Quasi-polynomial Local Search for Restricted Max-Min Fair Allocation*

Lukas Polacek[1] and Ola Svensson[2]

[1] KTH Royal Institute of Technology, Sweden
polacek@csc.kth.se
[2] EPFL, Switzerland
ola.svensson@epfl.ch

Abstract. The restricted max-min fair allocation problem (also known as the restricted Santa Claus problem) is one of few problems that enjoys the intriguing status of having a better estimation algorithm than approximation algorithm. Indeed, Asadpour et al. [1] proved that a certain configuration LP can be used to estimate the optimal value within a factor $1/(4+\epsilon)$, for any $\epsilon > 0$, but at the same time it is not known how to efficiently find a solution with a comparable performance guarantee.

A natural question that arises from their work is if the difference between these guarantees is inherent or because of a lack of suitable techniques. We address this problem by giving a quasi-polynomial approximation algorithm with the mentioned performance guarantee. More specifically, we modify the local search of [1] and provide a novel analysis that lets us significantly improve the bound on its running time: from $2^{O(n)}$ to $n^{O(\log n)}$. Our techniques also have the interesting property that although we use the rather complex configuration LP in the analysis, we never actually solve it and therefore the resulting algorithm is purely combinatorial.

1 Introduction

We consider the problem of indivisible resource allocation in the following classical setting: a set $\mathcal{R}$ of available resources shall be allocated to a set $\mathcal{P}$ of players where the value of a set of resources for player i is given by the function $f_i : 2^{\mathcal{R}} \mapsto \mathbb{R}$. This is a very general setting and dependent on the specific goals of the allocator several different objective functions have been studied.

One natural objective, recently studied in [7,8,11,15], is to maximize the social welfare, i.e., to find an allocation $\pi : \mathcal{R} \mapsto \mathcal{P}$ of resources to players so as to maximize $\sum_{i \in \mathcal{P}} f_i(\pi^{-1}(i))$. However, this approach is not suitable in settings where the property of "fairness" is desired. Indeed, it is easy to come up with examples where an allocation that maximizes the social welfare assigns all resources to even a single player. In this paper we address this issue by studying algorithms for finding "fair" allocations. More specifically, fairness is modeled by evaluating an allocation with respect to the satisfaction of the least happy player, i.e., we wish to find an allocation π that maximizes $\min_{i \in \mathcal{P}} f_i(\pi^{-1}(i))$.

* A full version of this paper is available at http://arxiv.org/abs/1205.1373. This research was supported by ERC Advanced investigator grants 228021 and 226203.

A. Czumaj et al. (Eds.): ICALP 2012, Part I, LNCS 7391, pp. 726–737, 2012.

In contrast to maximizing the social welfare, the problem of maximizing fairness is already NP-hard when players have linear value functions. In order to simplify notation for such functions we denote $f_i(j)$ by $v_{i,j}$ and hence we have that $f_i(\pi^{-1}(i)) = \sum_{j \in \pi^{-1}(i)} v_{i,j}$. This problem has recently received considerable attention in the literature and is often referred to as the *max-min fair allocation* or the *Santa Claus* problem.

One can observe that the max-min fair allocation problem is similar to the classic problem of scheduling jobs on unrelated machines to minimize the makespan, where we are given the same input but wish to find an allocation that minimizes the maximum instead of one that maximizes the minimum. In a classic paper [13], Lenstra, Shmoys & Tardos gave a 2-approximation algorithm for the scheduling problem and proved that it is NP-hard to approximate the problem within a factor less than 1.5. The key step of their 2-approximation algorithm is to show that a certain linear program, often referred to as the assignment LP, yields an additive approximation of $v_{\max} = \max_{i,j} v_{i,j}$. Bezáková and Dani [5] later used these ideas for max-min fair allocation to obtain an algorithm that always finds a solution of value at least $OPT - v_{\max}$, where OPT denotes the value of an optimal solution. However, in contrast to the scheduling problem, this algorithm and more generally the assignment LP gives no approximation guarantee for max-min fair allocation in the challenging cases when $v_{\max} \geq OPT$.

In order to overcome this obstacle, Bansal & Sviridenko [3] proposed a stronger linear program relaxation, known as the configuration LP, for the max-min fair allocation problem. The configuration LP that we describe in detail in Section 2 has been vital to the recent progress on better approximation guarantees. Asadpour & Saberi [2] used it to obtain a $\Omega(1/\sqrt{|\mathcal{P}|}(\log|\mathcal{P}|)^3)$-approximation algorithm which was later improved by Bateni et al. [4] and Chakrabarty et al. [6] to algorithms that return a solution of value at least $\Omega(OPT/|\mathcal{P}|^\epsilon)$ in time $O(|\mathcal{P}|^{1/\epsilon})$.

The mentioned guarantee $\Omega(OPT/|\mathcal{P}|^\epsilon)$ is rather surprising because the integrality gap of the configuration LP is no better than $O(OPT/\sqrt{|\mathcal{P}|})$ [3]. However, in contrast to the general case, the configuration LP is significantly stronger for the prominent special case where values are of the form $v_{i,j} \in \{v_j, 0\}$. This case is known as the *restricted* max-min fair allocation or the restricted Santa Claus problem and is the focus of our paper. The worst known integrality gap for the restricted case is 1/2 and it is known [5] that it is NP-hard to beat this factor (which is also the best known hardness result for the general case). Bansal & Sviridenko [3] first used the configuration LP to obtain an $O(\log\log\log|\mathcal{P}|/\log\log|\mathcal{P}|)$-approximation algorithm for the restricted max-min fair allocation problem. They also proved several structural properties that were later used by Feige [9] to prove that the integrality gap of the configuration LP is in fact constant in the restricted case. The proof is based on repeated use of Lovász local lemma and was turned into a polynomial time algorithm [12].

The approximation guarantee obtained by combining [9] and [12] is a large constant and is far away from the best known analysis of the configuration LP by Asadpour et al. [1]. More specifically, they proved in [1] that the integrality gap is lower bounded by 1/4 by designing a beautiful local search algorithm that even-

tually finds a solution with the mentioned approximation guarantee, but is only known to converge in exponential time. As the configuration LP can be solved up to any precision in polynomial time, this means that we can approximate the value of an optimal solution within a factor $1/(4+\epsilon)$ for any $\epsilon > 0$ but it is not known how to efficiently find a solution with a comparable performance guarantee. Few other problems enjoy this intriguing status (see e.g. the overview article by Feige [10]). One of them is the restricted assignment problem[1], for which the second author in [14] developed the techniques from [1] to show that the configuration LP can be used to approximate the optimal makespan within a factor $33/17+\epsilon$ improving upon the 2-approximation by Lenstra, Shmoys & Tardos [13]. Again it is not known how to efficiently find a schedule of the mentioned approximation guarantee. However, these results indicate that an improved understanding of the configuration LP is likely to lead to improved approximation algorithms for these fundamental allocation problems.

In this paper we make progress that further substantiates this point. We modify the local search of [1] and present a novel analysis that allows us to significantly improve the bound on the running time from an exponential guarantee to a quasi-polynomial guarantee.

Theorem 1. *For any $\epsilon \in (0,1]$, we can find a $\frac{1}{4+\epsilon}$-approximate solution to restricted max-min fair allocation in time $n^{O\left(\frac{1}{\epsilon}\log n\right)}$, where $n = |\mathcal{P}| + |\mathcal{R}|$.*

In Section 3.1, we give an overview of the local search of [1] together with our modifications. The main modification is that at each point of the local search, we carefully select which step to take in the case of several options, whereas in the original description [1] an arbitrary choice was made. We then use this more stringent description with a novel analysis (Section 3.3) that uses the dual of the configuration LP as in [14]. The main advantage of our analysis (of the modified local search) is that it allows us to obtain a better upper bound on the search space of the local search and therefore also a better bound on the run-time. Furthermore, our techniques have the interesting property that although we use the rather complex configuration LP in the analysis, we never actually solve it. This gives hope to the interesting possibility of a polynomial time algorithm that is purely combinatorial and efficient to implement (in contrast to solving the configuration LP) with a good approximation ratio.

Finally, we note that our approach currently has a similar dependence on ϵ as in the case of solving the configuration LP since, as mentioned above, the linear program itself can only be solved approximately. However, our hidden constants are small and for a moderate ϵ we expect that our combinatorial approach is already more attractive than solving the configuration LP.

2 The Configuration LP

The intuition of the configuration linear program (LP) is that any allocation of value T needs to allocate a bundle or configuration C of resources to each player

[1] Also here the restricted version of the problem is the special case where $v_{ij} \in \{v_j, \infty\}$ (∞ instead of 0 since we are minimizing).

i so that $f_i(C) \geq T$. Let $\mathcal{C}(i,T)$ be the set of those configurations that have value at least T for player i. In other words, $\mathcal{C}(i,T)$ contains all those subsets of resources that are feasible to allocate to player i in an allocation of value T. For a guessed value of T, the configuration LP therefore has a decision variable $x_{i,C}$ for each player $i \in \mathcal{P}$ and configuration $C \in \mathcal{C}(i,T)$ with the intuition that this variable should take value one if and only if the corresponding set of resources is allocated to that player. The configuration LP $CLP(T)$ is a feasibility program and it is defined as follows:

$$\sum_{C \in \mathcal{C}(i,T)} x_{i,C} \geq 1 \qquad i \in \mathcal{P}$$

$$\sum_{i,C: j \in C, C \in \mathcal{C}(i,T)} x_{i,C} \leq 1 \qquad j \in \mathcal{R}$$

$$x \geq 0$$

The first set of constraints ensures that each player should receive at least one bundle and the second set of constraints ensures that a resource is assigned to at most one player.

If $CLP(T_0)$ for some T_0 is feasible, then $CLP(T)$ is also feasible for all $T \leq T_0$, because $\mathcal{C}(i,T_0) \subseteq \mathcal{C}(i,T)$ and thus a solution to $CLP(T_0)$ is a solution to $CLT(T)$ as well. Let T_{OPT} be the maximum of all such values. Every feasible allocation is a feasible solution of configuration LP, hence T_{OPT} is an upper bound on the value of the optimal allocation.

We note that the LP has exponentially many constraints; however, it is known that one can approximately solve it to any desired accuracy by designing a polynomial time (approximate) separation algorithm for the dual [3]. Although our approach does not require us to solve the linear program, the dual shall play an important role in our analysis. By associating a variable y_i with each constraint in the first set of constraints, a variable z_j with each constraint in the second set of constraints, and letting the primal have the objective function of minimizing the zero function, we obtain the dual program:

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

$$y_i \leq \sum_{j \in C} z_j \qquad i \in \mathcal{P}, C \in \mathcal{C}(i,T)$$

$$y, z \geq 0$$

3 Local Search with Better Run-Time Analysis

In this section we modify the algorithm by Asadpour et al. [1] in order to significantly improve the run-time analysis: we obtain a $1/(4+\epsilon)$-approximate solution in run-time bounded by $n^{O(1/\epsilon \log n)}$ whereas the original local search is only known to converge in time $2^{O(n)}$. For better comparison, we can write $n^{O(1/\epsilon \log n)} = 2^{O(1/\epsilon \log^2 n)}$. Moreover, our modification has the nice side effect that we actually never solve the complex configuration LP — we only use it in the analysis.

3.1 Description of Algorithm

Throughout this section we assume that T — the guessed optimal value — is such that $CLP(T)$ is feasible. We shall find an $1/\alpha$ approximation where α is a parameter such that $\alpha > 4$. As we will see, the selection of α has the following trade-off: the closer α is to 4 the worse bound on the run-time we get.

We note that if $CLP(T)$ is not feasible and thus T is more than T_{OPT}, our algorithm makes no guarantees. It might fail to find an allocation, which means that $T > T_{OPT}$. We can use this for a standard binary search on the interval $[0, \frac{1}{|\mathcal{P}|}\sum_i v_i]$ so that in the end we find an allocation with a value at least T_{OPT}/α.

Max-min Fair Allocation Is a Bipartite Hypergraph Problem. Similar to [1], we view the max-min fair allocation problem as a matching problem in the bipartite hypergraph $G = (\mathcal{P}, \mathcal{R}, E)$. Graph G has an hyperedge $\{i\} \cup C$ for each player $i \in \mathcal{P}$ and configuration $C \subseteq \mathcal{R}$ that is feasible with respect to the desired approximation ratio $1/\alpha$, i.e., $f_i(C) \geq T/\alpha$, and minimal in the sense that $f_i(C') < T/\alpha$ for all $C' \subset C$. Note that the graph might have exponentially many edges and the algorithm therefore never keeps an explicit representation of all edges.

From the construction of the graph it is clear that a matching covering all players corresponds to a solution with value at least T/α. Indeed, given such a matching M in this graph, we can assign matched resources to the players and everyone gets resources with total value of at least T/α.

Alternating Tree of "Add" and "Block" Edges. The algorithm of Asadpour et al. [1] can be viewed as follows. In the beginning we start with an empty matching and then we increase its size in every iteration by one, until all players are matched. In every iteration we build an alternating tree rooted in a currently unmatched player p_0 in the attempt to find an alternating path to extend our current matching M. The alternating tree has two types of edges: edges in the set A that we wish to *add* to the matching and edges in the set B that are currently in the matching but intersect edges in A and therefore *block* them from being added to the matching. While we are building the alternating tree to find an alternating path, it is important to be careful in the selection of edges, so as

to guarantee eventual termination. As in [1], we therefore define the concept of addable and blocking edges.

Before giving these definitions, it will be convenient to introduce the following notation. For a set of edges F, we denote by $F_{\mathcal{R}}$ all resources contained in edges in F and similarly $F_{\mathcal{P}}$ denotes all players contained in edges in F. We also write $e_{\mathcal{R}}$ instead of $\{e\}_{\mathcal{R}}$ for an edge e and use $e_{\mathcal{P}}$ to denote the player in e.

Definition 1. *We call an edge e addable if $e_{\mathcal{R}} \cap (A_{\mathcal{R}} \cup B_{\mathcal{R}}) = \emptyset$ and $e_{\mathcal{P}} \in \{p_0\} \cup A_{\mathcal{P}} \cup B_{\mathcal{P}}$.*

Definition 2. *An edge b in the matching M is blocking e if $e_{\mathcal{R}} \cap b_{\mathcal{R}} \neq \emptyset$.*

Note that an addable edge matches a player in the tree with resources that currently do not belong to any edge in the tree and that the edges blocking an edge e are exactly those in the matching that prevent us from adding e. For a more intuitive understanding of these concepts see Figure 1 in Section 3.2.

The idea of building an alternating tree is similar to standard matching algorithms using augmenting paths. However, one key difference is that the matching can be extended once an alternating path is found in the graph case, whereas the situation is more complex in the considered hypergraph case, since a single hyperedge might overlap several hyperedges in the matching. It is due to this complexity that it is more difficult to bound the running time of the hypergraph matching algorithm of [1] and our improved running time is obtained by analyzing a modified version where we carefully select in which order the edges should be added to the alternating tree and drop edges from the tree beyond certain distance.

We divide resources into 2 groups. *Fat resources* have value at least T/α and *thin resources* have less than T/α. Thus any edge containing a fat resource contains only one resource and is called *fat edge*. Edges containing thin resources are called *thin edges*. Our algorithm always selects an addable edge of minimum distance to the root p_0 according to the following convention. The length of a thin edge in the tree is one and the length of a fat edge in the tree is zero. Edges not in the tree have infinite length. Hence, the *distance of a vertex* from the root is the number of thin edges between the vertex and the root and, similarly, the *distance of an edge e* is the number of thin edges on the way to e from p_0 including e itself. We also need to refer to distance of an addable edge that is not yet in the tree. In that case we take the distance as if it was in the tree. Finally, by the *height of the alternating tree* we refer to the maximum distance of a resource from the root.

Algorithm for Extending a Partial Matching. Algorithm 1 summarizes the modified procedure for increasing the size of a given matching by also matching a previously unmatched player p_0. For better understanding of the algorithm, we included an example of an algorithm execution in Figure 1 in Section 3.2.

Input : A partial matching M
Output: A matching of increased size assuming that T is at most T_{OPT}

Find an unmatched player $p_0 \in \mathcal{P}$, make it a root of the alternating tree
while *there is an addable edge within distance* $2\log_{(\alpha-1)/3}(|\mathcal{P}|)+1$ **do**
  Find an addable edge e of minimum distance from the root
  $A \leftarrow A \cup \{e\}$
  if *e has blocking edges $b_1, \ldots, b_k$* **then**
    $B \leftarrow B \cup \{b_1, \ldots, b_k\}$
  else// `collapse procedure`
    while *e has no blocking edges* **do**
      if *there is an edge $e' \in B$ such that $e'_{\mathcal{P}} = e_{\mathcal{P}}$* **then**
        $M \leftarrow M \setminus \{e'\} \cup \{e\}$
        $A \leftarrow A \setminus \{e\}$
        $B \leftarrow B \setminus \{e'\}$
        Let $e'' \in A$ be the edge that e' was blocking
        $e \leftarrow e''$
      else
        $M \leftarrow M \cup \{e\}$
        return M
      end if
    end while
    Drop from A and B all edges of greater or the same distance as e
  end if
end while
return T_{OPT} is less than T

Algorithm 1. Increase the size of the matching

Note that the algorithm iteratively tries to find addable edges of minimum distance to the root. On the one hand, if the picked edge e has blocking edges that prevents it from being added to the matching, then the blocking edges are added to the alternating tree and the algorithm repeatedly tries to find addable edges so as to make progress by removing the blocking edges.

On the other hand, if edge e has no blocking edges, then this means that the set of resources $e_{\mathcal{R}}$ is free, so we make progress by adding e to the matching M. If the player was not previously matched, it is the root p_0 and we increased the size of the matching. Otherwise the player $e_{\mathcal{P}}$ was previously matched by an edge $e' \in B$ such that $e'_{\mathcal{P}} = e_{\mathcal{P}}$, so we remove e' from M and thus it is not a blocker anymore and can be removed from B. This removal has decreased the number of blockers for an edge $e'' \in A$. If e'' has 0 blockers, we recurse and repeat the same procedure as with e. Note that this situation can be seen on Figure 1(b) and 1(c) in Section 3.2.

3.2 Example of Algorithm Execution

Figure 1 is a visualization of an execution of Algorithm 1. The right part of every picture is the alternating tree and to the left we display the positions of edges in the tree in the bipartite graph. Gray edges are A-edges and white are B-edges.

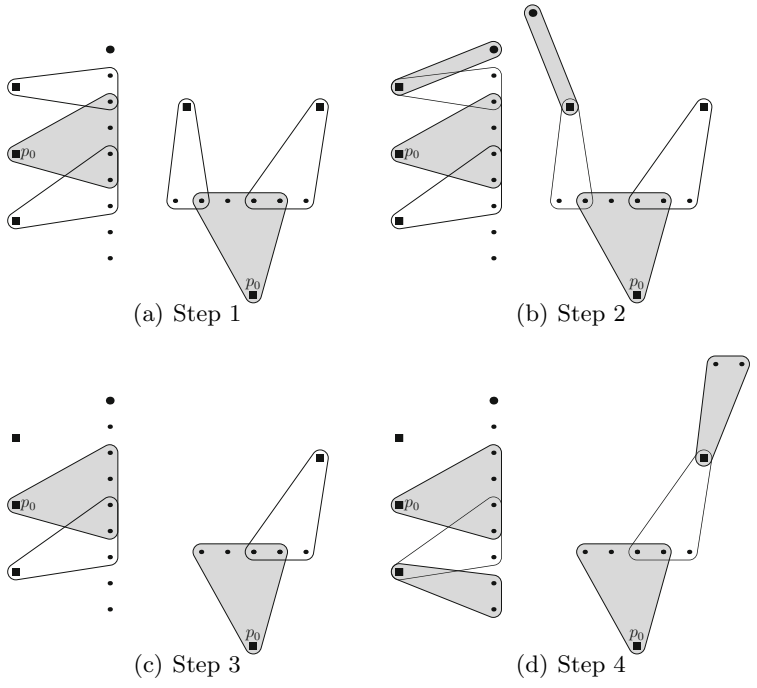

Fig. 1. Alternating tree visualization. The right part of every picture is the alternating tree and to the left we display the positions of edges in the tree in the bipartite graph. Gray edges are in the set A and white edges are in the set B.

In Figure 1(a) we start by adding an A-edge to the tree. There are 2 edges in the matching intersecting this edge, so we add them as blocking edges. Then in Figure 1(b) we add a fat edge that has no blockers, so we add it to the matching and thus remove one blocking edge, as we can see in Figure 1(c). Then in Figure 1(d) we add a thin edge which has no blockers. Now the A and B edges form an alternating path, so by swapping them we increase the size of the matching and the algorithm terminates.

Note that the fat edge in step 2 is added before the thin edge from step 4, because it has shorter distance from the root p_0. Recall that the distance of an edge e is the number of thin edges between e and the root including e, thus the distance of the fat edge is 2 and the distance of the thin edge is 3.

3.3 Analysis of Algorithm

Let the parameter α of the algorithm equal $4 + \epsilon$ for some $\epsilon \in (0, 1]$. We first prove that Algorithm 1 terminates in time $n^{O\left(\frac{1}{\epsilon}\log n\right)}$ where $n = |\mathcal{P}| + |\mathcal{R}|$ and, in the following subsection, we show that it returns a matching of increased size if $CLP(T)$ is feasible.

Theorem 1 then follows from that, for each guessed value of T, Algorithm 1 is at most invoked n times and we can find the maximum value T for which our algorithm finds an allocation by binary search on the interval $[0, \frac{1}{|\mathcal{P}|}\sum_i v_i]$. Since we can assume that the numbers in the input have bounded precision, the binary search only adds a polynomial factor to the running time.

Run-Time Analysis. We bound the running time of Algorithm 1 using that the alternating tree has height at most $O(\log_{(\alpha-1)/3}|\mathcal{P}|) = O\left(\frac{1}{\epsilon}\log|\mathcal{P}|\right)$. The proof is similar to the termination proof in [1] in the sense that we associate a signature vector with each tree and then show that its lexicographic value decreases. However, one key difference is that instead of associating a value with *each edge* of type A in the tree, we associate a value with each "layer" that consists of *all edges* of a certain distance from the root. This allows us to directly associate the run-time with the height of the alternating tree.

Lemma 1. *For a desired approximation guarantee of* $1/\alpha = 1/(4+\epsilon)$*, Algorithm 1 terminates in time* $n^{O\left(\frac{1}{\epsilon}\log n\right)}$.

Proof. We analyze the run-time of Algorithm 1 by associating a signature vector with the alternating tree of each iteration. When considering an alternating tree it is convenient to partition A and B into $A_0, A_1, \ldots, A_{2k}$ and $B_0, B_1, \ldots, B_{2k}$ respectively by the distance from the root, where $2k$ is the maximum distance of an edge in the alternating tree (it is always an even number). The signature vector of an alternating tree is then defined to be

$$(-|A_0|, |B_0|, -|A_1|, |B_1|, \ldots, -|A_{2k}|, |B_{2k}|, \infty).$$

We prove that each addition of an edge decreases the lexicographic value of the signature or increases the size of the matching.

On the one hand, if we add an edge with no blocking edges, we either completely collapse the alternating tree or collapse only a part of it and change the signature to $(-|A_0|, |B_0|, \ldots, -|A_{2\ell}|, |B_{2\ell}| - 1, \infty)$ for some $0 \leq \ell \leq k$ as the algorithm drops all edges farther away or in the same distance from the root as the edge last added to the matching. Thus we either increase the size of the matching or decrease the signature of the alternating tree.

On the other hand, if the added edge e has blocking edges, there are two cases. We either open new layers $A_{2k+1} = \{e\}$ and B_{2k+2} where e is a thin edge and the signature gets smaller, since $-|A_{2k+1}| < \infty$. If we do not open a new layer, we increase the size of some A_ℓ and $-(|A_\ell|+1) < -|A_\ell|$, so in this case the signature decreases too.

The algorithm only runs as long as the height of the alternating tree is at most $O(\log_{(\alpha-1)/3}|\mathcal{P}|) = O(\log_{1+\epsilon/3}|\mathcal{P}|)$. This can be rewritten as $O\left(\frac{\log|\mathcal{P}|}{\log(1+\epsilon/3)}\right) = O\left(\frac{\log|\mathcal{P}|}{\epsilon}\right)$ where the equality follows from $x \leq 2\log(1+x)$ for $x \in (0,1]$ and we only consider $\epsilon \in (0,1]$. There are at most $|\mathcal{P}|$ possible values for each position in a signature, so the total number of signatures encountered during the execution

of Algorithm 1 is $|\mathcal{P}|^{O\left(\frac{1}{\epsilon}\log|\mathcal{P}|\right)}$. As adding an edge happens in polynomial time in $n = |\mathcal{P}|+|\mathcal{R}|$, we conclude that Algorithm 1 terminates in time $n^{O\left(\frac{1}{\epsilon}\log n\right)}$. □

Correctness of Algorithm 1. We show that Algorithm 1 is correct, i.e., that it returns an increased matching if $CLP(T)$ is feasible.

We have already proved that the algorithm terminates in Lemma 1. The statement therefore follows from proving that the condition of the while loop always is satisfied assuming that the configuration LP is feasible. In other words, we will prove that there always is an addable edge within the required distance from the root. This strengthens the analogous statement of [1] that states that there always is an addable edge (but without the restriction on the search space that is crucial for our run-time analysis). We shall do so by proving that the number of thin blocking edges increases quickly with respect to the height of the alternating tree and, as there cannot be more than $|\mathcal{P}|$ blocking edges, this in turn bounds the height of the tree.

For this purpose, let us again partition A and B into $A_0, A_1, \ldots, A_{2k}$ and $B_0, B_1, \ldots, B_{2k}$ respectively by the distance from the root. Note that B_i is empty for all odd i. Also, A_{2i} contains only fat edges and A_{2i+1} only thin edges. For a set of edges F denote by F^t all the thin edges in F and by F^f all the fat edges in F. We also use $\mathcal{R}^t$ to denote thin resources and $\mathcal{R}^f$ to denote fat resources.

We are now ready to state the key insight behind the analysis that shows that the number of blocking edges increases as a function of α and the height of the alternating tree.

Lemma 2. *Let $\alpha > 4$. Assuming that $CLP(T)$ is feasible, if there is no addable edge e within distance $2D+1$ from the root for some integer D, then*

$$\frac{\alpha - 4}{3} \sum_{i=1}^{D} |B_{2i}^t| < |B_{2D+2}^t|.$$

Before giving the proof of Lemma 2, let us see how it implies that there always is an addable edge within distance $2\log_{(\alpha-1)/3}(|\mathcal{P}|)+1$ from the root assuming the configuration LP is feasible, which in turn implies the correctness of Algorithm 1.

Corollary 1. *If $\alpha > 4$ and $CLP(T)$ is feasible, then there is always an addable edge within distance $2D+1$ from the root, where $D = \log_{(\alpha-1)/3}|\mathcal{P}|$.*

The proof of the corollary follows intuitively from that Lemma 2 says that the number of blocking edges increases exponentially in terms of the height of the tree and therefore, as there are at most $|\mathcal{P}|$ blocking edges, the height must be $O_\alpha(\log|\mathcal{P}|)$. The detailed proof can be found in the full version of this paper. We complete the correctness analysis of the algorithm by presenting a proof sketch of the key lemma.

Proof (Lemma 2). We give an overview of the proof of Lemma 2 (the complete proof can be found in the full version of this paper).

Suppose toward contradiction that there is no addable edge within distance $2D+1$ and

$$\frac{\alpha-4}{3}\sum_{i=1}^{D}|B_{2i}^t| \geq |B_{2D+2}^t|.$$

We show that this implies that the dual of the configuration LP is unbounded, which in turn contradicts the assumption that the primal is feasible. Recall that the objective function of the dual is $\max \sum_{i\in\mathcal{P}} y_i - \sum_{j\in\mathcal{R}} z_j$. Furthermore, as each solution (y,z) of the dual can be scaled by a scalar c to obtain a new solution $(c\cdot y, c\cdot z)$, any solution with positive objective implies unboundedness. We proceed by defining such solution (y^*,z^*), that is determined by the alternating tree. More precisely, we take

$$y_i^* = \begin{cases} \binom{\alpha-1}{\alpha} & \text{if } i\in\mathcal{P} \text{ is within distance } 2D \text{ from the root,} \\ 0 & \text{otherwise,} \end{cases}$$

and

$$z_j^* = \begin{cases} (\alpha-1)/\alpha & \text{if } j\in\mathcal{R} \text{ is fat and within distance } 2D \text{ from the root,} \\ v_j/T & \text{if } j\in\mathcal{R} \text{ is thin and within distance } 2D+2 \text{ from the root,} \\ 0 & \text{otherwise.} \end{cases}$$

It can be shown that (y^*,z^*) is a feasible solution provided that there is no addable edge within distance $2D+1$.

The proof is completed by showing that

$$\sum_{j\in\mathcal{R}} z_j \leq \binom{\alpha-1}{\alpha}\sum_{i=0}^{D}|B_{2i}^f| + \binom{\alpha-1}{\alpha}\sum_{i=1}^{D}|B_{2i}^t| < \binom{\alpha-1}{\alpha}\left(1+\sum_{i=0}^{D}|B_{2i}|\right) = \sum_{i\in\mathcal{P}} y_i,$$

so the dual is unbounded, which contradicts the assumption that the primal is feasible. □

4 Conclusions

Asadpour et al. [1] raised as an open question whether their local search (or a modified variant) can be shown to run in polynomial time. We made progress toward proving this statement by showing that a modified local search procedure finds a solution in quasi-polynomial time. Moreover, based on our findings, we conjecture the stronger statement that there is a local search algorithm that does not use the LP solution, i.e., it is combinatorial, and it finds a $1/(4+\epsilon)$-approximate solution in polynomial time for any fixed $\epsilon > 0$.

References

1. Asadpour, A., Feige, U., Saberi, A.: Santa claus meets hypergraph matchings. In: Proceedings of the 11th International Workshop and 12th International Workshop on Approximation, Randomization and Combinatorial Optimization, pp. 10–20 (2008); see authors' homepages for the lower bound of 1/4 instead of the claimed 1/5 in the conference version

2. Asadpour, A., Saberi, A.: An approximation algorithm for max-min fair allocation of indivisible goods. In: Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing, STOC 2007, pp. 114–121. ACM, New York (2007)
3. Bansal, N., Sviridenko, M.: The santa claus problem. In: Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing, STOC 2006, pp. 31–40. ACM Press, New York (2006)
4. Bateni, M., Charikar, M., Guruswami, V.: Maxmin allocation via degree lower-bounded arborescences. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, pp. 543–552. ACM, New York (2009)
5. Bezáková, I., Dani, V.: Allocating indivisible goods. SIGecom. Exch. 5, 11–18 (2005)
6. Chakrabarty, D., Chuzhoy, J., Khanna, S.: On allocating goods to maximize fairness. In: Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science FOCS 2009, pp. 107–116. IEEE Computer Society Press, Washington, DC (2009)
7. Dobzinski, S., Schapira, M.: An improved approximation algorithm for combinatorial auctions with submodular bidders. In: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm, SODA 2006, pp. 1064–1073. ACM, New York (2006)
8. Feige, U.: On maximizing welfare when utility functions are subadditive. In: Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing, STOC 2006, pp. 41–50. ACM, New York (2006)
9. Feige, U.: On allocations that maximize fairness. In: Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, pp. 287–293. Society for Industrial and Applied Mathematics, Philadelphia (2008)
10. Feige, U.: On estimation algorithms vs approximation algorithms. In: Hariharan, R., Mukund, M., Vinay, V. (eds.) IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2008). Leibniz International Proceedings in Informatics (LIPIcs), vol. 2, pp. 357–363. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2008)
11. Feige, U., Vondrak, J.: Approximation algorithms for allocation problems: Improving the factor of 1 - 1/e. In: 47th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2006, pp. 667–676 (October 2006)
12. Haeupler, B., Saha, B., Srinivasan, A.: New constructive aspects of the lovasz local lemma. In: 2010 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 397–406 (October 2010)
13. Lenstra, J.K., Shmoys, D.B., Tardos, É.: Approximation algorithms for scheduling unrelated parallel machines. Math. Program. 46, 259–271 (1990)
14. Svensson, O.: Santa claus schedules jobs on unrelated machines. In: Proceedings of the 43rd Annual ACM Symposium on Theory of Computing, STOC 2011, pp. 617–626. ACM, New York (2011)
15. Vondrak, J.: Optimal approximation for the submodular welfare problem in the value oracle model. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, STOC 2008, pp. 67–74. ACM, New York (2008)