

APPROXIMATING THE NASH SOCIAL WELFARE WITH INDIVISIBLE ITEMS*

RICHARD COLE[†] AND VASILIS GKATZELIS[‡]

Abstract. We study the problem of allocating a set of indivisible items among agents with additive valuations, with the goal of maximizing the geometric mean of the agents' valuations, i.e., the Nash social welfare. This problem is known to be NP-hard, and our main result is the first efficient constant-factor approximation algorithm for this objective. We first observe that the integrality gap of the natural fractional relaxation is exponential, so we propose a different fractional allocation which implies a tighter upper bound and, after appropriate rounding, yields a good integral allocation.

An interesting contribution of this work is the fractional allocation that we use. The relaxation of our problem can be solved efficiently using the Eisenberg-Gale program, whose optimal solution can be interpreted as a market equilibrium with the dual variables playing the role of item prices. Using this market-based interpretation, we define an alternative equilibrium allocation where the amount of spending that can go into any given item is bounded, thus keeping the highly priced items under-allocated, and forcing the agents to spend on lower priced items. The resulting equilibrium prices reveal more information regarding how to assign items so as to obtain a good integral allocation.



Key words. Nash Social Welfare, Competitive Equilibrium from Equal Incomes, Approximation Algorithms, Fair Division

AMS subject classifications. 68Q25, 91A99

1. Introduction. We study the problem of allocating a collection of m indivisible items among a set of n agents ($n \leq m$), aiming to maximize the Nash social welfare (NSW). Since the items are indivisible, an allocation x assigns each item to a single agent. We assume that the agents have additive valuations, i.e., each agent i has a non-negative value v_{ij} for each item j , and her value for an allocation x that assigns to her some bundle of items B_i , is $v_i(x) = \sum_{j \in B_i} v_{ij}$. The NSW objective is to compute an allocation maximizing the geometric mean of the agents' values, i.e.,

$$\max_x \left(\prod_i v_i(x) \right)^{1/n}.$$

The motivation for this problem is closely related to the well-studied *Santa Claus problem* [9, 6, 4, 21, 8, 13, 5, 3], where the objective is to compute an allocation that maximizes the *minimum value* across all agents, i.e., $\max_x \min_i v_i(x)$. The story behind the Santa Claus problem is that Santa is carrying presents (the items) which will be given to children (the agents) and his goal is to allocate the presents in a way which ensures that the least happy child is as happy as possible. As we discuss later on, the geometric mean objective, just like the max-min objective, aims to reach a balanced distribution of value, so both these problems belong to the long literature on fair division.

Social Choice Theory. Allocating resources among a set of agents in a fair manner is one of the fundamental goals in economics, and, in particular, social choice theory. Before embarking on computing fair allocations, one first needs to ask what is the right objective for fairness. This question alone has been the subject of long debates

*A preliminary version of this paper appeared at the 47th ACM Symposium on Theory of Computing (STOC 2015).


[†]Courant Institute, New York University, cole@cims.nyu.edu

[‡]Computer Science, Stanford University, gkatz@cs.stanford.edu

in both social science and game theory, leading to a very rich literature. At the time of writing this paper, there are at least *five* academic books [40, 11, 37, 27, 7] written on the topic of fair division, providing an overview of various proposed solutions for fairness.

Over the last decade, the computer science and operations research communities have contributed to this literature, mostly focusing on the tractability of computing allocations that maximize specific fairness objectives. The result of this work has been a deeper understanding of the extent to which some of these objectives can be optimized or approximated in polynomial time which, in turn, serves as a signal regarding the appropriateness and applicability of these objectives.

Nash Social Welfare. The objective we seek to maximize in this work, the Nash social welfare (also known as Bernouli-Nash social welfare), dates back to the fifties [28, 23], when it was proposed by Nash as a solution for bargaining problems, using an axiomatic approach. This objective, like other standard welfare objectives, is captured by the following family of functions known as generalized means, or power means:

$$M_p(x) = \left(\frac{1}{n} \sum_i [v_i(x)]^p \right)^{1/p} .$$


In particular, the NSW corresponds to $M_0(x)$, i.e., the limit of $M_p(x)$ as p goes to zero. Other important examples of welfare functions captured by $M_p(x)$ include the max-min objective that the Santa Claus problem studies, known as *egalitarian social welfare*, as well as the *utilitarian social welfare*, which maximizes the average value across the agents. The former corresponds to the limit of $M_p(x)$ as p goes to $-\infty$, while the latter corresponds to $M_1(x)$. Two of the most notable properties that the NSW objective satisfies follow. (Additional appealing properties are discussed by Moulin [27].)

- The optimal allocation with respect to the NSW objective is scale-free, i.e., it is independent of the scale of each agent's valuations.
- The NSW objective provides a natural compromise between fairness and efficiency.

The fact that the NSW objective is scale-free means that choosing the desired allocation does not require interpersonal comparability of the individual's preferences. The agents only need to report relative valuations, i.e., how much more they like an item compared to another. In other words, maximizing the NSW using the v_{ij} values is equivalent to using $\alpha_i v_{ij}$ as values instead, where $\alpha_i > 0$ is some constant for each agent i . This property is particularly useful in settings where the agents are not paying for the items they are allocated, in which case the scale in which their valuations are expressed may not have any real meaning¹. Note that neither the egalitarian nor the utilitarian social welfare objectives are scale-free. As a result, the former could end up allocating most of the items to an agent who has a low value for all the items, while the latter could end up allocating all the items to just one agent.

Regarding the second property, the two alternative welfare objectives mentioned above, i.e., the egalitarian and the utilitarian objectives, correspond to extreme fairness and efficiency considerations respectively. The former objective maximizes the happiness of the least satisfied agent, irrespective of how much inefficiency this might be causing, and, on the other extreme, the utilitarian social welfare approach maximizes efficiency while disregarding how unsatisfied some agents might become. The

¹If the agents were paying for the items they are allocated, then v_{ij} could be interpreted as the amount of money that agent i is willing to pay for item j .

NSW objective lies between these two extremes and strikes a natural balance between them, since maximizing the geometric mean leads to more balanced valuations, but without neglecting efficiency.

Applications. A strong signal regarding the importance and the usefulness of this objective is the fact that it has been independently discovered and used in different communities. Maximizing the geometric mean, or equivalently the sum of the logarithms², of the agents' valuations yields what is known in economics as the *competitive equilibrium from equal incomes (CEEI)* [22, 39], and what is known as *proportional fairness (PF)* [24] in the TCP congestion control literature.

In particular, the CEEI is the market equilibrium that would arise if every agent was allocated the same budget of some artificial currency, a testament to the fairness and efficiency properties of this objective. In fact, since this equilibrium is not guaranteed to exist when the items are indivisible, recent work has proposed an approximate-CEEI solution, which is being used in practice for allocating seats in classes to business school students [12]. On the other hand, proportional fairness is the de facto solution for bandwidth sharing in the networking community, and the most widely implemented solution in practice (for instance see [2]).

Computational Complexity. The computational complexity of the Nash social welfare has been studied for various types of valuation functions [36, 18] (see [30] for a survey of the known results). For the types of valuations that we consider here, i.e., for additive valuations, this problem is known to be NP-hard [38, 29]. Nguyen and Rothe [31] studied the approximability of the Nash social welfare objective, which led to the first approximation algorithm. In particular, they show that their algorithm approximates the geometric mean objective within a factor of $(m - n + 1)$. In this paper we provide the first polynomial time constant factor approximation algorithm for this objective. Following up on an earlier version of this paper [15], this problem was also shown to be APX-hard [25].

1.1. Other Related Work. There has been a lot of work on algorithms that approximate a fair allocation of indivisible items. For the max-min objective of the Santa Claus problem, which is also known to be NP-hard, Bansal and Sviridenko [6], and Asadpour and Saberi [4] proposed the first approximation algorithms. These algorithms are derived by rounding a linear programming relaxation of the problem. Another fairness measure that has received a lot of attention is envy-freeness. Unlike the fairness measures that we mentioned above, envy-freeness is a property rather than an objective, so it is either satisfied by the outcome or not. In particular, an allocation x is envy-free if no agent prefers to swap her bundle of items in x with another agent. Note that, when the items are indivisible, such an allocation might not exist at all. For example, if every agent strongly prefers the same item, only one of them can receive it, so the other agents are bound to envy her. Lipton et al. [26] defined an approximate version of this property, thus turning it into an objective as well. Among other results, they provide a polynomial time algorithm that computes approximately envy-free allocations. This approach was further generalized by Chevaleyre et al. [14] who proposed a framework for defining the “degree of envy” of an allocation. Bouveret and Lemaître [10] also focus on the allocation of indivisible items and they study a hierarchy of fairness properties in this setting. Other common notions of

²As we discuss later, maximizing $(\prod_i v_i(x))^{1/n}$ is equivalent to maximizing $\sum_i \log(v_i(x))$. This can be verified by taking a logarithmic transformation of the former objective.

fairness that have been studied in the fair allocation literature are, proportionality³, and equitability [40, 11, 37, 27, 7].

Recent work has also suggested some new solutions for fairness. Budish [12] showed that, although the CEEI outcome might not exist for indivisible items, there always exists some allocation that is an approximate equilibrium, though computing such an allocation was recently shown to be hard by Othman et al. [34]. Also, Bouveret and Lemaître [10] very recently introduced the min-max fair-share benchmark for fairness which has a max-min flavor. For indivisible items, there might not exist an optimal allocation with respect to this objective, but, for additive valuations, subsequent work from Procaccia and Wang [35] and Amanatidis et al. [1] provided algorithms that are guaranteed to approximate this benchmark on every instance.

Our setting is also closely related to the large topic of cake-cutting [40, 11, 37, 27, 7], which has been studied since the 1940's. This literature uses the $[0, 1]$ interval as the standard representation of a cake, which can be thought of as a collection of infinitely divisible items. When the items are divisible, computing the fractional allocation that maximizes the NSW objective is a special case of the Fisher market equilibrium with affine utility buyers; the latter problem was solved in (weakly) polynomial time by Devanur et al. [19]; Orlin later suggested a strongly polynomial time algorithm [33]. Finally, when the items are divisible but the valuations are private information, Cole et al. [17, 16] propose mechanisms for approximating the NSW objective while ensuring that the agents will report their values truthfully.

1.2. Our Results. In this work we study the NP-hard combinatorial optimization problem of allocating a set of indivisible items among agents with additive valuations, aiming to maximize the Nash social welfare. Our main result is the first polynomial time algorithm that guarantees a constant factor approximation of the geometric mean of the agents' valuations. In particular, we prove that our algorithm achieves an approximation factor of at most $2 \cdot e^{1/e} \approx 2.889$.

We first observe that, although our objective is not convex, we can solve the natural fractional relaxation of the problem optimally using the Eisenberg-Gale convex program, but the integrality gap of this relaxation grows with the size of the problem. To circumvent the integrality gap, we leverage the fact that the solution of the Eisenberg-Gale program can be interpreted as the equilibrium of a market where the agents pay in order to buy fractions of the items.

Motivated by this market-based interpretation of the fractional solution, we propose a new type of market equilibrium. In particular, we introduce a constraint that restricts the total amount of money that the agents can spend on any given item. The induced "spending-restricted" equilibrium may motivate some agents to avoid the highly-demanded items and to spend on lower-priced items instead. As a result, the fractional allocation of this equilibrium uncovers useful information regarding how the less desired items should be allocated, and our rounding algorithm uses this information in order to compute a good integral allocation. Apart from serving as a guide toward an integral solution, this fractional allocation also implies an upper bound for the optimal integral solution that approximates it very closely, and thus allows us to prove the constant factor guarantee.

An interesting fact about the spending constraint that we introduce is that it involves a combination of both the primal and the dual variables of the Eisenberg-Gale

³It is worth distinguishing the notion of proportional fairness from that of proportionality by noting that the latter is a much weaker notion, directly implied by the former.

program. As a result, to compute this solution we borrow ideas from the combinatorial algorithms for solving the Eisenberg-Gale program. We present both a simple weakly-polynomial time algorithm, and a more elaborate strongly-polynomial one.

2. Preliminaries. Given a set M of m items and a set N of n agents ($n \leq m$) with additive valuations, our goal is to compute an integral allocation of items to agents aiming to optimize the geometric mean of the agents' valuations. This problem can be expressed as the following integer program, IP:

$$\begin{aligned} \text{maximize: } & \left(\prod_{i \in N} u_i \right)^{1/n} \\ \text{subject to: } & \sum_{j \in M} x_{ij} v_{ij} = u_i, \quad \forall i \in N \\ & \sum_{i \in N} x_{ij} = 1, \quad \forall j \in M \\ & x_{ij} \in \{0, 1\}, \quad \forall i \in N, j \in M \end{aligned}$$

We observe that, for any fixed number of agents n , maximizing the geometric mean of their valuations is equivalent to maximizing the sum of the logarithms of their valuations⁴. As a result, the Nash social welfare maximization problem can be expressed as a convex integer program. In fact, the fractional relaxation of this convex program is an instance of the very well studied Eisenberg-Gale program [20]:

$$\begin{aligned} \text{maximize: } & \sum_{i \in N} \log u_i \\ \text{subject to: } & \sum_{j \in M} x_{ij} v_{ij} = u_i, \quad \forall i \in N \\ & \sum_{i \in N} x_{ij} \leq 1, \quad \forall j \in M \\ & x_{ij} \geq 0, \quad \forall i \in N, j \in M \end{aligned}$$

One important implication of this observation is the fact that we can compute the optimal solution for the fractional relaxation of IP in polynomial time. In fact, thanks to recent work on the Eisenberg-Gale program, this solution can be computed using combinatorial algorithms [19, 33]. **Another very important implication, which we use to design our approximation algorithm, is that this fractional solution can be interpreted as the equilibrium allocation for the linear case of Fisher's market model [32, Chapter 5].** In this model, each agent has a certain budget, and she is using this budget in order to buy fractions of the available items. **Although the agents in our setting are *not* using money,** this market-based interpretation of the optimal solution of IP's fractional relaxation will provide some very useful intuition. All of the technical sections of this paper are described using this market-based interpretation, so we spend part of this section clarifying the connection.

Market-Based Interpretation. In the Fisher market corresponding to our problem the items are divisible and the valuation of agent i who is receiving a fraction $x_{ij} \in [0, 1]$ of each item j is $\sum_{j \in M} x_{ij} v_{ij}$. Each agent has the same budget of, say, \$1 to

⁴To verify this fact, one can apply a logarithmic transformation to the initial objective.

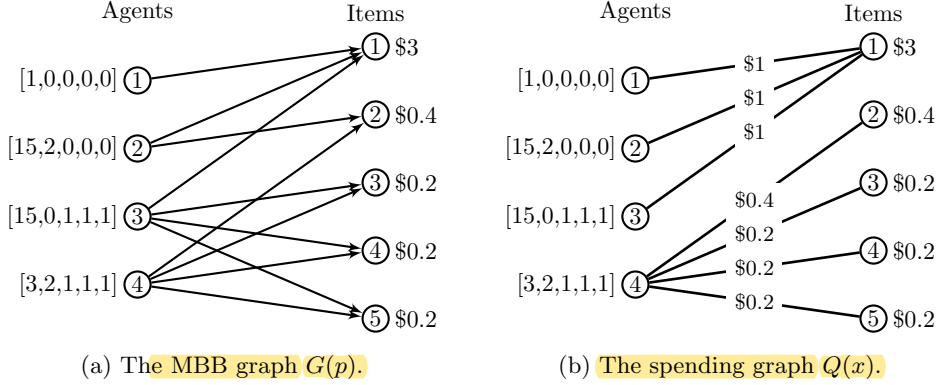


Fig. 1: The MBB graph and the spending graph for the instance of Example 2.1.

spend on items and each item j has a price p_j . If agent i spends b_{ij} on an item whose price is p_j , then she receives a fraction $x_{ij} = b_{ij}/p_j$ of that item (there is one unit of each item, so $\sum_{i \in N} b_{ij} \leq p_j$). A vector of item prices $p = (p_1, \dots, p_m)$ induces a market equilibrium if every agent is spending all of her budget on her “optimal” items given these prices, and the market clears, i.e., all of the items are allocated fully. To be more precise, the “optimal” items for agent i , given prices p , are the ones that maximize the ratio v_{ij}/p_j , also known as the *maximum bang per buck* (MBB) items.

The allocation of items to agents in the market equilibrium corresponds to the primal variables of the Eisenberg-Gale program (x_{ij}) and the prices of the market correspond to its dual variables (p_j). A closer look at the market equilibrium conditions reveals that they are closely related to the KKT conditions for the Eisenberg-Gale program [32, Chapter 5].

- Every item $j \in M$ is allocated fully: $\sum_{i \in N} x_{ij} = 1$.
- Every agent spends all of her budget: $\sum_{j \in M} x_{ij} p_j = 1$.
- Every agent spends her budget only on her MBB items: If $x_{ij} > 0$, then $j \in \arg \max_{j' \in M} \{v_{ij'}/p_{j'}\}$.

The MBB Graph and the Spending Graph. In order to describe our algorithms and their outcomes, we will be using bipartite graphs whose vertices correspond to the set of agents on one side, and items on the other.

For instance, given prices p , the *MBB graph* is a directed bipartite graph $G(p)$ with an edge (i, j) between agent i and item j if and only if j is an MBB item of i at prices p . Also, given some allocation x , the *spending graph* $Q(x)$ is an undirected bipartite graph with an edge (i, j) between agent i and item j if and only if $x_{ij} > 0$. In all the allocations that we consider, agents are allocated fractions only of their MBB items. The spending graph edges will therefore be a subset of the MBB edges.

Given prices p , there may be multiple equally valuable ways for each agent to distribute her budget across MBB items. Hence, there may be multiple spending graphs with the same Nash social welfare. In fact, one can always rearrange the spending of the agents so as to ensure that the induced spending graph is a forest of trees, without affecting those agents’ valuations (e.g., see [33]). Therefore, throughout the paper we assume that the spending graph is always a forest of trees.

EXAMPLE 2.1. *For a concrete example, consider the problem instance of Figure 1a, which comprises 4 agents (the vertices on the left) and 5 items (the vertices on the right). The valuations of the agents appear on the left of each agent's vertex, so Agent 1 values only Item 1, whereas Agent 2 values this item 7.5 times more than Item 2, and has no value for the other items. The prices that appear on the right of each item's vertex correspond to their prices in the corresponding market equilibrium, and the directed graph of Figure 1a is the MBB graph $G(p)$ at these prices. Note that the sum of these equilibrium prices is equal to the number of agents, which is no coincidence, since it has to be equal to the overall budget.*

The optimal solution of the fractional relaxation of IP for this problem can be seen in the form of a spending graph in Figure 1b. Note that every agent is spending all of her budget of \$1 on items that are MBB for her at the given prices. Also, the total spending going into any item is exactly equal to its price, so all of the items are allocated fully. According to the spending graph, Agents 1, 2, and 3 each get a $1/3$ fraction of Item 1, and Agent 4 gets all of the other items.

Approximation Guarantee. Let x^* denote the integral allocation that maximizes the Nash social welfare. Our goal is to design an efficient algorithm that computes an integral allocation x which is always within a factor ρ of the optimal one, i.e.,

$$\left(\prod_{i \in N} v_i(x^*) \right)^{1/n} \leq \rho \cdot \left(\prod_{i \in N} v_i(x) \right)^{1/n},$$

where $\rho \geq 1$, is as small as possible. The best previously known approximation guarantee was $\rho \in O(m)$ [31]; in this work we provide an algorithm that guarantees a factor of at most $\rho \approx 2.889$.

In terms of inapproximability statements, some previous work considered the problem of approximating the product of the valuations instead of the geometric mean, showing that this problem is APX-hard [29]. In fact, we note that we cannot hope for a reasonable approximation of the product of the valuations. If an algorithm achieves an approximation factor of $f(n)$ for instances with n agents, then copying a worst-case instance κ times yields a new instance with κn agents and approximation $f(\kappa n) = f(n)^\kappa$. Since the problem is NP-hard, $f(n) > 1$, which implies that the approximation factor grows exponentially with n . The most recent inapproximability result shows that optimizing the geometric mean is APX-hard as well [25].

3. Approximation Algorithm.

3.1. Integrality Gap. Since we can compute the fractional relaxation of the IP program using the Eisenberg-Gale program, a standard technique for designing an approximation algorithm would be to take the fractional allocation and “round” it in an appropriate way to get an integral one. The hope would be that the fractional allocation provides some useful information regarding what a good integral allocation should look like.

In addition to guidance regarding how to allocate the items, the fractional relaxation of IP would also provide an upper bound for the geometric mean of the optimal integral solution x^* . The standard way of verifying an approximation bound for the rounding algorithm proves a bound w.r.t. this upper bound instead. Unfortunately, one can verify that the *integrality gap* of IP, i.e., the ratio of the geometric mean of the fractional solution and that of x^* , is not constant.

LEMMA 3.1. *The integrality gap of IP is $\Omega(2^m)$.*

Proof. Consider instances comprising n identical agents and m items. Each agent has a value of 1 for each of the first $m - 1$ items, and a value of 2^m for the last item. A fractional allocation can split every item equally among the agents, thus giving every one of them a bundle of value $(2^m + m - 1)/n$, leading to a geometric mean of $(2^m + m - 1)/n \geq 2^m/n$. On the other hand, any integral allocation has to assign the highly valued item to just one agent, leading to a geometric mean of at most

$$2^{m/n}[(m - 1)/(n - 1)]^{(n-1)/n} \leq 2^{m/n}m.$$

The integrality gap of IP for these instances is at least

$$\frac{2^m}{2^{m/n}mn} = \frac{2^{\frac{n-1}{n}m}}{mn}.$$

Thus, for any fixed value of $n \geq 2$, the integrality gap grows exponentially with m . \square

Lemma 3.1 implies that the fractional solution of IP cannot be used for proving a constant-factor approximation guarantee using standard techniques. Furthermore, there are instances where this fractional solution provides very limited information regarding how the items should be allocated. For instance, in the example of Figure 1b, although Agents 2 and 3 are quite different in terms of their preferences, they are identical w.r.t. the fractional allocation. Hence, when Agent 1 receives Item 1, the fractional solution provides no useful information for deciding which items these other two agents should receive.

3.2. Spending-Restricted Equilibrium. Motivated by the observation that the fractional solution of IP may not provide sufficient information for rounding, and aiming to circumvent the integrality gap, we introduce an interesting new constraint on the fractional solution. In particular, we relax the restriction that the items need to be fully allocated, and instead we restrict the total amount of money spent on any item to at most \$1, i.e., at most the budget of a single agent. For any item j , the solution needs to satisfy $\sum_{i \in N} x_{ij}p_j \leq 1$; a constraint which combines both the primal (x_{ij}) and the dual (p_j) variables of the Eisenberg-Gale program.

DEFINITION 3.2. A spending-restricted (SR) equilibrium is a fractional allocation \bar{x} and a price vector \bar{p} such that every agent spends all of her budget on her MBB items at prices \bar{p} , and the total spending on each item is equal to $\min\{1, \bar{p}_j\}$.

EXAMPLE 3.3. To provide more intuition regarding this spending-restricted market equilibrium, we revisit the problem instance that we considered in Example 2.1. In the unrestricted equilibrium of this instance, the price of the highly demanded Item 1 was \$3, and three agents were spending all of their budgets on it. In the spending-restricted equilibrium this would not be acceptable, so the price of this item would need to be increased further until only Agent 1, who has no other alternative, is spending her budget on it. The spending graph of this SR equilibrium can be seen in Figure 2a. Note that, unlike the unrestricted market equilibrium spending graph, this spending graph reveals much more information regarding the preferences of Agents 2 and 3.

Normalizing the Valuations. As we observed above, the NSW objective is scale-free, so the scale of each agent's valuations has no effect in the outcome. Using this fact, and aiming to simplify the statements and proofs of this section, we henceforth assume that each agent's valuations are normalized as follows.

DEFINITION 3.4. Given SR prices \bar{p} , we say the valuations are scaled for \bar{p} when each agent i has $v_{ij} = \bar{p}_j$ for her MBB items j at \bar{p} , and $v_{ij} < \bar{p}_j$ for any other items.

Given SR prices \bar{p} and some agent i whose valuations v_i' do not satisfy this property, we get valuations v_i that are scaled for \bar{p} by multiplying every v_{ij}' by the same

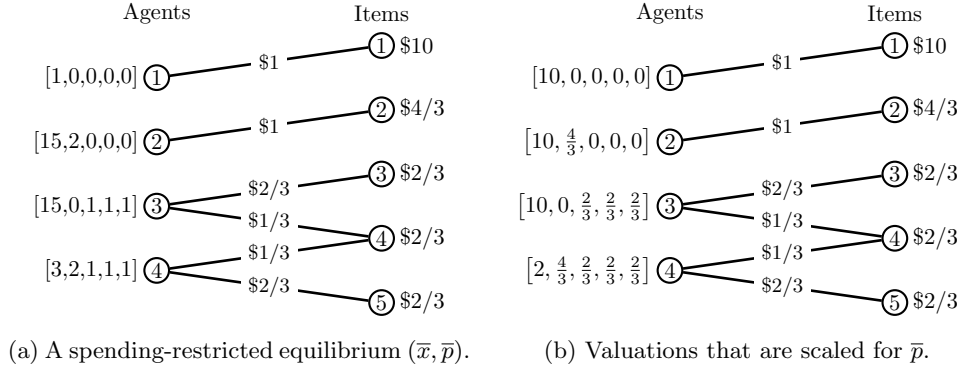


Fig. 2: A spending-restricted equilibrium (\bar{x}, \bar{p}) for the instance of Example 2.1.

value $\alpha_i = \min_k \{\bar{p}_k / v'_{ik}\}$, i.e., the inverse of the maximum bang per buck ratio. Every item j that is MBB for i at \bar{p} satisfies $v'_{ij} / \bar{p}_j = 1 / \alpha_i$, so $v_{ij} = \alpha_i v'_{ij} = \bar{p}_j$. On the other hand, for all other items $v'_{ij} / \bar{p}_j < 1 / \alpha_i$, so $v_{ij} < \bar{p}_j$. For instance, given the spending-restricted equilibrium (\bar{x}, \bar{p}) of Figure 2a, the scaled valuations for \bar{p} would be the ones in Figure 2b.

3.3. Upper Bound. Using the scaled valuations, the following theorem provides a new upper bound for the geometric mean of the optimal integral solution x^* based on the prices \bar{p} . Let $H(\bar{p})$ (resp. $L(\bar{p})$) be the set of items j with $\bar{p}_j > 1$ (resp. $\bar{p}_j \leq 1$).

THEOREM 3.5. *Given SR prices \bar{p} and agent valuations v that are scaled for \bar{p} , the optimal geometric mean is upper bounded as follows:*

$$(3.1) \quad \left(\prod_{i \in N} v_i(x^*) \right)^{1/n} \leq \left(\prod_{j \in H(\bar{p})} \bar{p}_j \right)^{1/n}.$$

Proof. If we keep the SR prices \bar{p} fixed, increasing the valuations of the agents can only increase the left hand side of Inequality (3.1), without affecting the right hand side. Therefore, since $v_{ij} \leq \bar{p}_{ij}$ for every agent i and item j (the valuations are scaled for \bar{p}), it suffices to show that (3.1) holds when $v_{ij} = \bar{p}_j$ for every i and j . Therefore, for the rest of the proof we assume that $v_{ij} = \bar{p}_j$ for every i and j . In fact, for the rest of the proof we also relax the integrality constraint for the optimal allocation x^* when it comes to items $j \in L(\bar{p})$. In other words, we let x^* be the optimal allocation when the items with price at most 1 are divisible which, once again, can only increase the left hand side of Inequality (3.1), without affecting the right hand side.

Since x^* is integral when it comes to items in $H(\bar{p})$, it allocates each one of these items to at most one agent, so at most $|H(\bar{p})|$ agents receive one of these items in x^* . Let $N_L \subseteq N$ be the set of the remaining, at least $n - |H(\bar{p})|$, agents to whom x^* assigns only items in $L(\bar{p})$. The total value that the agents in N_L can be allocated is

$$\sum_{i \in N_L} v_i(x^*) \leq \sum_{j \in L(\bar{p})} \bar{p}_j = n - |H(\bar{p})|,$$

since, in the SR equilibrium, the spending on each item in $H(\bar{p})$ is exactly 1, and the prices in $L(\bar{p})$ have to add up to the remaining budget, i.e., $n - |H(\bar{p})|$. As a result,

there are at least $n - |H(\bar{p})|$ agents in N_L , and their total value in x^* is at most $n - |H(\bar{p})|$, so $\min_{i \in N_L} \{v_i(x^*)\} \leq 1$.

We now show that every agent to whom x^* allocates a fraction of some item $j \in L(\bar{p})$ has value at most 1 in x^* . This implies that the value of every agent in N_L is at most 1. Aiming for a contradiction, assume that there exists some agent α with $v_\alpha(x^*) > 1$ who receives value $v_S > 0$ from a fraction of some item $j \in L(\bar{p})$. As we showed above, in x^* there exists some agent β with $v_\beta(x^*) \leq 1$. Given our assumption that all the agents are identical, agent β also has a value of v_S for the fraction of item j assigned to α . Therefore, if we were to take a fraction of item j of some positive value $v < v_\alpha(x^*) - v_\beta(x^*)$ from α and assign it to β instead, we would get an allocation x^{**} which contradicts the optimality of x^* :

$$\frac{\prod_{i \in N} v_i(x^{**})}{\prod_{i \in N} v_i(x^*)} = \frac{[v_\alpha(x^*) - v][v_\beta(x^*) + v]}{v_\alpha(x^*)v_\beta(x^*)} = 1 + \frac{[v_\alpha(x^*) - v_\beta(x^*) - v]v}{v_\alpha(x^*)v_\beta(x^*)} > 1.$$

Using an almost identical sequence of arguments, it is also easy to show that no agent receives more than one item from $H(\bar{p})$ in x^* . If this were the case, then reassigning one of these items to the agent with the smallest valuation would once again lead to a higher NSW. Therefore, there are exactly $|H(\bar{p})|$ agents that are each allocated a single item $j \in H(\bar{p})$ of value \bar{p}_j . These agents are allocated no fraction of items $j \in L(\bar{p})$, because their value in x^* is greater than 1 (since $\bar{p}_j > 1$).

As a result, we have shown that the product of the values of the agents in N_L in x^* is at most 1, and the product from the agents matched to an item in $j \in H(\bar{p})$ is equal to $\prod_{j \in H(\bar{p})} \bar{p}_j$, which implies the theorem. \square

3.4. Spending-Restricted Rounding Algorithm. Our approximation algorithm, *Spending-Restricted Rounding* (SRR), begins by computing an SR allocation \bar{x} and prices \bar{p} and then appropriately allocates each item to one of the agents who was receiving some of it in \bar{x} , i.e., to one of its neighbors in the spending graph $Q(\bar{x})$. In doing so, we ensure that most of the agents get at least half of the value that they would have received in the fractional solution and that every other agent receives a significant enough portion of that value to guarantee a constant factor approximation.

As we discussed in Section 2, we assume that the spending graph of the fractional allocation \bar{x} is a forest of trees. Also, since every item is (partially) allocated to at least one agent, every tree in this forest includes a vertex corresponding to an agent. Once the SRR algorithm computes this forest in Step 1, for each tree it chooses one of its vertices that corresponds to an agent to be the root of the tree. Therefore, the vertices at depth 1 all correspond to items that the root-agent is spending on, those at depth 2 correspond to agents that are spending on items of depth 1, and so on.

The next two steps of the SRR algorithm (Steps 3 and 4) make the first integral assignments. Any items that correspond to leaves in the rooted trees of the spending graph are assigned to their parent-agent, and items whose price \bar{p}_j is at most $1/2$ are assigned to their parent-agent as well. Given the spending graph, the first type of rounding (Step 3) is straightforward, since the parent-agent is the only one spending on the leaf-items. The second (Step 4) is less obvious since there could be several child-agents of this item that are spending on it in \bar{x} . Nevertheless, since the price of any item allocated in this step is no more than $1/2$, any child-agent is spending at most half of her budget on it; the rest of her budget is spent on her children-items. Also, note that, after Step 4, each one of the child-agents becomes the root of a new tree in the spending subgraph induced by the agents and the remaining items.

The last step of our algorithm assigns each one of the remaining, highly priced, items to a distinct agent that is adjacent to that item. In other words, it computes a matching of items to agents, which is restricted by the spending graph edges. In fact, the algorithm computes the *best* possible such matching, given the assignment of items that took place during Steps 3 and 4. This is implemented using a simple maximum weight matching algorithm on an appropriately weighted spending graph. To define the appropriate weights for the spending edges, let $v_i(x')$ denote the value of agent i for the items that she was previously allocated (if any) during Steps 3 and 4. The weight of the spending edge between an agent i and an item j that was not previously allocated is then set to $w_{ij} = \log(v_{ij} + v_i(x')) - \log(v_i(x'))$. Since $v_i(x')$ is the value of agent i if she is not matched in Step 5, and $v_{ij} + v_i(x')$ is her value if she is matched to item j , the weight of each edge (i, j) is the marginal increase for agent i in terms of the logarithm of her value if she is matched to item j . Therefore, the maximum weight matching on this graph computes, over all the possible matchings of the remaining items, the one that maximizes the sum of the logarithms of the valuations, which is equivalent to maximizing the Nash social welfare.

Algorithm 1: Spending-Restricted Rounding (SRR).

- 1 Compute a spending-restricted equilibrium (\bar{x}, \bar{p}) .
 - 2 Choose a root-agent for each tree in the spending graph $Q(\bar{x})$.
 - 3 Assign any leaf-item in the trees to its parent-agent.
 - 4 Assign any item j with $\bar{p}_j \leq 1/2$ to its parent-agent.
 - 5 Compute the optimal matching of the remaining items to adjacent agents.
-

THEOREM 3.6. *The integral allocation \tilde{x} computed by the Spending-Restricted Rounding algorithm always satisfies*

$$\left(\prod_{i \in N} v_i(x^*) \right)^{1/n} \leq 2.889 \left(\prod_{i \in N} v_i(\tilde{x}) \right)^{1/n}.$$

Since the last step of the SRR algorithm computes the optimal matching of the remaining items, in order to prove Theorem 3.6, it suffices to show that there always exists some matching of these items that provides the desired approximation guarantee. In order to define a matching that satisfies this property, we begin by pruning the rooted spending subgraph of \bar{x} induced by the agents and the remaining items. Given this spending graph, we extract the *pruned* spending graph $P(\bar{x})$ by removing all but one of the child-agents of each item. Specifically, for each item j that has more than one child-agents in the initial spending graph, we remove the edges connecting it to all but the one child-agent that spends the most money on j (i.e., the one with the largest \bar{x}_{ij} value).

Every agent that loses an edge due to this pruning becomes the root of a new tree in the forest of $P(\bar{x})$. We refer to the trees of the pruned graph $P(\bar{x})$ as the *matching-trees*. For each item in a matching-tree, there are exactly two agents competing for it, the parent-agent and the child-agent that survived the pruning. Therefore, each matching-tree that contains $k > 0$ agents also contains exactly $k - 1$ items.

In the rest of this section, we show that there exists a matching that satisfies the desired inequality while using only edges of the pruned graph. In proving this, we

first prove Lemmas 3.7 and 3.8, which provide some intuition regarding the matching-trees. If T is some matching-tree of the pruned spending graph $P(\bar{x})$, then let M_T denote the union of items in T with the items that were assigned to agents in T in Steps 3 and 4. The following lemma provides a lower bound for the total money that is spent on these items in \bar{x} .

LEMMA 3.7. *For any matching-tree T with k agents*

$$\sum_{j \in M_T} \min\{1, \bar{p}_j\} \geq k - 1/2.$$

Proof. We first observe that there can be at most one item $j \notin M_T$ such that some agent $i \in T$ is spending on j in \bar{x} . To verify this, note that the only reason why j is not in M_T although the edge (i, j) exists in the initial spending graph $Q(\bar{x})$ is either that j was assigned to its parent-agent by Step 4, or that the edge (i, j) was removed during the pruning. In both cases the agent who “loses” j is its child-agent, which implies that i becomes the root of the matching-tree that she belongs to. Therefore, if any such item j exists, it has to be the case that the root of T was its child-agent, and there is only one such item.

The total spending of the agents in T is equal to k so, having shown that there is at most one item $j \notin M_T$ that is receiving some of that spending in \bar{x} , it suffices to show that the root of T is spending at most $1/2$ on it. If j was lost during Step 4, its price is at most $1/2$ and, hence, so is i ’s spending on it. If, on the other hand, the edge (i, j) was removed during the pruning, then i is not the highest-spending child of j . Since we have enforced that the spending on any item in \bar{x} is at most 1, this once again means that i is not spending more than $1/2$ on j , which proves the lemma. \square

LEMMA 3.8. *For any matching-tree T with k agents, there exists an agent $i \in T$ who, during Steps 3 and 4 received one or more items that she values at least $1/(2k)$.*

Proof. Since the total spending on any item is at most 1, the total spending on the $k - 1$ items in T is at most $k - 1$. But, according to Lemma 3.7, the total spending on the items in M_T is at least $k - 1/2$, implying that the spending on items in $M_T \setminus T$ is at least $1/2$. Since each one of these items was assigned to an agent for whom it is MBB, the total value that the agents in T received from these items is also at least $1/2$, so at least one of these agents received a value of at least $1/(2k)$. \square

Using Lemma 3.8, we can now prove our main result.

Proof. [of Theorem 3.6] As we observed above, after the pruning each item of a matching-tree T is competed for by exactly two agents in T and, if T contains $k > 0$ agents, then it also contains exactly $k - 1$ items. Therefore, any matching restricted to the edges of $P(\bar{x})$ has to leave one agent unmatched for each matching-tree T . In light of Lemma 3.8, one naive way of implementing the matching would be to choose some agent whose value is already at least $1/(2k)$ and to exclude her from the matching. Then, each other agent i would be matched to one of her MBB items j , so her value for it would be $v_{ij} = \bar{p}_j > 1/2$. If $N_T \subseteq N$ is the set of k agents of tree T , then this matching would give

$$\prod_{i \in N_T} v_i(x) \geq \left(\frac{1}{2}\right)^{k-1} \frac{1}{2k} = \frac{1}{2^k k}.$$

The last step of our algorithm chooses the best possible matching with respect to

the geometric mean so it will do at least as well. Therefore, our algorithm guarantees

$$(3.2) \quad \left(\prod_{i \in N} v_i(\tilde{x}) \right)^{1/n} \geq \left(\prod_T \frac{1}{2^{k(T)} k(T)} \right)^{1/n} \geq \frac{1}{2e^{1/e}},$$

where $k(T)$ is the number of agents in each tree T . The second inequality holds because, over all the ways in which the initial spending graph can be partitioned into trees, the one that minimizes the RHS of the first inequality splits the spending tree into r equal sized sub-trees of $k = n/r$ agents each; this is, in turn, minimized when $n/r = e$.

If $H(\bar{p})$ is empty, then the upper bound of Theorem 3.5 is equal to 1, and Inequality 3.2 implies that our approximation factor is at most $2e^{1/e} \approx 2.889$ for this case. If $H(\bar{p})$ is not empty, the matching assigns each item in $H(\bar{p})$ to a distinct agent for whom this item is MBB. The analysis above assumes only that these agents get a value more than $1/2$, but they get $\bar{p}_j > 1$, i.e., even more than $\bar{p}_j/2$. Substituting $\bar{p}_j/2$ for $1/2$ for these agents and using the same arguments, we get

$$\left(\prod_{i \in N} v_i(\tilde{x}) \right)^{1/n} \geq \frac{1}{2e^{1/e}} \left(\prod_{j \in H(\bar{p})} \bar{p}_j \right)^{1/n}.$$

Given Theorem 3.5, this inequality proves the theorem. \square

4. Computing the Spending-Restricted Equilibrium. In this section we show that a Spending-Restricted allocation \bar{x} and prices \bar{p} that the Spending Restricted Rounding algorithm uses can be computed in polynomial time. As Lemma 4.1 shows, it actually suffices to find the spending graph of \bar{x} , i.e., $Q(\bar{x})$, so our algorithms focus on computing this spending graph.

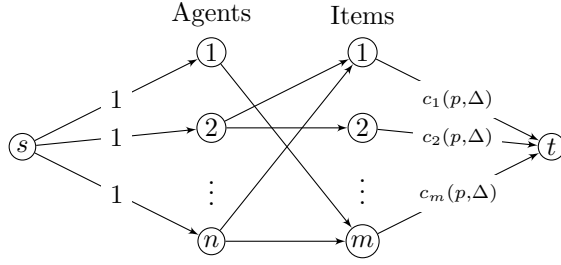
LEMMA 4.1. *Given the spending graph $Q(\bar{x})$, the allocation \bar{x} and prices \bar{p} can be computed in polynomial time.*

Proof. Let T be a connected component in the spending graph. For any two items in T , the ratio of their prices is known since all the edges in the path connecting these items are MBB edges of the corresponding agents. Hence, setting one item's price in T also sets all the rest, and we can also sort them in non-increasing order. If there are k agents in T , then the prices need to satisfy $\sum_{j \in T} \min\{p_j, 1\} = k$. If we knew that q of the items had $\bar{p}_j > 1$, then, we could find the right prices by removing the q highest priced items from T and solving $\sum_{j \in T} p_j = k - q$ for the rest. We try this out for all possible values of q , until the prices set for the remaining items are at most 1, which yields \bar{p} . Given \bar{p} and the spending graph, we can compute \bar{x} in a bottom-up fashion, starting from the leaves of the spending graph. \square

Throughout this section, apart from restricting the total spending on any item to always being at most 1, we are also restricting it to being a multiple of $\Delta = 1/2^r$ for some $r \in \mathbb{N}$; the smaller Δ is, the finer the discretization of the spending space.

DEFINITION 4.2. *An allocation x is a Δ -allocation with respect to prices p if every agent is spending only on her MBB items, and the total spending on item j is $\min\{1, \Delta \lceil p_j / \Delta \rceil\}$, unless p_j is a multiple of Δ . If $p_j / \Delta \in \mathbb{N}$, the spending on j is at least $\min\{1, p_j\}$ and at most $\min\{1, p_j + \Delta\}$. The spending capacity of item j is then*

$$c_j(p, \Delta) = \begin{cases} \min\{1, \Delta \lceil p_j / \Delta \rceil\} & \text{if } p_j \text{ is not a multiple of } \Delta \\ \min\{1, p_j + \Delta\} & \text{otherwise.} \end{cases}$$

Fig. 3: Δ -allocation flow network at prices p .

As p_j increases, so does $c_j(p, \Delta)$. In particular, $c_j(p, \Delta)$ increases by Δ whenever $p_j < 1$ becomes a multiple of Δ . Note that a Δ -allocation may not exist for some price vector p . If, for example, $\sum_{j \in M} \min\{p_j, 1\} > n$, then the agents do not have enough budget to cover these costs. Also, note that, in a Δ -allocation, there may exist agents i that are not spending all of their budgets, i.e., $\sum_j x_{ij} p_j < 1$ for some i .

DEFINITION 4.3. *If every agent is spending all of her budget in a Δ -allocation, then we call it a full Δ -allocation. We say that p supports a (full) Δ -allocation when there exists a (full) Δ -allocation at prices p .*

OBSERVATION 4.4. *Given some Δ and prices p that support a Δ -allocation, we can compute a Δ -allocation as the maximum flow of the network in Figure 3.*

The source s is connected to the agent-vertices via edges of capacity 1, which corresponds to each agent's budget. Each agent's vertex is then connected to the vertices of her MBB items at prices p via edges of unbounded capacity. Finally, each item j 's vertex is connected to the sink t via edges of capacity equal to $c_j(p, \Delta)$. Given a maximum flow for this network, the amount of flow between an agent vertex i and an item vertex j corresponds to the amount that i spends on j . If the capacities of the edges leaving s are saturated, the Δ -allocation is full.

In the rest of this section we provide algorithms that discover the spending graph of \bar{x} by computing full Δ -allocations for appropriate values of Δ . In doing so, our algorithms gradually increase the prices, while making sure these prices always support a Δ -allocation for some $\Delta > 0$. Henceforth, whenever we refer to computing a Δ -allocation, we mean one that minimizes the total amount of unspent budget, which is precisely what the maximum flow approach of Observation 4.4 does. Finally, in order to simplify our statements regarding the run time of our algorithms, rather than focusing on explicit bounds we just use $\text{poly}(\cdot)$ to denote polynomial dependence.

4.1. Price Increase Algorithm. We begin with an algorithm that receives as input some Δ and prices p that support a Δ -allocation, and outputs prices $p' \geq p$ that support a full Δ -allocation. To reach p' , the algorithm gradually increases the prices of selected items *in proportion*, i.e., it multiplies all of them by the same constant c , so from p_j they become $c \cdot p_j$. More precisely, given some p and Δ , the PriceIncrease algorithm computes a Δ -allocation x aiming to spend as much of the agents' budget as possible. If there exists some agent i who is not spending all of her budget in x , then the algorithm increases the prices of her MBB items, which are currently over-demanded. Apart from these items' prices, the algorithm also increases the prices of any item in i 's *reachable set* R , i.e., any item that is reachable from agent i via paths

alternating between directed edges of the MBB graph $G(p)$ and undirected edges of the spending graph $Q(x)$.

The increase of the prices of the items in R can create opportunities for i to spend her remaining budget. For example, it can cause the appearance of a new MBB edge. More precisely, since the reachable items become increasingly less appealing due to the increased prices, some agent i' spending on them may become interested in a different item $j' \notin R$. This introduces an alternating path from the agent i who has unspent budget in x to this item j' . If the spending capacity of j' is not saturated in x , then the alternating path could be used for i to spend more of her budget. For instance, say that, in x , agent i' is spending money on an item j that is MBB for i . In this case the alternating path is $i \rightarrow j \rightarrow i' \rightarrow j'$, where the second edge is an undirected spending edge in $Q(x)$ and the other two are directed edges in $G(p)$. Clearly, if i' spends some amount $b > 0$ on item j' , this allows i to spend b on item j .

Similarly, if the increase of prices in R causes the capacity of some item to increase, the corresponding alternating path could be used for i to spend more of her budget. Therefore, whenever one of these events takes place, the *PriceIncrease* algorithm computes a new Δ -allocation with the latest prices, and the maximum flow of the corresponding network takes advantage of any opportunities to spend more of the agents' budget. Once all of the budgets are fully spent, the algorithm terminates.

Algorithm 2: The *PriceIncrease*(p, Δ) algorithm.

```

1 Compute a  $\Delta$ -allocation  $x$  for prices  $p$ 
2 while there exists an agent  $i$  with unspent money in  $x$  do
3   Let  $R$  be  $i$ 's reachable set via  $G(p)$  and  $Q(x)$  edges.
4   Increase prices  $p$  of items in  $R$  in proportion until one of these events:
5   if a new MBB edge appears in  $G(p)$  then compute a  $\Delta$ -allocation for  $p$ .
6   if  $c_j(p, \Delta)$  increases for some  $j \in R$  then compute a  $\Delta$ -allocation for  $p$ .
7 Return  $p$ 
```

The following lemma upper bounds the running time of *PriceIncrease*(p, Δ) as a function of the unspent budget in the Δ -allocation at prices p . This bound will prove very useful when we use this algorithm as a subroutine in the rest of the section.

LEMMA 4.5. *If the total unspent budget in a Δ -allocation at p is $s\Delta$, then *PriceIncrease*(p, Δ) terminates in $\text{poly}(m, s)$ time.*

Proof. We first point out that, given prices p and a Δ -allocation x , computing the prices p' at which the next event (Step 5 or 6) takes place requires time $\text{poly}(m)$. For the second type of event, we just need to find the item in R whose price will become a multiple of Δ first, i.e., the one with the smallest $\Delta \lceil p_j / \Delta \rceil / p_j$. For the first type of event, for each agent i' who is spending on items in R , we compute when its first MBB edge will appear. This amounts to finding the item $j \notin R$ for which agent i' has the highest bang per buck ratio, $v_{i'j} / p_j$, and dividing i' 's *maximum* bang per with this ratio for item j . Clearly, all of the above need at most $\text{poly}(m)$ time.

We now show that the number of events that take place during the execution of *PriceIncrease* is $\text{poly}(m, s)$. When the capacity of an item increases (Step 6), Δ more budget is spent in the Δ -allocation for the new prices, hence at most s such events can take place. Also, once a new MBB edge appears (Step 5), it can disappear only after the capacity of some item increases. Therefore, for each capacity increase event

there can be $O(mn)$ new MBB edge appearance events. \square

LEMMA 4.6. *PriceIncrease(p, Δ) returns p that support a full Δ -allocation.*

Proof. We first show that every time the PriceIncrease algorithm computes a Δ -allocation at prices p (Steps 1, 5, and 6), these prices actually support a Δ -allocation. We know that this is the case for Step 1 since we require that the input to the algorithm satisfy this property. Regarding Step 5, when a new MBB edge appears, the capacities of the items have not changed, so the Δ -allocation that was computed before this event took place remains a Δ -allocation. Therefore, the new prices still support a Δ -allocation. Similarly for Step 6, when the capacity of some item $j \in R$ increases, the Δ -allocation x computed before this event took place remains a Δ -allocation. To verify this fact, note that, the capacity of j increases from some c_j to $c_j + \Delta$ when p_j becomes a multiple of Δ . Since agent i was not spending all of her budget in x , the total spending on j had to be equal to c_j ; otherwise i could take advantage of it using the alternating path. Therefore x remains a Δ -allocation for the new prices since a spending of c_j is enough for j at the new prices.

The stopping condition of the while loop (Step 2) implies that, when this algorithm terminates, the agents are spending all their money in the Δ -allocation at p' . Since Lemma 4.5 proves that the algorithm always terminates, the results follows. \square

The smaller the value of Δ , the more the full Δ -allocation resembles the desired SR equilibrium. The following lemma, which is an adaptation of an observation due to Orlin [33], shows that, in fact, if Δ is small enough, the spending graph of the full Δ -allocation is the same as the one for \bar{x} .

LEMMA 4.7 (Orlin [33]). *There is $r \in O(m \log V_{\max})$, where $V_{\max} = \max_{i,j,k} \left\{ \frac{v_{ij}}{v_{ik}} \right\}$, such that a full Δ -allocation for $\Delta = 1/2^r$ has the same spending graph as \bar{x} .*

Lemmas 4.6 and 4.7 suggest a first way of computing the spending graph of \bar{x} : let $\Delta = 1/2^r$, where $r \in O(m \log V_{\max})$, and let the prices p be small enough to support a Δ -allocation. Then we call *PriceIncrease(p, Δ)*, which returns prices p' . Computing a Δ -allocation for p' yields the desired spending graph.

Unfortunately, if we use this approach then s , the total unspent budget in the initial Δ -allocation can be as high as n/Δ , i.e., $\Omega(2^m V_{\max})$. Therefore, the run time of *PriceIncrease(p, Δ)* might not be polynomial, and hence this approach is not efficient. In what follows we show how to use *PriceIncrease* as a subroutine in order to get a polynomial time algorithm.

4.2. A weakly polynomial algorithm. Following the approach of Devanur et al. [19] and the simplification of Orlin [33], we can compute the spending graph of \bar{x} in weakly polynomial time. Rather than using the *PriceIncrease* algorithm just once with a very small Δ , Algorithm 3 uses it repeatedly as a subroutine, with gradually decreasing values of Δ .

Initially, $\Delta = 1/(2^{\bar{m}})$, where $\bar{m} = 2^{\lceil \log 2m \rceil}$ is the smallest integer power of 2 at least as large as $2m$; the prices are initialized at $p_j = \frac{1}{2} \max_i \{v_{ij} / \sum_k v_{ik}\}$ for each item j . These prices support a Δ -allocation since each agent i can afford all the items for which $p_j = v_{ij} / \sum_k v_{ik}$, including spending up to an extra Δ on each item beyond its price.

Algorithm 3 calls the *PriceIncrease* algorithm on this input, which returns prices that support a full Δ -allocation. The value of Δ is then halved and the *PriceIncrease* algorithm is called again using this new Δ and the latest prices. We will henceforth refer to this pair of operations (see Steps 4 and 5) as a Δ -halving step. By repeating this Δ -halving step until Δ reaches the desired value in $\Theta(2^{-m}/V_{\max})$, Algorithm 3 reaches a full Δ -allocation with the desired spending graph.

Algorithm 3: The weakly polynomial algorithm.

```

1 Let  $\Delta = 1/(2\bar{m})$  and  $p_j = \frac{1}{2} \max_i \{v_{ij} / \sum_k v_{ik}\}$ .
2  $p = \text{PriceIncrease}(p, \Delta)$ .
3 for  $r = 1$  to  $r \in O(m \log V_{\max})$  do
4    $\Delta = \Delta/2$ .
5    $p = \text{PriceIncrease}(p, \Delta)$ .
6 Return  $p$ 

```

LEMMA 4.8. *Algorithm 3 terminates in $\text{poly}(m, \log(V_{\max}))$ time.*

Proof. It suffices to show that for every one of the $O(m \log V_{\max})$ times that $\text{PriceIncrease}(p, \Delta)$ is called, it terminates in $\text{poly}(m)$ time. The first time PriceIncrease is called, the value of Δ is large enough so that the number of iterations is limited. In particular, the overall budget among the agents is n , so the unspent budget can be at most $(n/\Delta)\Delta = (2\bar{m}n)\Delta$. Therefore, substituting $s = O(mn)$ in Lemma 4.5 implies that the PriceIncrease process will terminate in $\text{poly}(m)$ steps. For all the subsequent calls to $\text{PriceIncrease}(p, \Delta)$, note that the price vector p computed in the previous call supports a full 2Δ -allocation. The Δ -allocation at prices p either leaves the spending on an item unchanged or reduces it by Δ , and thus the unspent budget is at most $m\Delta$. Once again, by Lemma 4.5, this implies that the subroutine will terminate in $\text{poly}(m)$ steps. \square

4.3. A strongly polynomial algorithm. In this section, inspired by the work of Orlin [33], we show that we can also compute the spending-restricted equilibrium in *strongly* polynomial time. To remove the running time dependence on V_{\max} , we propose an accelerated version of the weakly polynomial algorithm which follows a very similar price increase trajectory, but skips some steps in order to discover the spending graph of \bar{x} faster. Our algorithm builds on the following lemma.

LEMMA 4.9. *If, in a full Δ -allocation, some subset \mathcal{E} of the spending graph edges all have spending at least $2\bar{m}\Delta$ then, for any $r \in \mathbb{N}$ and $\Delta' = \Delta/2^r$, there exists a full Δ' -allocation such that the spending on every edge in \mathcal{E} is at least $2\bar{m}\Delta'$.*

Proof. We show that the claim is true for $\Delta' = \Delta/2$; the lemma follows by induction on Δ' . Given the full Δ -allocation we construct the desired full Δ' -allocation.

Given prices p , halving Δ may reduce the spending capacity of some items, i.e., there may exist items j such that $c_j(p, \Delta/2) < c_j(p, \Delta)$. More precisely, the drop in the capacity of an item is either 0 or $\Delta/2$. For each item whose capacity drops below the total spending on that item, we reduce the spending on it by $\Delta/2$ and return this spending in the form of unspent budget to some of the agents that were spending their budget on that item in the full Δ -allocation; this first phase yields a $\Delta/2$ -allocation. The total amount of unspent budget in this $\Delta/2$ -allocation is at most $m\Delta/2$. In the second phase, we apply $\text{PriceIncrease}(p, \Delta/2)$. As we described earlier, this algorithm gradually increases the prices of some items and then uses up the unspent budget by finding augmenting paths that lead to items with increased or spare capacity. These alternating paths may cause the spending on their edges to drop, but the total reduction is at most the total unspent budget. Therefore the maximum reduction in spending on any edge is at most $m\Delta/2$.

To conclude the proof, we observe that we reduced the spending on each edge by at most $\Delta/2$ in the first phase and at most $m\Delta/2$ in the second. Hence, every edge whose spending was at least $2\bar{m}\Delta$ in the full Δ -allocation has at least $2\bar{m}\Delta - (m+1)\Delta/2 \geq$

$2\bar{m}\Delta - \bar{m}\Delta = 2\bar{m}\Delta'$ spending in the full Δ' -allocation that we constructed. \square

In light of Lemma 4.9, consider the following modification of the weakly polynomial algorithm of the previous subsection. Once the algorithm computes a full Δ -allocation that has a non-empty subset \mathcal{E} of spending graph edges with spending at least $2\bar{m}\Delta$, then it maintains these edges in the spending graph of *all* the subsequent full Δ' -allocations that it computes (we explain how this can be done below). As a result, this algorithm gradually discovers the edges that will eventually form the computed spending graph $Q(\bar{x})$.

DEFINITION 4.10. *Given a full Δ -allocation x , we call the spending graph edges with spending at least $2\bar{m}\Delta$ “discovered edges”, and we maintain them as discovered edges in all subsequent full Δ' -allocations. We call the connected components of the spending graph $Q(x)$ with respect to the discovered edges the “discovered components”.*

Given a full Δ -allocation with a set \mathcal{E} of discovered edges, how can we make sure that these edges remain discovered in the full Δ' -allocation for $\Delta' = \Delta/2^r$ and $r \in \mathbb{N}$? To achieve this, we essentially “pre-allocate” the desired amount of $2\bar{m}\Delta'$ spending on all these edges, and Lemma 4.9 guarantees that such a Δ' -allocation exists. To be more precise, our strongly polynomial algorithm keeps track of all the edges that have been discovered and, whenever it computes a Δ' -allocation via the flow network of Figure 3, it appropriately adjusts the amount of budget for the agents and the amount of spending capacity for the items. In particular, when computing a Δ' -allocation, the capacity of an edge connecting the source s to the vertex of some agent i in the flow network of Figure 3 is $1 - q_i(2\bar{m}\Delta')$, where q_i is the number of discovered edges that involve i . Similarly, the capacity of an edge connecting the vertex of some item j to the sink t is $c_j(p, \Delta') - q_j(2\bar{m}\Delta')$, where q_j is the number of discovered edges incident to j . As a result, the maximum flow of this network yields a Δ' -allocation that keeps these edges discovered and defines how to distribute the remaining budget.

Initially, before any edge is discovered, there are $m + n$ discovered components, each of which corresponds either to an agent or to an item. This number gradually decreases as more edges of $Q(\bar{x})$ are discovered, thus generating larger components that contain both agents and items. The weakly polynomial algorithm gradually discovers these edges by repeatedly halving the value of Δ and using the PriceIncrease subroutine, but it may take more than a strongly polynomial number of Δ -halving steps between two successive discoveries of such edges. To avoid this issue, our strongly polynomial algorithm checks whether the next discovery might be too far away, in which case it uses a new subroutine, FindNext. This subroutine returns new values of p and Δ which are guaranteed to be within $O(\log m)$ Δ -halving steps (Steps 5 and 6 in Algorithm 4) of the next discovery. Hence, Algorithm 4 discovers the $O(n + m)$ edges of the SR spending graph in strongly polynomial time (we prove this formally for Theorem 4.15).

DEFINITION 4.11. *Given a discovered component C and prices p , let $R(C, p)$ be the set of discovered components reachable from C using directed MBB and undirected discovered edges. Also, let the surplus $s(C, p)$ of C at prices p be the difference between the total budget of agents in C and the spending restricted prices of items in C , i.e.,*

$$s(C, p) = |N \cap C| - \sum_{j \in M \cap C} \min\{1, p_j\}.$$

The components containing a single agent have a positive surplus of 1, and the ones containing a single item j have a negative surplus of $-\min\{1, p_j\}$. As edges are

Algorithm 4: The strongly polynomial algorithm.

```

1 Let  $\Delta = 1/(2\bar{m})$  and  $p_j = \frac{1}{2} \max_i \{v_{ij} / \sum_k v_{ik}\}$ .
2  $p = \text{PriceIncrease}(p, \Delta)$ .
3 while  $\Delta > 0$  do
4   while  $\min_C \{s(C, p)\} \leq \frac{-\Delta}{4\bar{m}}$  do
5      $\Delta = \Delta/2$ .
6      $p = \text{PriceIncrease}(p, \Delta)$ .
7      $(p, \Delta) = \text{FindNext}(p, \Delta)$ .
8      $p = \text{PriceIncrease}(p, \Delta)$ .
9 Return  $p$ 

```

discovered, the surplus of the components formed can be either negative or positive, depending on the number of agents and the prices of the items they contain. The sum of the surpluses across all the components is always non-negative, otherwise the prices would add up to more than total budget, and hence would not support a Δ -allocation. The following important lemma shows that, if the surplus of some component is sufficiently negative, then a new edge will be discovered within a small number of Δ -halving steps.

LEMMA 4.12. *If at prices p some discovered component C has $s(C, p) \leq -\Delta/(4\bar{m})$, a new edge will be discovered in $O(\log m)$ Δ -halving steps.*

Proof. We show that in the full Δ' -allocation, where $\Delta' = \Delta/(4\bar{m}^3)$, at least one agent that belongs to a discovered component $D \neq C$ spends at least $2\bar{m}\Delta'$ on an item in C . Therefore, this edge will be discovered after $\log(4\bar{m}^3)$ steps of Δ -halving, and a new component that combines this edge with C and D will be formed.

To show that such an edge exists in the full Δ' -allocation, note that the Δ -halving steps can only raise the prices in C , and thus the surplus of C can only decrease further. Therefore, in the full Δ' -allocation, the surplus of C remains at most $-\Delta/(4\bar{m}) = -\bar{m}^2\Delta'$. Since the spending graph of the full Δ' -allocation is a tree, there are at most $n - 1$ spending edges between agents not in C and items in C . Also, in the full Δ' -allocation the negative surplus of C needs to be covered by these agents, which implies that the spending on one of these edges is at least $\bar{m}^2\Delta'/(n - 1) \geq \bar{m}^2\Delta'/m \geq 2\bar{m}\Delta'$. \square

The result of Lemma 4.12 allows us to check whether the discovery of the next edge of $Q(\bar{x})$ will take place within $O(\log m)$ Δ -halving steps (Step 4). If no discovered component has a sufficiently negative surplus, Algorithm 4 calls the subroutine $\text{FindNext}(p, \Delta)$, which returns prices $p' \geq p$ and either $\Delta' = \Delta/2^r$ (for some $r \geq 1$), such that the next edge discovery is near, or $\Delta' = 0$, in which case p' are the desired prices. **The FindNext algorithm is not well defined for $S_{\max} = 0$!** If, on the other hand, there exists some component with sufficiently negative surplus, Step 4 of Algorithm 4 also checks whether there exists a component with strictly positive surplus. If no such component exists, then all the components have a surplus of zero, and all the edges of $Q(\bar{x})$ have been discovered. In this case, the next call to FindNext returns a value of $\Delta = 0$ and the algorithm terminates. Finally, as long as there exists a component with sufficiently negative surplus, and one with strictly positive surplus, the algorithm enters the loop of Steps 4–6, which guarantees a new edge discovery within every $O(\log m)$ iterations.

4.3.1. FindNext Subroutine. In order to complete the description of our strongly polynomial algorithm, we now explain how the FindNext subroutine works. At a very high level, this subroutine is quite similar to the PriceIncrease algorithm, which chooses an agent who is not spending enough and then increases the prices of the items reachable from that agent in order to spend some of her budget. Instead of focusing on agents, FindNext focuses on discovered components. For each component C , FindNext increases the prices of the items in that component, as well as those of items in components that are reachable from C , i.e., in the components of $R(C, p)$.

The prices of the items in components reachable from C are increased until either the surplus of C becomes 0, or until the surplus of some component D reachable from C drops below $-s(C, \tilde{p})/(4\overline{m})$ (see Step 6). In both these cases, the reasoning behind the stopping condition is the need to guarantee that the prices support a Δ' -allocation for some appropriate Δ' . In the former case, when $s(C, \tilde{p}) = 0$, the agents in C have just enough money to pay for the items in their component. In the latter case, the goal is to ensure that the negative surplus of components like D can still be covered using the positive surplus of C .

After going through this first round of price increases, each component C suggests an estimate S^C regarding the smallest surplus that it can reach while ensuring that the induced prices support some Δ' -allocation. Among all these estimates, we choose the most conservative one, i.e., the largest one, called S_{\max} . Then, using S_{\max} , we perform a second round of the same price increase process for each component C , but this time stopping before the component's surplus drops below S_{\max} . The price p'_j returned by FindNext for item j is set to the maximum price that the item reached during one of the price increase processes of the second round. The value of Δ' that FindNext returns is set to approximately $S_{\max}/(4\overline{m})$; in order to ensure that $\Delta' = 1/2^r$ for some $r \in \mathbb{N}$, its precise value is set to the smallest power of $1/2$ that is larger than $S_{\max}/(4\overline{m})$.

Algorithm 5: The $FindNext(p, \Delta)$ algorithm.

```

1 for each discovered component  $C$  do
2   Let  $\tilde{p} := p$ 
3   Increase prices  $\tilde{p}$  in  $R(C, \tilde{p})$  in proportion, until one of these events:
4   if a new MBB edge appears then
5     | update the set  $R(C, \tilde{p})$  and go to Step 3
6   if  $s(C, \tilde{p}) = 0$  or  $s(D, \tilde{p}) \leq -s(C, \tilde{p})/(4\overline{m})$  for some  $D \in R(C, \tilde{p})$  then
7     | let  $S^C := s(C, \tilde{p})$  and stop increasing
8 Let  $S_{\max} := \max_C S^C$ 
9 Let  $\overline{S}_{\max} := 2^{\lceil \log S_{\max} \rceil}$ 
10 for each discovered component  $C$  do
11   Let  $p^C := p$ 
12   Increase prices  $p^C$  in  $R(C, p^C)$  in proportion, until one of these events:
13   if a new MBB edge appears then
14     | update the set  $R(C, p^C)$  and go to Step 12
15   if  $s(C, p^C) \leq S_{\max}$  then
16     | stop increasing
17 For each item  $j$ , let  $p'_j := \max_C p^C_j$ 
18  $\Delta' := \overline{S}_{\max}/(4\overline{m})$ 
19 Return  $(p', \Delta')$ 

```

The following lemmas, whose proofs are deferred to the appendix, show that FindNext satisfies the desired properties. Lemma 4.13 guarantees that the value Δ' returned by FindNext is at most $\Delta/2$, and hence shows that the value of Δ keeps shrinking. Lemma 4.14 (i) ensures that the prices p' that FindNext returns support a Δ' -allocation; these two properties guarantee that the algorithm always terminates. Finally, Lemma 4.14 (ii) implies that, once FindNext is called, a new edge discovery is guaranteed to take place within a logarithmic number of Δ -halving steps.

LEMMA 4.13. *The value Δ' computed by FindNext(p, Δ) satisfies $\Delta' \leq \Delta/2$.*

LEMMA 4.14. *The prices p' and value Δ' computed by FindNext are such that: (i) p' supports a Δ' -allocation, (ii) some discovered component D has $s(D, p') \leq \frac{-\Delta'}{4m}$.*

Given these properties, we can now conclude with the main result of this section.

THEOREM 4.15. *Algorithm 4 computes $Q(\bar{x})$ in time $\text{poly}(n, m)$.*

Proof. The fact that Algorithm 4 computes the desired spending graph $Q(\bar{x})$ is implied by Lemmas 4.13 and 4.14, which show that the FindNext algorithm returns $\Delta' < \Delta/2$ and prices p' that support a Δ' -allocation. As we showed for the weakly polynomial algorithm, if every time $\text{PriceIncrease}(p, \Delta)$ is called, the value of Δ is halved, and the prices p support a Δ -allocation, then the algorithm will eventually discover all the edges of $Q(\bar{x})$. Once all the edges are discovered, every component will have non-negative surplus, so Algorithm 4 will keep calling FindNext and, each time FindNext is called, it will reduce the surplus of some component to zero. This will eventually lead to a surplus of 0 for every component, and cause the algorithm to terminate. The rest of the proof shows that, in fact, the algorithm terminates in $\text{poly}(n, m)$ time.

We first note that, whenever the algorithm enters the while loop of Steps 4 to 6, it leaves this loop within $\text{poly}(n, m)$ time. In particular, as long as the execution remains in the loop, the algorithm discovers a new edge of $Q(\bar{x})$ after every $O(\log m)$ iterations. Since the total number of edges in $Q(\bar{x})$ is $\text{poly}(n, m)$, the algorithm discovers all the edges in $\text{poly}(n, m)$ time, which leads to a surplus of zero for every component, and thus a departure from the loop. We can also bound the number of calls to FindNext in a similar manner. Lemma 4.14 (ii) implies that, at least one new edge is discovered between any two consecutive calls to FindNext. Therefore, the number of calls to FindNext is also $\text{poly}(n, m)$.

Showing that FindNext runs in $\text{poly}(n, m)$ time is straightforward using arguments very similar to the ones we used for the PriceIncrease algorithm, so we conclude by showing that, whenever $\text{PriceIncrease}(p, \Delta)$ is called in Algorithm 4, the values of p and Δ are such that PriceIncrease terminates in polynomial time. In particular, we show that the unspent budget in a Δ -allocation at prices p is at most $s\Delta$ where s is $\text{poly}(n, m)$ and, by Lemma 4.5, this implies that its running time is also $\text{poly}(n, m)$.

The initial call at Step 2 uses $\Delta = 1/(2\bar{m})$, so the unspent budget is at most $n \leq n2\bar{m}\Delta$. Similarly, in Step 6 PriceIncrease is called for (p, Δ) such that p supports a full 2Δ -allocation, so the unspent budget in a Δ -allocation is at most $m\Delta$. Finally, for Step 8 (i.e., after a call to FindNext) the unspent money is at most $2\bar{m}^2\Delta$, as argued next. To verify this, let p' be the prices returned by $\text{PriceIncrease}(p, \Delta)$. The unspent budget is at most the sum of the positive surpluses, which is at most $nS_{\max} \leq m\bar{S}_{\max} = 4m\bar{m}\Delta \leq 2\bar{m}^2 \square$

Acknowledgments. The second author would like to thank Paul Dütting, Zhiyi Huang, and Tim Roughgarden for stimulating discussions. This work was partly supported by NSF awards CCF-1217989 and CCF-1215965, CCF-1527568, and an ONR PECASE award.

References.

- [1] G. Amanatidis, E. Markakis, A. Nikzad, and A. Saberi. Approximation algorithms for computing maximin share allocations. In *ICALP*, 2015 (to appear).
- [2] M. Andrews, L. Qian, and A. L. Stolyar. Optimal utility based multi-user throughput allocation subject to throughput constraints. In *INFOCOM*, pages 2415–2424, 2005.
- [3] C. Annamalai, C. Kalaitzis, and O. Svensson. Combinatorial algorithm for restricted max-min fair allocation. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1357–1372, 2015.
- [4] A. Asadpour and A. Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. *SIAM J. Comput.*, 39(7):2970–2989, 2010.
- [5] A. Asadpour, U. Feige, and A. Saberi. Santa claus meets hypergraph matchings. *ACM Transactions on Algorithms*, 8(3):24, 2012.
- [6] N. Bansal and M. Sviridenko. The santa claus problem. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 31–40, 2006.
- [7] J. Barbanel. *The Geometry of Efficient Fair Division*. Cambridge University Press, 2004.
- [8] M. Bateni, M. Charikar, and V. Guruswami. Maxmin allocation via degree lower-bounded arborescences. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 543–552, 2009.
- [9] I. Bezáková and V. Dani. Allocating indivisible goods. *SIGecom Exch.*, 5(3): 11–18, Apr. 2005. ISSN 1551-9031.
- [10] S. Bouveret and M. Lemaître. Characterizing conflicts in fair division of indivisible goods using a scale of criteria. In *Proceedings of the International Conference on Autonomous Agents and Multi-agent Systems, AAMAS '14*, pages 1321–1328, 2014.
- [11] S. Brams and A. Taylor. *Fair Division: from cake cutting to dispute resolution*. Cambridge University Press, Cambridge, 1996.
- [12] E. Budish. The combinatorial assignment problem: Approximate competitive equilibrium from equal incomes. *Journal of Political Economy*, 119(6):1061 – 1103, 2011.
- [13] D. Chakrabarty, J. Chuzhoy, and S. Khanna. On allocating goods to maximize fairness. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 107–116, 2009.
- [14] Y. Chevaleyre, U. Endriss, S. Estivie, and N. Maudet. Reaching envy-free states in distributed negotiation settings. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1239–1244, 2007.
- [15] R. Cole and V. Gkatzelis. Approximating the nash social welfare with indivisible items. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 371–380, 2015.
- [16] R. Cole, V. Gkatzelis, and G. Goel. Positive results for mechanism design without money. In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13*, pages 1165–1166, 2013.
- [17] R. Cole, V. Gkatzelis, and G. Goel. Mechanism design for fair division: allocating divisible items without payments. In *ACM Conference on Electronic Commerce, EC '13, Philadelphia, PA, USA*, pages 251–268, 2013.
- [18] A. Darmann and J. Schauer. Maximizing nash product social welfare in allocating

- indivisible goods. 2014. URL <http://ssrn.com/abstract=2410766>.
- [19] N. Devanur, C. Papadimitriou, A. Saberi, and V. Vazirani. Market equilibrium via a primal-dual algorithm for a convex program. *JACM.*, 55(5):1–22, 2008.
 - [20] E. Eisenberg and D. Gale. Consensus of subjective probabilities: The pari-mutuel method. *Ann. Math. Stat.*, 30:165–168, 1959.
 - [21] U. Feige. On allocations that maximize fairness. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 287–293, 2008.
 - [22] A. Hylland and R. Zeckhauser. The Efficient Allocation of Individuals to Positions. *Journal of Political Economy*, 87(2):293–314, April 1979.
 - [23] M. Kaneko and K. Nakamura. The nash social welfare function. *Econometrica*, 47(2):pp. 423–435, 1979.
 - [24] F. P. Kelly. Charging and rate control for elastic traffic. *European Transactions on Telecommunications*, 8:33–37, 1997.
 - [25] E. Lee. APX-hardness of maximizing nash social welfare with indivisible items. *CoRR*, abs/1507.01159, 2015. URL <http://arxiv.org/abs/1507.01159>.
 - [26] R. J. Lipton, E. Markakis, E. Mossel, and A. Saberi. On approximately fair allocations of indivisible goods. In *Proceedings ACM Conference on Electronic Commerce (EC-2004), New York, NY, USA*, pages 125–131, 2004.
 - [27] H. Moulin. *Fair Division and Collective Welfare*. The MIT Press, 2003.
 - [28] J. Nash. The bargaining problem. *Econometrica*, 18(2):155–162, April 1950.
 - [29] N.-T. Nguyen, T. Nguyen, M. Roos, and J. Rothe. Computational complexity and approximability of social welfare optimization in multiagent resource allocation. *Autonomous Agents and Multi-Agent Systems*, 28(2):256–289, 2014. ISSN 1387-2532.
 - [30] T. Nguyen, M. Roos, and J. Rothe. A survey of approximability and inapproximability results for social welfare optimization in multiagent resource allocation. *Annals of Mathematics and Artificial Intelligence*, 68(1-3):65–90, 2013. ISSN 1012-2443.
 - [31] T. T. Nguyen and J. Rothe. Minimizing envy and maximizing average nash social welfare in the allocation of indivisible goods. *Discrete Applied Mathematics*, 179(0):54 – 68, 2014.
 - [32] N. Nisan, T. Roughgarden, É. Tardos, and V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA, 2007. ISBN 0521872820.
 - [33] J. B. Orlin. Improved algorithms for computing fisher’s market clearing prices: computing fisher’s market clearing prices. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC*, pages 291–300, 2010.
 - [34] A. Othman, C. H. Papadimitriou, and A. Rubinstein. The complexity of fairness through equilibrium. In *ACM Conference on Economics and Computation, EC ’14, Stanford , CA, USA*, pages 209–226, 2014.
 - [35] A. D. Procaccia and J. Wang. Fair enough: guaranteeing approximate maximin shares. In *ACM Conference on Economics and Computation, EC ’14, Stanford , CA, USA*, pages 675–692, 2014.
 - [36] S. Ramezani and U. Endriss. Nash social welfare in multiagent resource allocation. In *Agent-Mediated Electronic Commerce*, volume 59, pages 117–131. 2010.
 - [37] J. Robertson and W. Webb. *Cake-cutting algorithms - be fair if you can*. A K Peters, 1998. ISBN 978-1-56881-076-8.
 - [38] J. Uckelman and U. Endriss. Compactly representing utility functions using weighted goals and the max aggregator. *Artif. Intell.*, 174(15):1222–1246, 2010.
 - [39] H. Varian. Equity, envy, and efficiency. *Journal of Economic Theory*, 9(1):63–91,

1974.

[40] H. Young. *Equity*. Princeton University Press, 1995.

Appendix.

A.1. Proof of Lemma 4.13. The prices p in the input of $FindNext(p, \Delta)$ support a full Δ -allocation. In a full Δ -allocation at prices p , the total spending on each item j is at most $\min\{p_j + \Delta, 1\}$, and the total spending across all items is exactly equal to n , since all the budgets are spent fully. Therefore, $n \leq \sum_j \min\{p_j + \Delta, 1\}$, which implies that the total surplus at p is at most $m\Delta$:

$$\sum_C s(C, p) = n - \sum_j \min\{p_j, 1\} \leq \sum_j \min\{p_j + \Delta, 1\} - \sum_j \min\{p_j, 1\} \leq m\Delta.$$

When $FindNext(p, \Delta)$ is called, every discovered component has surplus at least $\frac{-\Delta}{4\bar{m}}$ (see Step 4 of Algorithm 4). Since there are at most m components with negative surplus (the ones containing items), the total surplus of these components is at least $\frac{-m\Delta}{4\bar{m}}$. Clearly, the highest positive surplus that a component could have is at most the total surplus minus the total surplus of the “negative” components, i.e.,

$$\max_C \{s(C, p)\} \leq \sum_C s(C, p) - \sum_{C: s(C, p) < 0} s(C, p) \leq m\Delta + \frac{m\Delta}{4\bar{m}} \leq \bar{m}\Delta.$$

We have $S_{\max} \leq \max_C \{s(C, p)\} \leq \bar{m}\Delta$, and $S_{\max} \geq \bar{S}_{\max}/2 = 2\bar{m}\Delta'$. Therefore, we conclude that $\Delta' \leq \Delta/2$. \square

A.2. Proof of Lemma 4.14 (i). The input to $FindNext$ is a price vector p and a value Δ with p supporting a full Δ -allocation. To show that $FindNext(p, \Delta)$ returns (p', Δ') with p' supporting a Δ' -allocation, we construct such an allocation using a full Δ -allocation at prices p .

We first consider the discovered components D whose prices do not change during the execution of $FindNext$. Since the prices in D do not change, this means that the prices and the surplus of all the components that have D in their reachable set does not change either. Therefore, since $\Delta' < \Delta/2$ (see Lemma 4.13), the amount of spending on each item need only be decreased, so we let the spending within these components in the (non-full) Δ' -allocation be at most as much as in the full Δ -allocation. The rest of the proof focuses on components whose surplus drops during the execution of $FindNext$.

For any discovered component D whose surplus decreases and becomes negative, or more negative, after the execution of $FindNext$, there exists some component C that caused this to happen. That is, D was part of the reachable set $R(C, p^c)$ (or becomes part of it during the execution of Step 14), which causes an increase of the prices in D . In particular, this component C is the one that defines the price p'_j of items $j \in D$ at Step 17 of the $FindNext$ algorithm.

Note that the spending capacity of each item j can be up to $p_j + \Delta'$, whereas the surplus computations disregard the additional capacity of Δ' per item. Nevertheless, the total additional capacity due to this Δ' per item is at most $m\Delta'$, i.e., at most $m\bar{S}_{\max}/(4\bar{m}) \leq \bar{S}_{\max}/8$ in total.

Also note that, for any component D whose item prices are defined by C in Step 17, the negative surplus is at most $S_{\max}/(4\bar{m})$. This is true because of Step 6 of the $FindNext$ algorithm, which stops the price increase of C before the surplus of D becomes less than $-s(C, p)/(4\bar{m})$. To be more precise, Step 15 actually stops the price increase when the surplus of C reaches S_{\max} , so the surplus of D may be strictly higher

than $-s(C, p)/(4\bar{m})$. Hence, since $s(C, p) \leq S_{\max}$ for all C (see Step 8), $s(D, p') \geq -S_{\max}/(4\bar{m})$ for each such component D . Since there are at most m components with negative surplus, the total negative surplus is at most $mS_{\max}/(4\bar{m}) \leq S_{\max}/8$.

On the other hand, the positive surplus of every component C that causes the surplus of other components $D \in R(C, p)$ to shrink is at least S_{\max} (see Step 15), which is at least $\bar{S}_{\max}/2$ (see Step 9). If we combine the two types of “unmet spending” that we analyzed above (that due to the discretization of the spending capacities and that due to the negative surpluses) the total amount is at most $\bar{S}_{\max}/8 + S_{\max}/8 \leq \bar{S}_{\max}/4$. Therefore, the total unmet spending is less than the surplus of C . To conclude the proof, we also show that up to $\bar{S}_{\max}/4$ of this positive surplus can in fact be redistributed across MBB edges so that a Δ' -allocation is reached.

Since $\bar{S}_{\max}/4 = \bar{m}\Delta'$ by Step 18, and $\bar{m}\Delta' \leq \bar{m}\Delta/2$ by Lemma 4.13, the total spending that needs to be covered is $\bar{S}_{\max}/4 \leq \bar{m}\Delta/2$. Therefore, the total spending that needs to be added in order to cover the unmet spending described above is at most $(\bar{m}\Delta)/2$. Now, note that the spending on *every* discovered edge in the full Δ -allocation is at least $2\bar{m}\Delta$, which is more than the *total* spending that needs to be added. We can therefore add the spending appropriately using alternating paths across the trees formed by the union of the discovered edges and the MBB edges which made D reachable from C during the execution of the FindNext algorithm. \square

A.3. Proof of Lemma 4.14 (ii). For $S_{\max} = 0$ the claim is clearly true. If $S_{\max} > 0$, let C be one of the discovered components with $S^C = S_{\max}$ in Step 8, and let \tilde{p} be the prices at which the price increase process for C stops in Step 7. Since $S_{\max} > 0$, we get $s(C, \tilde{p}) > 0$, and Step 6 implies that there exists some $D \in R(C, \tilde{p})$ with $s(D, \tilde{p}) \leq -s(C, \tilde{p})/(4\bar{m}) = -S_{\max}/(4\bar{m})$. Since $s(C, \tilde{p}) = S_{\max}$, the stopping condition of Step 15 is not satisfied before p^C becomes equal to \tilde{p} . Therefore, for every item $j \in D$, its price $p'_j := \max_C p_j^C$ is at least \tilde{p}_j . Hence, $s(D, p') \leq s(D, \tilde{p}) \leq -S_{\max}/(4\bar{m}) \leq -\Delta/(4\bar{m})$. \square