink(\Pi_1)$, q_1 must be unmatched in $M \oplus \Pi_1^+$, i.e., $q_1 \in \bar{P}_{M \oplus \Pi_1^+}$. Let Π_2 be an optimal solution of $G_{M \oplus \Pi_1^+}(\bar{P}_{M \oplus \Pi_1^+}, (\bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X_{\leqslant \ell})$. We have $|\Pi_2| = f_{M \oplus \Pi_1^+}(\bar{P}_{M \oplus \Pi_1^+}, (\bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X_{\leqslant \ell}) = f_M(\bar{P}_M, (\bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X_{\leqslant \ell})$ by Claim 2.1(ii). Then, inductive assumption gives $|\Pi_2| = |\bar{P}_M| = |\bar{P}_{M \oplus \Pi_1^+}|$ (both M and $M \oplus \Pi_1$ are maximum matchings). So $\bar{P}_{M \oplus \Pi_1^+} = source(\Pi_2)$, implying that there is a path $\pi \in \Pi_2$ originating from q_1. Let $q_2 = sink(\pi)$.

We claim that $q_2 \notin X_{\leqslant \ell}$. If π is a trivial path, the claim is true because $q_2 = q_1 \notin X_{\leqslant \ell}$. Suppose that π is non-trivial. Suppose, for the sake of contradiction, that $q_2 \in X_{\leqslant \ell}$. This allows us to apply Claim 2.4 and use π to convert Π_1 to an equal-sized set of node-disjoint paths from $Y_{\leqslant \ell - 1}$ to $X_{\leqslant \ell} \cup I$. But then $f_M(Y_{\leqslant \ell - 1}, X_{\leqslant \ell} \cup I) \geqslant |\Pi_1| = f_M(Y_{\leqslant \ell - 1}, X_{\leqslant \ell} \cup I \cup \{q_1\}) \geqslant f_M(Y_{\leqslant \ell - 1}, X_{\leqslant \ell} \cup I)$. That is, $f_M(Y_{\leqslant \ell - 1}, X_{\leqslant \ell} \cup I \cup \{q_1\}) = f_M(Y_{\leqslant \ell - 1}, X_{\leqslant \ell} \cup I)$, contradicting the addability of q_1. This proves our claim that $q_2 \notin X_{\leqslant \ell}$.

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

Properties (i) and (iii) are straightforwardly satisfied by φ', z'_ℓ, $D_{\varphi'}$, and X'_ℓ.

By induction assumption, (ii) holds for players in $D_{\varphi'} \setminus \{q_1\} = D_\varphi$. It remains to check the validity of (ii) for $\varphi'(q_1) = q_2$. If π is a trivial path, then (ii) holds because $q_2 = q_1$ and q_1 is addable. Assume that π is non-trivial. By Claim 2.4, we can use π to convert Π_1 to an equal-sized set of node-disjoint paths in G_M from $Y_{\leqslant \ell - 1}$ to $X_{\leqslant \ell} \cup I \cup \{q_2\}$. Thus, $f_M(Y_{\leqslant \ell - 1}, X_{\leqslant \ell} \cup I \cup \{q_2\}) \geqslant |\Pi_1| = f_M(Y_{\leqslant \ell - 1}, X_{\leqslant \ell} \cup I \cup \{q_1\}) = f_M(Y_{\leqslant \ell - 1}, X_{\leqslant \ell} \cup I) + 1$ as q_1 is addable. Therefore, q_2 is also an addable player at the time when $\mathcal{X}_\ell$ gains a thin edge incident to q_1.

Consider (iv). If π is a trivial path, i.e., $q_1 = q_2$, then (iv) holds because $(\bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X_{\leqslant \ell} \subseteq (\bar{P}_{M^*} \setminus (\mathrm{Im}_\varphi \cup \{q_2\})) \cup X_{\leqslant \ell} \cup \{q_1\} = (\bar{P}_{M^*} \setminus \mathrm{Im}_{\varphi'}) \cup X'_{\leqslant \ell}$. Suppose that π is non-trivial. Recall that Π_2 is an optimal solution of $G_{M \oplus \Pi_1^+}(\bar{P}_{M \oplus \Pi_1^+}, \bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X_{\leqslant \ell})$,

and $|\Pi_2| = |\bar{P}_{M \oplus \Pi_1^+}|$. 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 $|\bar{P}_{M \oplus \Pi_1^+}|$ sinks of Π_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 $M \oplus \Pi_1^+$. This implies that $(sink(\Pi_2) \setminus \{q_2\}) \cup \{q_1\}$ is exactly $\bar{P}_{M'}$. Since $sink(\Pi_2) \subseteq (\bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X_{\leqslant \ell}$, we get $\bar{P}_{M'} \subseteq \big(((\bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X_{\leqslant \ell}) \setminus \{q_2\}) \cup \{q_1\} \subseteq (\bar{P}_{M^*} \setminus (\mathrm{Im}_\varphi \cup \{q_2\})) \cup X_{\leqslant \ell} \cup \{q_1\} = (\bar{P}_{M^*} \setminus \mathrm{Im}_{\varphi'}) \cup X'_{\leqslant \ell}$. Then, we can apply Claim 2.1(i) to obtain $|\bar{P}_M| \geqslant f_M(\bar{P}_M, (\bar{P}_{M^*} \setminus \mathrm{Im}_{\varphi'}) \cup X'_{\leqslant \ell}) \geqslant f_M(\bar{P}_M, \bar{P}_{M'}) = |\bar{P}_M|$. Hence, (iv) holds.

Clearly, steps 4–6 of BUILD do not affect φ.

Collapse phase. Suppose that we are going to collapse the layer L_t. Since we will set $\ell := t$ at the end of collapsing L_t, we only need to prove (i)—(iv) with ℓ substituted by t.

Clearly, step 1 of COLLAPSE has no effect on φ.

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 φ'' at that time that satisfies (i)–(iv). We set $\varphi := \varphi''$, $\mathrm{D}_\varphi := \mathrm{D}_{\varphi''}$, and $\mathrm{Im}_\varphi := \mathrm{Im}_{\varphi''}$.

In step 3 of COLLAPSE, the maximum matching M may change, so only (iv) is affected. Nonetheless, by Claim 2.1(ii), the value of $f_M(\bar{P}_M, (\bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X_\ell)$ remains the same after updating M. So (iv) is satisfied afterwards.

In step 4 of COLLAPSE, we may remove some edges from $\mathcal{X}_t$ and add some of these removed edges to $\mathcal{I}$. Adding edges to $\mathcal{I}$ does not affect φ. We need to update φ when an edge is removed from $\mathcal{X}_t$. Suppose that we are going to remove from $\mathcal{X}_t$ an edge that covers a player q_1. Recall that z_t was defined in the last construction of the layer L_t, and it has remained fixed despite possible changes to $\mathcal{X}_t$ since then. Let $X'_{\leqslant t}$, φ', $\mathrm{D}_{\varphi'}$, and $\mathrm{Im}_{\varphi'}$ denote the updated $X_{\leqslant t}$, φ, D_φ, and Im_φ, respectively. Note that $X'_{\leqslant t} = X_{\leqslant t} \setminus \{q_1\}$. We show how to define φ', $\mathrm{D}_{\varphi'}$, and $\mathrm{Im}_{\varphi'}$ appropriately.

Consider property (iv). If (iv) is not affected by the deletion of q_1, that is, $f_M(\bar{P}_M, (\bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X'_{\leqslant t}) = |\bar{P}_M|$, then we simply set $\mathrm{D}_{\varphi'} := \mathrm{D}_\varphi \setminus \{q_1\}$ and $\varphi'(p) := \varphi(p)$ for all $p \in \mathrm{D}_{\varphi'}$. It is easy to verify that φ' satisfies (iv). Suppose that property (iv) is affected, and therefore, $f_M(\bar{P}_M, (\bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X'_{\leqslant t}) = |\bar{P}_M| - 1$. Since $f_M(\bar{P}_M, \bar{P}_{M^*}) = |\bar{P}_M|$, we have that $f_M(\bar{P}_M, \bar{P}_{M^*} \cup X'_{\leqslant t}) = |\bar{P}_M|$. Comparing the two equations above, we conclude that there must a player $q_2 \in \mathrm{D}_\varphi$ such that $f_M(\bar{P}_M, (\bar{P}_{M^*} \setminus (\mathrm{Im}_\varphi \setminus \{\varphi(q_2)\})) \cup X'_{\leqslant t}) = |\bar{P}_M|$. So we set $\mathrm{D}_{\varphi'} := \mathrm{D}_\varphi \setminus \{q_1, q_2\}$, and $\varphi'(p) := \varphi(p)$ for all $p \in \mathrm{D}_{\varphi'}$. Property (iv) is satisfied afterwards.

Irrespective of which definition of φ' above is chosen, 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 $|Y_{k+1}| < \sqrt{\mu} |Y_{\leqslant k}|$ but no layer below L_{k+1} is collapsible. Let k be the smallest such index. So $|Y_{i+1}| \geqslant \sqrt{\mu} |Y_{\leqslant i}|$ for every $i \in [1, k-1]$.

Consider the moment immediately after the last construction of the $(k+1)$-th layer. Let $(M', \mathcal{E}', \mathcal{I}', (L'_1, \ldots, L'_{k+1}))$ be the state of the algorithm at that moment. No layer below L'_{k+1} is collapsible immediately after the construction of L'_{k+1} since this is the last construction of the $(k+1)$-th layer. We will derive a few inequalities that hold given the existence of k. Then we will obtain a contradiction by showing that the system made up of these inequalities is infeasible.

We first define some notations. Let $\mathcal{X}'_i$ and $\mathcal{Y}'_i$ denote the set of blocked addable (thin) edges and the set of blocking (thin) edges associated with L'_i. Then, X'_i, Y'_i, $\mathcal{X}'_{\leqslant i}$, $\mathcal{Y}'_{\leqslant i}$, $X'_{\leqslant i}$, and $Y'_{\leqslant i}$ are correspondingly defined. Let M^* be a maximum matching induced by an optimal allocation OPT. Let φ' and $D_{\varphi'}$ be the injection and its domain associated with $X'_{\leqslant t+1}$ as defined in Lemma 5.2 with respect to M^*. For all $p \in D_{\varphi'}$, define $w_p := value(B)$, where (p, B) is the thin edge for p in $\mathcal{X}'_{\leqslant k+1}$. By invariant 1 in Table 1, w_p is well defined (as $D_{\varphi'} \subseteq X'_{\leqslant k+1}$ by Lemma 5.2(i) and no player is covered by two edges in $\mathcal{X}'_{\leqslant k+1}$) and $w_p \in [\tau/\lambda, \tau + \tau/\lambda]$.

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

$$value(\mathcal{X}'_{\leqslant k+1} \cup \mathcal{Y}'_{\leqslant k}) \geqslant value(\mathcal{X}'_{\leqslant k+1}) \geqslant \sum_{p \in D_{\varphi'}} w_p, \quad \frac{\tau}{\lambda}|D_{\varphi'}| \leqslant \sum_{p \in D_{\varphi'}} w_p \leqslant (\tau + \frac{\tau}{\lambda})|D_{\varphi'}|.$$

▶ **Claim 5.3.** $|D_{\varphi'}| \leqslant |Y'_{\leqslant k}|$.

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

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

The proofs of above claims use Lemma 5.2. In particular, Lemma 5.2 plays a key role in the proof of Claim 5.5. Here we give a rough idea. Consider the moment we just finish adding edges to $\mathcal{X}'_{k+1}$. Lemma 5.2(iv) ensures that roughly $(|Y'_{\leqslant k}| - |D_{\varphi'}|)$ players in $\bar{P}_{M^*} \setminus \text{Im}_{\varphi'}$ are still addable. Since there are no more addable edges (otherwise they will be added to $\mathcal{X}'_{k+1}$), each of these addable players can access less than τ/λ worth of thin resources that are not in the stack, and hence each of them contribute at least $\tau - \tau/\lambda$ worth of thin resources to the stack. This gives the first term $(\tau - \frac{\tau}{\lambda})(|Y'_{\leqslant k}| - |D_{\varphi'}|)$. For each player $\varphi'(p) \in \text{Im}_{\varphi'}$, as we explained in section 5.1, it contributed at least $\tau - w_p$ worth of thin resources to the stack at the time player p was picked. This gives the second term $\sum_{p \in D_{\varphi'}}(\tau - w_p)$. The third term is just slack in the analysis.

Putting all the inequalities together gives the following system.

- $value(\mathcal{X}_{\leqslant k+1} \cup \mathcal{Y}_{\leqslant k}) \geqslant \sum_{p \in D_{\varphi'}} w_p$,
- $\frac{\tau}{\lambda}|D_{\varphi'}| \leqslant \sum_{p \in D_{\varphi'}} w_p \leqslant (\tau + \tau/\lambda)|D_{\varphi'}|$,
- $|D_{\varphi'}| \leqslant |Y'_{\leqslant k}|$,
- $value(\mathcal{X}'_{\leqslant k+1} \cup \mathcal{Y}'_{\leqslant k}) \leqslant \frac{\tau}{\lambda}|D_{\varphi'}| + \frac{2\tau}{\lambda}|Y'_{\leqslant k}| + \frac{\delta_1\tau}{\lambda}|Y'_{\leqslant k}|$,
- $value(\mathcal{X}'_{\leqslant k+1} \cup \mathcal{Y}'_{\leqslant k}) \geqslant (\tau - \tau/\lambda)(|Y'_{\leqslant k}| - |D_{\varphi'}|) + \sum_{p \in D_{\varphi'}}(\tau - w_p) - \frac{\delta_2\tau}{\lambda}|Y'_{\leqslant k}|$.

Divide the above system by $\frac{\tau}{\lambda}|D_{\varphi'}|$. To simplify the notation, define the variables $B_1 := value(\mathcal{X}_{\leqslant k+1} \cup \mathcal{Y}_{\leqslant k})/(\frac{\tau}{\lambda}|D_{\varphi'}|)$, $B_2 := |Y'_{\leqslant k}|/|D_{\varphi'}|$, 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.

$$B_1 \geqslant B_3, \quad 1 \leqslant B_3 \leqslant \lambda + 1, \quad 1 \leqslant B_2, \quad B_1 \leqslant 1 + 2B_2 + \delta_1 B_2,$$
$$B_1 \geqslant (\lambda - 1)(B_2 - 1) + \lambda - B_3 - \delta_2 B_2.$$

The first, fourth, and fifth inequalities give $2(1 + 2B_2 + \delta_1 B_2) \geqslant B_1 + B_3 \geqslant (\lambda - 1)(B_2 - 1) + \lambda - \delta_2 B_2 \Rightarrow 2 + (4 + 2\delta_1)B_2 \geqslant (\lambda - 1 - \delta_2)B_2 + 1 \Rightarrow (\lambda - 5 - 2\delta_1 - \delta_2)B_2 \leqslant 1$. On the other hand, $\lambda - 5 - 2\delta_1 - \delta_2 > 1$ for a sufficiently small μ because when μ tends to zero, both δ_1 and δ_2 tend to 0. Hence, $(\lambda - 5 - 2\delta_1 - \delta_2)B_2 > 1$ as $B_2 \geqslant 1$ by the third inequality. But it is impossible that $(\lambda - 5 - 2\delta_1 - \delta_2)B_2 \leqslant 1$ and $(\lambda - 5 - 2\delta_1 - \delta_2)B_2 > 1$ simultaneously.

References

1   Chidambaram Annamalai, Christos Kalaitzis, and Ola Svensson. Combinatorial algorithm for restricted max-min fair allocation. *ACM Trans. Algorithms*, 13(3):37:1–37:28, 2017.

2   Arash Asadpour, Uriel Feige, and Amin Saberi. Santa claus meets hypergraph matchings. *ACM Trans. Algorithms*, 8(3):24:1–24:9, 2012.

3   Arash Asadpour and Amin Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. In *Proc. 39th ACM Symposium on Theory of Computing*, pages 114–121, 2007.

4   Nikhil Bansal and Maxim Sviridenko. The santa claus problem. In *Proc. 38th ACM Symposium on Theory of Computing*, pages 31–40, 2006.

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

6   Deeparnab Chakrabarty, Julia Chuzhoy, and Sanjeev Khanna. On allocating goods to maximize fairness. In *Proc. 50th IEEE Symposium on Foundations of Computer Science*, pages 107–116, 2009.

7   T.-H. Hubert Chan, Zhihao Gavin Tang, and Xiaowei Wu. On (1, epsilon)-restricted max-min fair allocation problem. In *Proc. 27th International Symposium on Algorithms and Computation*, volume 64, pages 23:1–23:13, 2016.

8   Uriel Feige. On allocations that maximize fairness. In *Proc. 19th ACM-SIAM Symposium on Discrete Algorithms*, pages 287–293, 2008.

9   Daniel Golovin. Max-min fair allocation of indivisible good. *Technical report, Carnegie Mellon University*, 2005.

10 Bernhard Haeupler, Barna Saha, and Aravind Srinivasan. New constructive aspects of the lovász local lemma. *Journal of the ACM*, 58(6):28:1–28:28, 2011.

11 László Lovász and Michael D. Plummer. *Matching Theory*. American mathematical society, 2009.

12 Barna Saha and Aravind Srinivasan. A new approximation technique for resource-allocation problems. In *Proc. 1st Symposium on Innovations in Computer Science*, pages 342–357, 2010.