Ascending-Price Mechanism for General Multi-Sided Markets

Dvir Gilor, Rica Gonen and Erel Segal-Halevi

Abstract

We present an ascending-price mechanism for a multi-sided market with a variety of participants, such as manufacturers, logistics agents, insurance providers, and assemblers. Each deal in the market may consist of a combination of agents from separate categories, and different such combinations are simultaneously allowed. This flexibility lets multiple intersecting markets be resolved as a single global market. Our mechanism is obviously-truthful, strongly budget-balanced, individually rational, and attains almost the optimal gain-from-trade when the market for every allowed combination of categories is sufficiently large. We evaluate the performance of the suggested mechanism with experiments on real stock market data and synthetically produced data.

1 Introduction

The aim of this paper is to automatically arrange the trade in complex multi-lateral markets. As an example, consider a market for a certain kind of laptop computer, and assume for simplicity that it is made of only two types of components, e.g. four CPUs and two RAMs. Even in this simplified market, there may be several different categories of traders: 1. Buyers, who are interested in a laptop; 2. Laptop producers, who produce whole laptops; 3. CPU producers; 4. RAM producers; 5. Constructors, who construct a laptop from its parts; 6. Transporters, who take a laptop and bring it to an end consumer. A deal in this market can take one of two forms:

- A buyer buys a laptop from a laptop-producer, and asks a transporter to transport it to his/her place. This involves traders of categories 1, 2 and 6.
- A buyer buys four CPUs, two RAMs and a construction service, and has the final product transported. This involves traders of categories 1, 3, 4, 5 and 6.

In each category there may be many different traders, with potentially different utilities for participating in a deal. Typically, the value of a buyer is positive and the value of a producer or service-provider is negative. The main questions of interest for automatically arranging the trade is *who* will trade and *how much* they will pay (or receive). The answers to these questions should satisfy several natural requirements:

(1) *Individual rationality (IR)*: no agent should lose from participating: the amount paid by a trading agent should be at most as high as the agent's value (in particular, if the value is negative then the agent should receive money). A non-trading agent should pay nothing.

(2) *Weak budget balance (WBB)*: the total amount paid by all agents together should be at least 0, so that the market manager does not lose money. WBB allows the market-manager to gain money, and this surplus might consume most of the gain-from-trade, leaving little gain for the actual traders. This might drive traders away from the market. Therefore, we consider a stronger requirement called *strong budget balance (SBB)*: the total amount paid by all agents should be exactly 0.[1]

[1]The following procedure can be used to convert any WBB mechanism to an SBB mechanism: Add an initial stage to the mechanism in which a trader is chosen randomly and removed from the trade and market. After the mechanism allocation and pricing is finished, give the chosen trader all the surplus (if any). This procedure might encourage the arrival of agents who have nothing to do with the market (e.g. sellers with nothing to sell or buyers with no money), only because of the

(3) High *gain-from-trade (GFT)*: the GFT is the sum of values of all agents actively participating in the trade[2] For example, suppose a certain buyer values a laptop at 2000, the laptop-producer values it at -1500 (the cost of production is 1500), the CPU and RAM producers and constructor value their efforts at -200 each, and the transporter values the deal at -50 (the cost of transportation is 50). Then, the GFT from a deal involving categories 1, 2, 6 is $2000 - 1500 - 50 = 450$, and the GFT from a deal involving categories 1, 3, 4, 5, 6 is $2000 - 200 \cdot 4 - 200 \cdot 2 - 200 - 50 = 550$. Maximizing the GFT implies that the latter deal is preferred.

(4) *Truthfulness*: the agents' values are their private information. We assume that the agents act strategically to maximize their utility (assumed to be their value minus the price they pay). Truthfulness means that such a utility-maximizing agent reports his/her true valuation. A stronger requirement called *obvious truthfulness* [30] is that, for each agent, the lowest utility he may get by acting truthfully is at least as high as the highest utility he may get by acting non-truthfully.

These concepts are formally defined in Section **2**.

1.1 Related Work

The study of truthful market mechanisms started with [38]. He considered a market with only *one category* of traders (buyers), where the famous *second-price auction* attains all four desirable properties: IR, WBB, maximum GFT and truthfulness.

When there are *two caterogies* of traders (buyers and sellers), the natural generalization of Vickrey's mechanism is no longer WBB — it may run a deficit. Moreover, [33] proved that *any* mechanism that is IR, truthful and maximizes the GFT must run a deficit. The way out of this impossibility paradox was found by [32]. In his seminal paper, he presents the first *double auction* (auction for a two-category market) that is IR, WBB, truthful, and *asymptotically* maximizes the GFT. By asymptotically we mean that its GFT is at least $(1 - 1/k)$ of the optimal GFT, where k is the number of deals in the optimal trade. Thus, when k approaches infinity, the GFT approaches the optimum.

McAfee's mechanism has been extended in various ways. Particularly relevant to our setting is the extension by [1], with *multiple categories* of traders, arranged in a *linear supply chain*. Their model contains a single *producer* category, a single *consumer* category, and several *converter* categories. Each deal must involve a single producer, a single consumer, and a single agent of each converter category. In our laptop example, their model covers either a market with the chain 1,2,6 or a market with the chain 1,3,4,5,6, but not a market where both chains are possible. For this model, they present an auction mechanism that is IR, WBB, truthful, and attains asymptotically-optimal GFT

Recently, [27], [24] considered a multiple-category market in which, like [1]'s market, all deals must be of the same structure, which they call a "recipe". Their recipes are more general than the linear supply chains of [1], since they are not restricted to a producer-converters-consumer structure. They present auctions that are IR, SBB, truthful and asymptotically-optimal, but only for a single-recipe market.

Recently, [22] focus on a special case in which computing the optimal welfare is tractable. They assume that the agent categories can be arranged in a *forest* (an acyclic undirected graph), and each recipe is a path from a root to a leaf in that forest. Their recipes are more general than those of [27] and [24], as they are not restricted to a single-recipe market structure. The authors present auctions that are IR, SBB, truthful, and asymptotically-optimal, but only for binary recipe-forests.

Comparison to other supply-chain mechanisms is presented in Appendix **A.1**, and a survey of

chance to win all the surplus. We do not use this procedure in our proposed solutions. Much like [14], our solutions achieve SBB from direct-trade mechanisms — mechanisms that give/take money only to/from agents who actually participate in the trade.

[2]We note that, with non-additive GFT functions, even computing the optimal trade when all valuations are common knowledge is NP-hard problem, known as the welfare-maximization problem. Furthermore, it is NP-hard even when the the GFT functions are submodular; see, for example, [17].

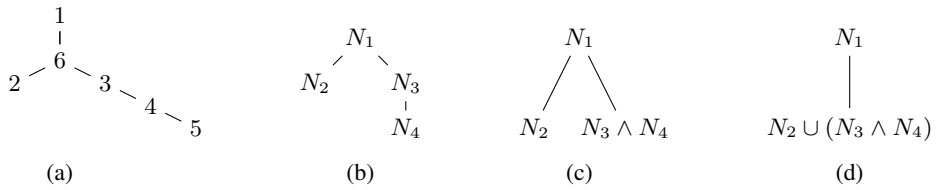


Figure 1: Examples of trees in a recipe-forest.

more recent works on two-sided markets is presented in Appendix **A.2**. More about obviously-truthful can be found in Appendix A.3.

1.2 Our Contribution

We study markets with multiple kinds of supply-chains which, following [27], [24], we call "recipes".

In a general multi-recipe market, computing the optimal trade — even without strategic considerations — is MAX-SNP-hard (see Appendix **E**). This means that, unless P=NP, there is no polytime algorithm that, given $\epsilon > 0$, computes a trade that attains $(1 - \epsilon)$ of the optimal GFT. In particular, it is unlikely that a mechanism that runs in polynomial time can be asymptotically-optimal.

Therefore, we focus on a special case in which computing the optimal welfare is tractable: We assume that the agent categories can be arranged in a *forest* (acyclic undirected graph), and each recipe is a path from a root to a leaf in that forest. Our laptop market corresponds to a forest with tree in Figure 1(a).

In Section G.1 we discuss the challenges in extending our results to recipe-sets that are not forests.

We present randomized ascending-prices mechanisms for markets based on recipe forests. Our mechanisms are IR, SBB and obviously-truthful. Moreover, all these properties hold *universally* — for every possible outcome of the randomization. The expected GFT of our mechanisms is asymptotically-optimal — it approaches the optimum when the optimal number of deals in all recipes approaches infinity (See Section 3.2 and Appendix B.2 for formal statements and proofs).

1.3 Paper Layout

Sections 2 presents the formal definitions. Section 3 and 4 in [22], which are also in Appendix B, presents a mechanism for the special case in which each trade requires either zero or one agents from each category, that is, the recipes are *binary*. Section 3 presents a mechanism for the more general case, in which each trade may require any non-negative integer number of agents from each category. The mechanism of [22], which is also in Appendix B, is a special case of the mechanism of Section 3; we present it in a separate section as the approximation guarantee of the binary mechanism, and it is not a special case of the approximation guarantee of the general mechanism. The general mechanism computation and proof techniques for reaching a GFT approximation bound are quite different and are more involved.

Appendix D presents some simulation experiments evaluating the performance of our auctions. Appendix G concludes by illustrating challenges in extending our current model and providing future work directions.

An open-source implementation of our auctions, including example runs and experiments, is available at https://github.com/dvirg/auctions.

2 Formal Definitions

2.1 Agents and Categories

A *market* is defined by a set of *agents* grouped into different *categories*. N is the set of agents, G is the set of agent categories, and N_g is the set of agents in category $g \in G$. The categories are pairwise-disjoint,[3] so $N = \sqcup_{g \in G} N_g$.

Each deal in the market requires a certain combination of traders. We call a subset of agents that can accomplish a single deal a *procurement-set* (PS).

A *recipe* is a vector of size $|G|$, denoted by $\mathbf{r} := (r_g)_{g \in G}$, where $r_g \in \mathbb{Z}_+$ for all $g \in G$. It describes the number of agents of each category that should be in each PS: each PS should contain r_1 agents of category 1, r_2 agents of category 2, and so on. If $r_g \in \{0, 1\}$ for all $g \in G$, then $\mathbf{r}$ is called a *binary recipe*. Otherwise, it is called an *integer recipe*. The set of recipes available in the market is denoted by R.

In the market of [32] each deal requires one buyer and one seller, so there is a single binary recipe and $R = \{(1, 1)\}$. In our initial laptop-market example there are two recipes and $R = \{(1, 1, 0, 0, 0, 1); (1, 0, 4, 2, 1, 1)\}$. The first one is a binary recipe corresponding to deals with a buyer, a producer and a transporter, and the second one in an integer recipe corresponding to deals with a buyer, four CPU producers, two RAM producers, a constructor and a transporter.

Each agent $i \in N$ has a *value* $v_i \in \mathbb{Z}$, which represents the material gain of an agent from participating in the trade. It may be positive, negative or zero. In a two-sided market for a certain good, the value of a buyer is typically positive, while the value of a seller is typically negative and represents the cost of producing the good. However, our model is general and allows the values of different agents in the same category to have different signs.

For simplicity, we assume that all the v_i are integer numbers, e.g., all valuations may be given in cents. We also assume that there are publicly known bounds on the possible valuations: for some sufficiently large V, $-V < v_i < V$ for all $i \in N$.

The agents are *quasi-linear in money*: the utility of agent i participating in some PS and paying p_i is $u_i := v_i - p_i$.

2.2 Recipe forests

Recall that a *forest* is an acyclic graph, composed of one or more *trees*; a *rooted forest* is a forest in which, in each tree, one vertex is denoted as its *root*.

Definition 2.1. A recipe-set R is called a *recipe-forest* if there exists a rooted forest T in which the set of nodes is G, and each recipe $\mathbf{r} \in R$ corresponds to a path P from the root of some tree in T to a leaf of that tree (that is, $r_g \geq 1$ for each $g \in P$ and $r_g = 0$ for each $g \notin P$).[4]

We use the same letter g to denote both the category index and the corresponding node in T. We assume that every $g \in G$ appears in all recipes with the same multiplicity, i.e., for every recipes $\mathbf{r}, \mathbf{r}' \in R$, if $r_g > 0$ and $r'_g > 0$ then $r'_g = r_g$.[5]

As an example, the set $R = \{(1, 2, 0, 0), (1, 0, 1, 2)\}$ is a recipe-forest with a single tree shown in Figure 1(b). The root category is N_1. The recipe $(1, 2, 0, 0)$ corresponds to a path from N_1 to the leaf N_2. The recipe $(1, 0, 1, 2)$ corresponds to a path from N_1 through N_3 to the leaf N_4.

A market with a single recipe is a special case of a recipe-forest with a single tree that is a path. Note that several different trees T may correspond to the same recipe-set R. For example, the

[3] In Section G.2 we discuss why removing this assumption is challenging.

[4] We can also assume, without loss of generality, that every path from a root to a leaf in T corresponds to a recipe in R. The reason is that, if some path P in T does not correspond to a recipe in R, then the leaf category of P can be removed from the market without affecting the trade, since it does not participate in any recipe. Removing leaves can be repeated until all remaining root–leaf paths in T correspond to recipes in R.

[5] In Section G.3 we discuss why removing this assumption is challenging.

singleton recipe-set $R = \{(1, 1)\}$ corresponds to the tree in which N_1 is the root and N_2 is the leaf, and also to the tree in which N_2 is the root and N_1 is the leaf.

Given a fixed rooted forest T, we use the following notation:

- CHILDREN(g) := the child nodes of the node g in its tree.
- LVS(g) := the leaf descendants of the node g in its tree (if g is a leaf then LVS(g) = $\{g\}$).
- PATH($g_1 \to g_2$) := the nodes in the unique path from g_1 to its descendant g_2, inclusive.
- HEIGHT(g) := the largest distance between the node g and a leaf of its tree. The height of a leaf is 0.
- DEPTH(g) := the unique distance between the node g and the root of its tree. The depth of a root is 0.
- MAXDEPTH := $\max_{g \text{ is a leaf in } T}$ DEPTH(g).
- WEIGHTD(g) := $\sum_{g' \in \text{PATH}(g \to root)} r_{g'}$ = the distance between g and the root of its tree (including the root), weighted by the integer quantities $r_{g'}$. The weighted depth of a root category g_0 is r_{g_0}.
- MAXWD := $\max_{g \text{ is a leaf in } T}$ WEIGHTD(g).

Tables summarizing the notations used in this paper can be found in Tables 5, 6 and 7 in Appendix.

2.3 Trades and Gains

The *gain-from-trade* of a procurement-set S, denoted $GFT(S)$, is the sum of all agent values $v_i \in S$:

$$\text{GFT}(S) := \sum_{i \in S} v_i.$$

In a standard two-sided market, the GFT of a PS with a buyer b and a seller s is $v_b - v_s$, where v_s is the seller's value for the sold item, since the seller's value for participating in the trade is $-v_s$.

Given a market (N, G, R), a *trade* is a collection of pairwise-disjoint procurement-sets. I.e, it is a collection of agent subsets, $S_1, \ldots, S_k \subseteq N$, such that for each $j \in [k]$, the composition of agents in S_j corresponds to some recipe $\mathbf{r} \in R$. The total GFT is the sum of the GFT of all procurement-sets participating in the trade:

$$\text{GFT}(S_1, \ldots, S_k) := \sum_{j=1}^{k} \text{GFT}(S_j)$$

A trade is called *optimal* if its GFT is maximum over all trades.

The *value* of agent i given trade $\mathbf{S} = (S_1, \ldots, S_k)$, denoted $v_i(\mathbf{S})$, is either v_i or 0: it is v_i if $i \in S_j$ for some $j \in [k]$, and 0 otherwise.

2.4 Mechanisms

The definitions below cover only the notions used in the present paper. For a more complete treatment of mechanisms and their properties see [34].

A *deterministic direct mechanism* is a function that takes as input a vector $\mathbf{b}$ containing agent bids, and returns as output a trade $\mathbf{S}(\mathbf{b})$ and a price-vector $\mathbf{p}(\mathbf{b})$. The *utility* of each agent i, given a fixed deterministic mechanism and a bid vector $\mathbf{b}$, is $u_i(\mathbf{b}) := v_i(\mathbf{S}(\mathbf{b})) - p_i(\mathbf{b})$.

A deterministic direct mechanism is *truthful* if the utility of every agent i is maximized when the agent bids v_i, for any fixed bids of the other agents. Formally, for every vector $\mathbf{b} = (b_1, \ldots, b_n)$, denote by $\mathbf{b}|_{b_i \leftarrow x}$ the vector $(b_1, \ldots, b_{i-1}, x, b_{i+1}, \ldots, b_n)$. A mechanism is truthful if for every agent i and vector $\mathbf{b}$: $u_i(\mathbf{b}|_{b_i \leftarrow v_i}) \geq u_i(\mathbf{b})$.

A deterministic direct mechanism is *individually-rational (IR)* if the utility of every agent i when the agent bids v_i is at least 0, regardless of the bids of the other agents: $u_i(\mathbf{b}|_{b_i \leftarrow v_i}) \geq 0$.

A *randomized direct mechanism* is a lottery over deterministic direct mechanisms. In other words, it is a mechanism in which the functions $\mathbf{S}$ and $\mathbf{p}$ may depend not only on the bids but also on some random variables.

A randomized direct mechanism is called *universally-truthful* if it is a lottery over truthful deterministic direct mechanisms. In a universally-truthful randomized mechanism, the utility of agent i is maximized when the agent bids v_i, regardless of the bids of the other agents, and regardless of the random variable values.[6] Similarly, a randomized direct mechanism is *universally-IR* if it is a lottery over IR deterministic direct mechanisms.

A direct mechanism is called *obviously truthful* if for every agent i and vectors $\mathbf{b}, \mathbf{b}'$: $u_i(\mathbf{b}|_{b_i \leftarrow v_i}) \geq u_i(\mathbf{b}')$. In other words, the lowest utility the agent can get when reporting truthfully is at least as high as the highest utility the agent can get when reporting untruthfully, where "lowest" and "highest" are w.r.t. all possible reports of the other agents. This is a very strong property that is not satisfied by non-trivial direct mechanisms. However, an analogous property is satisfied by some *sequential mechanisms*, described next.

In a *deterministic sequential mechanism*, at each time, an agent has to choose a *response* from a pre-specified set of responses. In order to give meaning to the notion of truthfulness, we assume that the "response" is an answer to a query on the agent's value: at time t, the designer presents a function q_t to some agent i, and the agent is expected to reveal $q_t(v_i)$. Our mechanisms will only use Boolean functions such as "is $v_i > 2$?". Based on the agents' answers so far, the auctioneer may decide to continue asking queries, or to end the mechanism. When the mechanism ends, the auctioneer examines the vector of answers $\mathbf{a}$, and determines the trade $\mathbf{S}(\mathbf{a})$ and the price-vector $\mathbf{p}(\mathbf{a})$.

Given an answer vector $\mathbf{a}$ and an agent i, denote by $\mathbf{a}|_{a_i \leftarrow x}$ the vector in which the answer of agent i to any query q_t is $q_t(x)$ (and the answers of other agents remain as in $\mathbf{a}$). A deterministic sequential mechanism is called *obviously truthful* if, at any step during the execution, and for any two vectors $\mathbf{a}$ and $\mathbf{a}'$ consistent with the history of answers up to the current step: $u_i(\mathbf{a}|_{a_i \leftarrow v_i}) \geq u_i(\mathbf{a}')$. In other words, the lowest utility the agent can get by answering truthfully, according to v_i, is at least as high as the highest utility he can get by answering untruthfully.

A deterministic direct mechanism is a special case of a deterministic sequential mechanism, in which there is only one step of queries, and the queries are "what is your value?". If such a mechanism is obviously-truthful, then it is also truthful (set $\mathbf{a} = \mathbf{a}' = \mathbf{b}$ in the definition of obvious-truthfulness).

3 Integer Recipes

This section extends Sections 3 and 4 in [22] which also appears in Appendix B, by allowing the recipes in the forest to be arbitrary vectors of non-negative integers, rather than binary vectors. For each category g there is an integer $r_g \geq 0$, and every PS of $r \in R$ such that $g \in G$ must contain exactly r_g traders from this category. We assume that, for every category g, r_g is the same for all recipes $\mathbf{r}$ in which $r_g > 0$. We call this number the *multiplicity* of category g.

The mechanism of Sections 3 and 4 in [22] is a special case of the mechanism of Section 3. We split them into two different sections because in the binary case we could prove a stronger

[6] Universal truthfulness is stronger than other notions of truthfulness studied in the literature, such as truthfulness-in-expectation. These weaker truthfulness notions are not covered in the present paper.

Algorithm 1 Finding the optimal GFT — integer recipes.

Input: A set of categories G, a set of traders N_g for all $g \in G$, and a recipe-forest R based on a forest T. For each agent $i \in \cup_g N_g$, the value v_i is public knowledge.

Output: Optimal trade in the market.

1. If T has a single vertex g:
   Return $\{V_i \in N_g | \text{sum}_{v_j \in V_i}(v_j) \geq 0\}$ — all subsets of N_g's agents with non-negative value sums.
2. Else, if T has two roots without children g_l and g_s:
   Do a horizontal contraction of g_l into g_s. Go back to step 1.
3. Else, if there is an arbitrary deepest leaf g_l that is a single child of its parent g_p:
   Do a vertical contraction of g_l into g_p. Go back to step 1.
4. Else, there is a leaf g_l with a sibling leaf g_s:
   Do a horizontal contraction of g_l into g_s. Go back to step 1.

Table 1: An example market. Boldface is optimal trade

Category g	Traders per deal r_g	Agents' values
N_1: buyers	1	**19, 18, 17, 13,** 6, 2
N_2: sellers	2	**-2, -2, -3, -4**, -5, -8
N_3: A-producers	1	**-1, -3,** -5, -7
N_4: B-producers	2	**-1, -2, -3, -4,** -6, -8

approximation ratio. In Appendix F we present examples showing that the approximation guarantee of the binary case is not true in the general case.

The concepts and notations introduced in the binary case should be adapted to the general case by weighting the quantity related to each category g by its multiplicity r_g. In particular:

- The sum of prices in each recipe is weighted: Prices-sum$(\mathbf{r}) := \sum_{g \in G} p_g \cdot r_g$.
- The depth of a node g is $\text{WEIGHTD}(g) := \sum_{g' \in \text{PATH}(g \to root)} r_{g'}$. The notation MAXWD (maximum weighted depth) is used to compute the initial price-sum for all paths.
- The quantity m_g is now $|M_g|$ divided by r_g, so that it represents the number of deals that can be done using the agents remaining in M_g. Note that m_g is not necessarily an integer number, so we round it down to the nearest integer.
- The notations 'cheap' and 'expensive' are determined based on sets of r_g agents of each category, rather than individual agents.

The notation differences between Sections 3 and 4 of [22] and Section 3 can be viewed in Tables 6 and 7 in Appendix.

3.1 Computing an Optimal Trade

We first present an algorithm for computing the optimal trade assuming all values are known. We illustrate the algorithm on the market in Table 1.

Similarly to [22] (and Appendix B), The algorithm is based on contracting the recipe-forest down to a single node. In addition to vertical and horizontal contractions, we need a third kind of contraction: an internal contraction.

An **internal contraction** acts on an individual category node $g \in G$. It converts agent values in the category to sets of size r_g in the following way. Sort all agents' values in descending order such that $v_1 \geq v_2 \geq \ldots \geq v_{m_v}$. Construct the sets $(v_1, \ldots, v_{r_g}) \geq (v_{r_g+1}, \ldots, v_{2r_g}) \geq \ldots \geq (\ldots, v_{m_v})$. If the last set $(\ldots, v_{m_v})$ contains less than r_g values, then remove it. For example, an

Algorithm 2 Ascending prices mechanism — integer recipes — main loop.

Input: A market N, a set of categories G and a recipe-forest R.
Output: Strongly-budget-balanced trade.
1. *Initialization:* Let $M_g := N_g$ for each $g \in G$. Using Equation (1), set an initial price p_g for each $g \in G$.
2. Using Algorithm 3, select a set $G^* \subseteq G$ of categories.
3. For each $g^* \in G^*$, ask each agent in $i \in M_{g*}$ whether $v_i > p_{g*}$.
   (a) If an agent $i \in M_{g*}$ answers "no", then remove i from M_{g*} and go back to step 1.
   (b) If all agents in M_{g*} for all $g^* \in G^*$ answer "yes", then for all $g^* \in G^*$, let $p_{g*} := p_{g*} + 1/r_{g^*}$.
   (c) If after the increase $\sum_{g \in G} p_g \cdot r_g = 0$ for some $\mathbf{r} \in R$, then go on to step 4.
   (d) else go back to step 3.
4. Determine final trade using Algorithm 4.

Algorithm 3 Finding a set of prices to increase — integer recipes.

Input: A set of categories G, a set of remaining traders M_g for all $g \in G$, and a recipe-forest R based on a forest T.
Output: A subset of G denoting categories for price-increase.
0. *Initialization*: For each category $g \in G$, let $m_g := |M_g|/r_g$.
1. If T contains two or more trees,
   Recursively run Algorithm 3 on each individual tree T'; Denote the outcome by $I_{T'}$.
   Return $\bigcup_{T' \in T} I_{T'}$.
2. Let g_0 be the category at the root of the single tree. Let $c_{g_0} := \sum_{g' \in \text{CHILDREN}(g_0)} \lfloor m_{g'} \rfloor$.
3. If $m_{g_0} > c_{g_0}$ [or g_0 has no children at all], then return the singleton $\{g_0\}$.
4. Else ($c_{g_0} \geq m_{g_0}$), for each child g' of g_0:
   Recursively run Algorithm 3 on the sub-tree rooted at g'; Denote the outcome by $I_{g'}$.
   Return $\bigcup_{g' \in child(g_0)} I_{g'}$.

internal contraction on N_4 in Table 1 gives $\{(-1,-2),(-3,-4),(-6,-8)\}$ since $r_4 = 2$, and on N_3 it gives $\{(-1),(-3),(-5),(-7)\}$ since $r_3 = 1$.

The vertical and horizontal contraction work similarly to the binary case. In the market described by Table 1, a vertical contraction of N_3 and N_4 yields $N_3 \wedge N_4$, which contains the values $\{((-1),(-1,-2)),((-3),(-3,-4)),((-5),(-6,-8))\}$ = $\{(-1,-1,-2),(-3,-3,-4),(-5,-6,-8)\}$. a horizontal contraction of N_2 with $N_3 \wedge N_4$ yields $N_2 \cup (N_3 \wedge N_4)$, which contains the values $\{(-1,-1,-2),(-2,-2),(-3,-4),(-3,-3,-4),(-5,-8),(-5,-6,-8)\}$. A vertical contraction of the latter category with N_1 yields: $\{(19,-1,-1,-2),\ (18,-2,-2),\ (17,-3,-4),\ (13,-3,-3,-4),\ (6,-5,-8),\ (2,-5,-6,-8)\}$. The optimal trade in that is the set of all deals with positive values, which in this case contains four deals with values $\{15,14,10,3\}$. This corresponds to an optimal trade with $k = 4$ deals:

- Buyer 19, A-producer -1, B-producers $-1,-2$;
- Buyer 18, sellers $-2,-2$;
- Buyer 17, sellers $-3,-4$;
- Buyer 13, A-producer -3, B-producers $-3,-4$.

The process is summarized as **Algorithm 1**.

Algorithm 4 Determining a feasible trade — integer recipes.

Input: A set of categories G, a set of remaining traders M_g for all $g \in G$, and a recipe-forest R based on a forest T.
Output: A set of PSs with remaining traders, each of which corresponds to a recipe in R.
1. Do a randomized internal contraction on all $g \in G$.
2. If T has a single vertex g, then return M_g — the set of traders remaining in category g.
3. If T has two roots without children g_l and g_s:
   Do a horizontal contraction of g_l into g_s. Go back to step 2.
4. Otherwise, pick an arbitrary deepest leaf category $g_l \in T$.
5. If g_l is a single child of its parent $g_p \in T$:
   Perform a randomized vertical contraction of g_l and g_p. Go back to step 2.
6. Otherwise, g_l has a sibling leaf $g_s \in T$:
   Perform a horizontal contraction of g_l into g_s. Go back to step 2.

3.2 Ascending Auction Mechanism

The ascending-price auction for integer recipes is presented as **Algorithm 2**. As in the binary case, for each category g, the auctioneer maintains a price p_g, and a subset $M_g \subseteq N_g$ of all traders whose value is higher than p_g. At each iteration, the auctioneer chooses a subset of the categories and increases their prices. In contrast to the binary case, the price-increase is not the same in all categories: the price of category g is increased by $1/r_g$. The reason is that the price-sum of every recipe $\mathbf{r}$ is a weighted sum of the category prices; increasing p_g by $1/r_g$ guarantees that the price-sum of every recipe increases by 1, so that it does not skip any integer value.

After each increase, the auctioneer asks each agent in turn whether their value is still higher than the price. An agent who answers "no" is permanently removed from the market. After each increase, the auctioneer computes the *weighted* sum of prices of the categories in each recipe: $\text{Prices-sum}(\mathbf{r}) := \sum_{g \in G} p_g \cdot r_g$. When this sum equals 0, the auction ends and the remaining agents trade in the final prices.

Similarly to the binary case, we explain (a) how the prices are initialized, (b) how the set of prices to increase is selected, and (c) how the final trade is determined.

(a) The prices are initialized as follows:

$$p_g := \begin{cases} -V & g \text{ is not a leaf} \\ -V \cdot (\textsc{MaxWD} - \textsc{WeightD}(g) + r_g)/r_g & g \text{ is a leaf} \end{cases} \tag{1}$$

This guarantees that the weighted price-sum in any path from the root to a leaf is the same: $-V \cdot$ MAXWD.

(b) The set of prices to increase is selected by **Algorithm 3**. In contrast to the binary case, the selection is based on the number of agents of each category g who are currently in the market, divided by r_g.[7] We denote this number by $m_g := |M_G|/r_g$.

Denote the root category of a tree by g_0. The algorithm first compares m_{g_0} to c_{g_0}, defined as the sum of the $\lfloor m_g \rfloor$ (m_g rounded down to the nearest integer[8]) for all children g of g_0. If m_{g_0} is larger, then the price selected for increase is the price of g_0; Otherwise (c_{g_0} is larger or equal), the prices to increase are the prices of some of its descendants' categories: for each child category, Algorithm 3 is used recursively to choose a subset of prices to increase, and all returned sets are combined. It is easy to prove by induction that the resulting subset contains exactly one price for each path from a root to a leaf. Therefore, if each price in a category g in the subset is increased

[7] Note that there is no division-by-zero issue, since when $r_g = 0$ there is no node of such g in the recipe-forest.

[8] Instead of rounding m_g downwards, we can keep m_g not rounded, or round it upwards. In Appendix F, we show for each of these three options, a market in which the approximation ratio depends on $|R|$, and all these approximation ratios are asymptotically similar (when $|R|$ is large).

simultaneously by $1/r_g$, then the weighted price-sum in all recipes increases simultaneously by one unit, so the price-sum in all recipes remains equal. Eventually the weighted price-sum reaches 0, and the auction stops.

(c) Once the auction ends, the final trade is computed using **Algorithm 4**.

The process is similar to the computation of the optimal trade, but the internal contraction operation is replaced with a *randomized internal contraction* operation. For each $g \in G$, denote $\bmod_g := |M_g|$ modulo r_g. Choose $\bmod_g$ agents uniformly at random and remove them from M_g. Then perform an internal contraction with the remaining agents. Note that after the internal contraction, all values $m_g = |M_g|/r_g$ are integers.

Similarly, the vertical contraction operation is replaced with a *randomized vertical contraction*. A leaf that is a single child is combined with its parent in the following way. Denote the leaf and parent category by l and p respectively, Let $m_{\min} := \min(m_l, m_p) =$ the integer number of procurement-sets that can be constructed from the agents in both categories. For each $g \in \{l, p\}$ if $m_g > m_{\min}$ then choose $m_g - m_{\min}$ sets of agents uniformly at random and remove them from M_g. Then perform a vertical contraction with the remaining sets of agents.[9].

Horizontal contractions can be performed deterministically, as no traders should be removed. The process of determining the final trade is summarized as **Algorithm 4**.

3.3 Example Run

We illustrate Algorithm 2 using the example in Table 1, where the recipe set is $R = \{(1,2,0,0),(1,0,1,2)\}$ and the recipe-forest contains the single tree shown in Figure 1(b). The execution is shown in Table 2.

Step 1 The weighted depths of the categories are $1, 3, 2, 4$, so MAXWD $= 4$. The initial prices determined by (1) are $-V, -(3/2)V, -V, -V$, and the weighted price-sum of each recipe is $-4V$.

Step 2 The categories whose price should be increased are determined using Algorithm 3. Initially, the numbers of remaining traders are $6, 6, 4, 6$. So $m_1 = 6, m_2 = 6/2, m_3 = 4, m_4 = 6/2$. Initially the algorithm compares m_1 to $\lfloor m_2 \rfloor + \lfloor m_3 \rfloor$. Since $6/1 < \lfloor 6/2 \rfloor + \lfloor 4/1 \rfloor$, the price of the root category (the buyers) is not increased, and the algorithm recursively checks the subtrees rooted at $g = 2$ and $g = 3$. In the former, there is only one category so it is returned; in the latter, there is one child $g = 4$. Since $m_3 > \lfloor m_4 \rfloor$, the parent $g = 3$ is selected. The final set of prices to increase is $\{p_2, p_3\}$. If the counts were $m_1 = 6, m_2 = 6/2, m_3 = 2, m_4 = 4/2$ instead, then the set of prices to increase would be $\{p_1\}$. Note that in both cases, a single price is increased in each recipe.

Step 3 The auctioneer increases the prices of each category $g^* \in G^*$ by $1/r_{g^*}$, until one agent of some category $g^* \in G^*$ indicates that his value is not higher than the price, and leaves the trade. In the example, the first agent who answers "no" is A-producer -7. While p_3 has increased to -7, p_2 has increased to $-V - (7/2)$ (it was incremented at steps of $1/r_2 = 1/2$), so the price-sum in each recipe remains the same: $-7 - 3V$. After A-producer -7 is removed, we return to step 2 to choose a new set of prices to increase. The algorithm keeps executing steps 2 and 3 as described in Table 2. Finally, while the algorithm increases p_1, and before buyer 13 exits the trade, the weighted price-sum in all recipes becomes 0 and the algorithm proceeds to step 4.

Step 4 The final trade is determined by Algorithm 4. First, a randomized internal contraction is done on all nodes, which uniformly at random removes surplus agents. In our example, there are surplus agents in buyers, sellers and B-producers categories, so we remove one buyer, one seller, and

[9] Alternatively, we can select $(m_g - m_{\min}) \cdot r_g$ agents uniformly at random and remove them. Then shift the remaining agents to fill in the 'holes' in sets. After the shift, the last $m_g - m_{\min}$ sets should be empty, so we remove them

Table 2: Execution of Algorithm 2 on market from Table 1

$\lvert M_i\rvert$	Compare	G^*	Price-increase stops when —	Updated prices	Price-sum
			Initial prices:	$-V, -3V/2, -V, -V$	$-4V$
$6, 6, 4, 6$	$6/1 \leq \lfloor 6/2 \rfloor + \lfloor 4/1 \rfloor \; ; 4/1 > \lfloor 6/2 \rfloor$	$2, 3$	A-producer -7 exits	$-V, -V-(7/2), -7, -V$	$-7-3V$
$6, 6, 3, 6$	$6/1 \leq \lfloor 6/2 \rfloor + \lfloor 3/1 \rfloor \; ; 3/1 \leq \lfloor 6/2 \rfloor$	$2, 4$	B-producer -8 exits	$-V, -(23/2), -7, -8$	$-23-V$
$6, 6, 3, 5$	$6/1 \leq \lfloor 6/2 \rfloor + \lfloor 3/1 \rfloor \; ; 3/1 > \lfloor 5/2 \rfloor$	$2, 3$	A-producer -5 exits	$-V, -(21/2), -5, -8$	$-21-V$
$6, 6, 2, 5$	$6/1 > \lfloor 6/2 \rfloor + \lfloor 2/1 \rfloor$	1	buyer 2 exits	$2, -(21/2), -5, -8$	-19
$5, 6, 2, 5$	$5/1 \leq \lfloor 6/2 \rfloor + \lfloor 2/1 \rfloor \; ; 2/1 \leq \lfloor 5/2 \rfloor$	$2, 4$	B-producer -6 exits	$2, -(17/2), -5, -6$	-15
$5, 6, 2, 4$	$5/1 \leq \lfloor 6/2 \rfloor + \lfloor 2/1 \rfloor \; ; 2/1 \leq \lfloor 4/2 \rfloor$	$2, 4$	seller -8 exits	$2, -8, -5, -(11/2)$	-14
$5, 5, 2, 4$	$5/1 > \lfloor 5/2 \rfloor + \lfloor 2/1 \rfloor$	1	buyer 6 exits	$6, -8, -5, -(11/2)$	-10
$4, 5, 2, 4$	$4/1 \leq \lfloor 5/2 \rfloor + \lfloor 2/1 \rfloor \; ; 2/1 \leq \lfloor 4/2 \rfloor$	$2, 4$	B-producer -4 exits	$6, -(13/2), -5, -4$	-7
$4, 5, 2, 3$	$4/1 \leq \lfloor 5/2 \rfloor + \lfloor 2/1 \rfloor \; ; 2/1 > \lfloor 3/2 \rfloor$	$2, 3$	A-producer -3 exits	$6, -(11/2), -3, -4$	-5
$4, 5, 1, 3$	$4/1 > \lfloor 5/2 \rfloor + \lfloor 1/1 \rfloor$	1	price-sum crosses 0	$11, -(11/2), -3, -4$	0

one B-producer all at random. Then we convert the agents of each category into sets of agents. After this, a randomized vertical contraction is done between the A-producers and B-producers. These categories are combined into a single set of three agents, one A-producer agent and two B-producers agents. Next, a horizontal contraction is done between the set of producers and the remaining two sets of two sellers each. Finally, a randomized vertical contraction is done between this combined category and the buyers' category. Since there are 4 remaining buyers, but only 3 sets in the child category, one of the buyers is chosen at random and removed. In conclusion, three deals are made: two deals follow the recipe $(1, 2, 0, 0)$ and involve a buyer and two sellers, and one deal follows the recipe $(1, 0, 1, 2)$ and involves a buyer, an A-producer and two B-producers.

3.4 Ascending Auction Properties

Due to space constraints, most proofs are in Appendix C.

Analogously to the binary case, we first ensure that the weighted price-sum along each path from the same node to a leaf is constant. We also ensure that it is an integer.

Lemma 3.1. *Throughout Algorithm 2, for any category $g \in G$, the weighted price-sum along any path from g to a leaf is an integer, and it is the same for all these paths.*

The strategic and economic properties of the auction are summarized in the following theorem.

Theorem 3.2. *Algorithm 2 is universally strongly-budget-balanced, individually-rational and obviously truthful.*

The proof is identical to Theorem B.2 and we omit it.

Lemma 3.3. *For all non-leaf categories $g \in G$, $c_g \leq m_g \leq c_g + 1$.*

We now adapt the definitions and lemmas regarding cheap and expensive paths from binary to general recipes.

Definition 3.4. Given a price-vector $\mathbf{p}$, a subset $G' \subseteq G$ is :
(a) *Cheap* — if $p_g \leq \min_{v_i \in V_{g,k_g+1}}(v_i)$ for all $g \in G'$;
(b) *Expensive* — if $p_g \geq \max_{v_i \in V_{g,k_g}}(v_i)$ for all $g \in G'$;

Similarly to the binary case, in a cheap path, the prices are sufficiently low to allow the participation of agents not from the optimal trade (those in V_{g,k_g+1}), while in an expensive path, the prices are sufficiently high to allow the participation of agents only from the optimal trade (not including those in V_{g,k_g}).

Lemma 3.5. *Let g_1, g_2 be two children of the same node $g_p \in T$. There cannot be simultaneously a cheap path from g_1 to a leaf and an expensive path from g_2 to a leaf.*

The proof is identical to that of Lemma B.7 so we omit it.

Recall that $\text{LVS}(g)$ is the set of leaf nodes that are g's descendants.

Lemmas 3.6 - 3.9 show some cases when Cheap and Expensive paths can and cannot exist in certain forest-trees. These lemmas are then used to prove Lemma 3.10, which is then used to prove our main theorem. Proofs of Lemmas 3.6 - 3.9 appear in Appendix C.

Lemma 3.6. *If $m_g < k_g - |\text{LVS}(g)|$ for some category $g \in G$, then there is an expensive path from g to a leaf.*

Lemma 3.7. *If $m_g \geq k_g + 1$ for some $g \in G$, then there is a cheap path from g to a leaf.*

Lemma 3.8. *If $m_g \leq k_g - 1$ for some $g \in G$, and there is an expensive path from g to a leaf, and Algorithm 3 decides to increase the price of g or a descendant of g, then, even before the increase, there is an expensive path from the root to g.*

Lemma 3.9. *If $m_g \geq k_g + |R| - |\text{LVS}(g)| + 1$ for some $g \in G$, then there is a cheap path from the root to a leaf (through g).*

Lemma 3.10. *When Algorithm 2 ends, $k_g - |\text{LVS}(g)| \leq m_g \leq k_g + |R|$ for all $g \in G$.*

Proof. The proof is by contradiction.

First, suppose that $m_g > k_g + |R|$ for some $g \in G$. Then $m_g > k_g + |R| - \text{LVS}(g) + 1$. By Lemma 3.9, there is a cheap path from the root to a leaf; denote the set of categories along this path by G'. By definition of a cheap path, $p_g \leq \min_{v_i \in V_{g,k_g+1}}(v_i)$ for all $g \in G'$. So the sum of prices of categories $g \in G'$ is at most the GFT of a non-optimal PS, which is negative. As long as the price-sum is negative, the algorithm does not terminate.

Second, suppose that $m_g < k_g - |\text{LVS}(g)|$ for some $g \in G$. Since at most a single agent is removed in each iteration, this means that the algorithm decided to increase the price of g while m_g was equal to $k_g - |\text{LVS}(g)|$. By Lemma 3.6 and Lemma 3.8, at that point there existed an expensive path from the root to a leaf; denote the set of categories along this path by G'. By definition of expensive path, $p_g \geq \max_{v_i \in V_{g,k_g}}(v_i)$ for each $g \in G$', so the sum of prices of categories $g \in G'$ is at least the GFT of an optimal PS, which is positive. However, the price-sum increases by a single unit each round, and the algorithm terminates when the price-sum hits zero, so the price-sum can never be positive. □

We can finally prove our main theorem.

Theorem 3.11. *For every recipe $\mathbf{r} \in R$, The expected $GFT_{\mathbf{r}}$ of Algorithm 2 is at least $max(0, (k_{\mathbf{r}} - |R|)/(k_{\mathbf{r}} + |R|))$ of the optimal $GFT_{\mathbf{r}}$. As a corollary, The GFT of Algorithm 2 is at least $\max(0, (k_{\min} - |R|)/(k_{\min} + |R|))$ of the optimal GFT.*

Proof. By Theorem 3.2 due to the property of Individually Rational no agent loses money by participating in the trade, so we never get a negative GFT. So in the case of $k_{\mathbf{r}} < |R|$ the lower bound is zero. For each recipe $\mathbf{r} \in R$ and for each category g with $r_g > 0$, by Lemma 3.10, the number of remaining agents satisfies: $r_g \cdot (k_g - |R|) \leq |M_g| \leq r_g \cdot (k_g + |R|)$, So there are at least $k_g - |R|$ and at most $k_g + |R|$ sets of r_g traders. Therefore, in the random selection of the final traders (Algorithm 4), at least $k_{\mathbf{r}} - |R|$ deals are done, and the participants are from the at most $k_{\mathbf{r}} + |R|$ highest sets of traders in each category g. Hence, the approximation ratio of the GFT coming from recipe $\mathbf{r}$ alone is at least $(k_{\mathbf{r}} - |R|)/(k_{\mathbf{r}} + |R|)$ of the optimum[10]

Taking the minimum over all recipes which participate in the optimal trade (i.e., with $k_{\mathbf{r}} > 0$), yields the ratio claimed in the corollary. □

When there is a single recipe, $k_{\min} = k$ and $|R| = 1$, so Theorem 3.11 provides the same guarantee $(k-1)/(k+1)$ as the guarantee of [24] for recipes with positive integer quantities.

[10] In Appendix F we show that this approximation ratio cannot be substantially improved.

References

[1] Moshe Babaioff and Noam Nisan. Concurrent Auctions Across the Supply Chain. *Journal of Artificial Intelligence Research (JAIR)*, 21:595–629, 2004. doi: 10.1613/jair.1316.

[2] Moshe Babaioff and William E. Walsh. Incentive-compatible, budget-balanced, yet highly efficient auctions for supply chain formation. *Decision Support Systems*, 39(1):123–149, March 2005. ISSN 01679236. doi: 10.1016/j.dss.2004.08.008.

[3] Moshe Babaioff and WilliamE Walsh. Incentive Compatible Supply Chain Auctions. In *Multiagent based Supply Chain Management*, volume 28, pages 315–350. Springer Berlin Heidelberg, 2006.

[4] Moshe Babaioff, Yang Cai, Yannai A Gonczarowski, and Mingfei Zhao. The best of both worlds: Asymptotically efficient mechanisms with a guarantee on the expected gains-from-trade. In *Proceedings of the 2018 ACM Conference on Economics and Computation*, pages 373–373. ACM, 2018. arXiv preprint 1802.08023.

[5] Moshe Babaioff, Kira Goldner, and Yannai A Gonczarowski. Bulow-klemperer-style results for welfare maximization in two-sided markets. In *Proceedings of SODA'20*, pages 2452–2471, 2020. arXiv preprint arXiv:1903.06696.

[6] L. Blumrosen and Y. Mizrahi. Approximating gains-from-trade in bilateral trading. In *WINE*, pages 400–413, 2016.

[7] Liad Blumrosen and Shahar Dobzinski. Reallocation mechanisms. In *EC*, pages 617–640, 2014.

[8] Liad Blumrosen and Shahar Dobzinski. (almost) efficient mechanisms for bilateral trading. In *Working paper*, 2018.

[9] Johannes Brustle, Yang Cai, Fa Wu, and Mingfei Zhao. Approximating Gains from Trade in Two-sided Markets via Simple Mechanisms, June 2017. URL http://arxiv.org/abs/1706.04637.

[10] Brahim Chaib-Draa and Jörg Müller. *Multiagent based supply chain management*, volume 28. Springer Science & Business Media, 2006.

[11] Rachel R Chen, Robin O Roundy, Rachel Q Zhang, and Ganesh Janakiraman. Efficient auction mechanisms for supply chain procurement. *Management Science*, 51(3):467–482, 2005.

[12] Leon Y. Chu and Zuo-Jun M. Shen. Agent Competition Double-Auction Mechanism. *Management Science*, 52(8):1215–1222, August 2006. ISSN 0025-1909. doi: 10.1287/mnsc.1060.0528. URL http://dx.doi.org/10.1287/mnsc.1060.0528. Online appendix available at https://pubsonline.informs.org/doi/suppl/10.1287/mnsc.1060.0528/suppl_file/mnsc.1060.0528-sm-chu_shen_8_06_ec1.pdf.

[13] Riccardo Colini-Baldeschi, Bart de Keijzer, Stefano Leonardi, and Stefano Turchetta. Approximately efficient double auctions with strong budget balance. In *SODA*, pages 1424–1443, 2016.

[14] Riccardo Colini-Baldeschi, Paul W Goldberg, Bart de Keijzer, Stefano Leonardi, Tim Roughgarden, and Stefano Turchetta. Approximately efficient two-sided combinatorial auctions. In *Proceedings of the 2017 ACM Conference on Economics and Computation*, pages 591–608. ACM, 2017.

[15] Marek Cygan. Improved approximation for 3-dimensional matching via bounded pathwidth local search. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 509–518. IEEE, 2013.

[16] P. Dütting, T. Roughgarden, and I. Talgam-Cohen. Modularity and greed in double auctions. *Games and Economic Behavior*, 105(C):59–83, 2017.

[17] Uriel Feige and Jan Vondrák. The submodular welfare problem with demand queries. *Theory of Computing*, 6(1):247–290, 2010.

[18] Moran Feldman and Rica Gonen. Removal and threshold pricing: Truthful two-sided markets with multi-dimensional participants. In *SAGT*, pages 163–175, 2018.

[19] Moran Feldman, Gonen Frim, and Rica Gonen. Multi-sided advertising markets: Dynamic mechanisms and incremental user compensations. In *GameSec*, pages 227–247, 2018.

[20] Diodato Ferraioli, Paolo Penna, and Carmine Ventre. Two-way greedy: Algorithms for imperfect rationality. In *International Conference on Web and Internet Economics*, pages 3–21. Springer, 2021.

[21] Matthias Gerstgrasser, Paul W Goldberg, Bart de Keijzer, Philip Lazos, and Alexander Skopalik. Multi-unit bilateral trade. In *Proceedings of the AAAI'19*, volume 33, pages 1973–1980, 2019. arXiv preprint 1811.05130.

[22] Dvir Gilor, Rica Gonen, and Erel Segal-Halevi. Ascending-price mechanism for general multi-sided markets. In *Multi-Agent Systems: 18th European Conference, EUMAS 2021, Virtual Event, June 28–29, 2021, Revised Selected Papers 18*, pages 1–18. Springer, 2021.

[23] Dvir Gilor, Rica Gonen, and Erel Segal-Halevi. Ascending-price mechanism for general multi-sided markets. In *Multi-Agent Systems*, pages 1–18. Springer Berlin Heidelberg, 2021.

[24] Dvir Gilor, Rica Gonen, and Erel Segal-Halevi. Strongly budget balanced auctions for multi-sided markets. *Artificial Intelligence*, 300:103548, 2021.

[25] Mira Gonen, Rica Gonen, and Pavlov Elan. Generalized trade reduction mechanisms. In *Proceedings of EC'07*, pages 20–29, 2007.

[26] Rica Gonen and Ozi Egri. Combima: Truthful, budget maintaining, dynamic combinatorial market. *Auton. Agents Multi Agent Syst.*, 34(1):14, 2020.

[27] Rica Gonen and Erel Segal-Halevi. Strongly budget balanced auctions for multi-sided markets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1998–2005, 2020.

[28] Viggo Kann. Maximum bounded 3-dimensional matching is max snp-complete. *Information Processing Letters*, 37(1):27–35, 1991.

[29] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[30] Shengwu Li. Obviously strategy-proof mechanisms. *American Economic Review*, 107(11):3257–87, 2017.

[31] R. P. McAfee. The gains from trade under fixed price mechanisms. *Applied Economics Research Bulletin*, 1, 2008.

[32] R. Preston McAfee. A dominant strategy double auction. *Journal of Economic Theory*, 56(2):434–450, April 1992. ISSN 00220531.

[33] Roger B. Myerson and Mark A. Satterthwaite. Efficient mechanisms for bilateral trading. *Journal of Economic Theory*, 29(2):265–281, April 1983. ISSN 00220531.

[34] Noam Nisan. Introduction to Mechanism Design (For Computer Scientists). In Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay Vazirani, editors, *Algorithmic Game Theory*, pages 209–241. Cambridge University Press, 2007. ISBN 978-0521872829.

[35] Erel Segal-Halevi, Avinatan Hassidim, and Yonatan Aumann. SBBA: A Strongly-Budget-Balanced Double-Auction Mechanism. In Martin Gairing and Rahul Savani, editors, *Algorithmic Game Theory*, volume 9928 of *Lecture Notes in Computer Science*, pages 260–272. Springer Berlin Heidelberg, 2016. doi: 10.1007/978-3-662-53354-3_21.

[36] Erel Segal-Halevi, Avinatan Hassidim, and Yonatan Aumann. MUDA: A Truthful Multi-Unit Double-Auction Mechanism. In *Proceedings of AAAI'18*. AAAI Press, February 2018a. arXiv preprint 1712.06848.

[37] Erel Segal-Halevi, Avinatan Hassidim, and Yonatan Aumann. Double Auctions in Markets for Multiple Kinds of Goods. In *Proceedings of IJCAI'18*. AAAI Press, July 2018b. Previous name: "MIDA: A Multi Item-type Double-Auction Mechanism". arXiv preprint: 1604.06210.

[38] William Vickrey. Counterspeculation, Auctions, and Competitive Sealed Tenders. *Journal of Finance*, 16 (No. 1 (Mar., 1961)):8–37, 1961.

APPENDIX

A More Related Work

A.1 Supply chain management

[10] provide a comprehensive survey of multiagent methods related to supply-chain management. The most general supply-chain auction we are aware of is the trade-reduction mechanism of [2, 3]. They allow procurement sets of multiple recipes. Their model differs from ours in several respects:

(a) They distinguish between "producer markets" and "consumer markets", with "goods" moving between markets, and impose constraints on the demand and supply of agents in each market. In contrast, our model is abstract and considers only the general notion of a "category", with no specific distinction between producers and consumers, and does not require the notion of a "good".

(b) Their "Unique Manufacturing Technology" requirement forbids some markets that are covered by our model, such as a market with one consumer-category and two producer-categories with the two recipes $(1, 1, 0)$ and $(1, 0, 1)$; see [3] Section 6.

(c) Their auction is WBB and truthful, while ours is SBB and obviously-truthful.

A.2 Two-sided markets

Two-sided markets have been extensively studied since the seminal work of [32]. Recently, [35] present a SBB variant of McAfee's mechanism, with similar GFT guarantees. Their mechanism may remove up to one *buyer* from the optimal trade, and it is the buyer with the lowest value among the buyers in the optimal trade. [9] present two simple mechanisms that are IR, truthful, and WBB, and obtain, in expectation, at least half of the expected GFT of the second-best efficiency benchmark.

[4] present approximation results for the double-auction setting and for double-auction with some added constraints on the pairs of agents who can trade with each other. [16] study similar double-sided setting to that of [4] however, their constraints are applied on each side of the market separately. [5] show a strong analogue to the result of Bulow-Klemperer for welfare in two-sided markets rather than revenue in auctions.

[11] consider a supply-chain auction with a sole buyer and single item-kind, but there are different producers in different supply-locations. The buyer needs a different quantity of the item in different demand-locations. The buyer conducts a reverse auction and has to pay, in addition to the cost of production, also the cost of transportation from the supply-locations to the demand-locations. They do not guarantee SBB. [25] generalize the above settings to a unified trade reduction procedure.

[31] designs a fixed-price SBB double auction under some assumptions of buyers' and sellers' bid distributions. Our result does not assume knowledge of the distribution of participating categories. Additionally, we also allow for any number of categories in each recipe, as opposed to two, as well as for multiple recipes to simultaneously trade.

[13, 8, 14] also present SBB auctions. [13, 8] auctions target double-sided and [14] target combinatorial markets. [26] present WBB ascending auction for combinatorial market. However, their goal is to maximize social welfare as opposed to our goal which is maximizing gain from trade[11]. Thus their mechanisms are not asymptotically-optimal for gain from trade. They also require a prior knowledge on the agents' valuations. Similarly to [14], [7] present a two-sided combinatorial market solution. [7]'s solution is WBB unlike our SBB solution and maximizes social welfare as opposed to our goal which is maximizing gain from trade.

[11] When optimizing GFT we optimize the difference between the total value of the sold items for the buyers and the total value of these items for the sellers. When optimizing social welfare in a market we optimize the sum of the buying agents' valuations plus the sum of the unsold items' value held by selling agents at the end of trade. Despite their conceptual similarity, the two objectives are rather different in approximation. In many cases the social welfare approximation is close to the optimal social welfare solution; however, the gain from trade approximation may not be within any constant factor of the optimal gain from trade.

Table 3: An example of a market with a recipe-tree as in Figure 1(b). Boldface values denote values of agents participating in the optimal trade.

Category	Agents' values
N_1: buyers	**17, 14, 13, 9,** 6, 2
N_2: sellers	**-4, -5,** -8, -10
N_3: A-producers	**-1, -3,** -5
N_4: B-producers	**-1, -4,** -6

[6] present a mechanism that obtains in expectation at least $1/e$ of the first-best GFT. However, they assume the buyer's valuation is drawn from a distribution satisfying the monotone hazard rate condition. In contrast our result does not assume knowledge of the distribution of participating categories, let alone a specific distribution.

The present work handles multiple categories of agents, but each agent is single-parametric. An orthogonal line of work ([21, 36, 37]) remains with two agent categories (buyers and sellers), but aims to handle multi-parametric agents. Another orthogonal line of work that also aims to handle multi-parametric agents ([19, 18]) remains with three agent categories (buyers, mediators and sellers). Moreover, their trade matches are conducted in two stages: first the mediator trade with the buyers on behalf of his sellers and then the mediator transfers payments to his matched sellers. Our auction unites all three categories of buyer, seller and mediator actions into a single simultaneous trade step.

A.3 Obviously-Truthful

A *randomized sequential mechanism* is a lottery over deterministic sequential mechanisms. [30] defined a randomized sequential mechanism as *universally obviously-truthful* if it is a lottery over obviously-truthful deterministic sequential mechanisms.

[20] characterize obviously-truthful mechanisms for binary allocation problems. They show that every obviously-truthful mechanism must use a greedy algorithm. Indeed, an ascending auction works similarly to a greedy algorithm, as we explain in Section G.3.

B Binary Recipes

In this section we assume that all recipes are binary, that is, $r_g \in \{0, 1\}$ for all $g \in G$, for all $\mathbf{r} \in R$.

We start by presenting an algorithm for computing an optimal trade, assuming all values are known (Section B.1). Then, we describe our ascending-prices mechanism for multi-sided markets with binary recipes (Section B.2). We continue by providing a detailed example of the mechanism execution (Section B.3). Finally, we prove the mechanism properties (Section B.4).

B.1 Computing an Optimal Trade

An algorithm for computing the optimal trade, assuming all values are known, is presented as Algorithm 5. We illustrate the algorithm on the market in Table 3.

The algorithm is based on contracting the recipe-forest down to a single node. Two types of contraction operations are used.

In a **vertical contraction**, a leaf that is a single child is combined with its parent in the following way. Suppose the sets of agent values in the child category are $v_1 \geq v_2 \geq \ldots \geq v_{m_v}$ and the agent values in the parent category are $u_1 \geq u_2 \geq \ldots \geq u_{m_u}$. Replace the parent category by a new category with $m := \min(m_v, m_u)$ values: $u_1 + v_1, u_2 + v_2, \ldots, u_m + v_m$. For example, a vertical contraction on the tree of Figure 1(b) results in the tree of Figure 1(c), where $N_3 \wedge N_4$ denotes the

Algorithm 5 Finding the optimal GFT — binary recipes.

Input: A set of categories G, a set of traders N_g for all $g \in G$,
and a recipe-forest R based on a forest T.
For each agent $i \in \cup_g N_g$, the value v_i is public knowledge.
Output: Optimal trade in the market.
1. If T has a single vertex g:
Return all agents in N_g with a non-negative value: $\{i \in N_g | v_i > 0\}$
2. Else, if T has two roots without children g_l and g_s:
Do a horizontal contraction of g_l into g_s. Go back to step 1.
3. Else, if there is a leaf g_l that is a single child of its parent g_p:
Do a vertical contraction of g_l into g_p. Go back to step 1.
4. Else, there is a leaf g_l with a sibling leaf g_s:
Do a horizontal contraction of g_l into g_s. Go back to step 1.

elementwise combination of N_3 and N_4. In the market in Table 3, $N_3 \wedge N_4$ contains the values $\{-1-1, -3-4, -5-6\} = \{-2, -7, -11\}$.

The rationale is that the unique root-leaf path that passes through the parent passes through its child too, and vice-versa. Therefore, any PS that contains an agent of the parent category must contain an agent of the child category, and vice-versa. In economic terms, these two categories are *complements*. Hence, elementwise combination of the two categories leads to a market with identical optimal GFT.

In a **horizontal contraction**, two sibling leaves are combined by taking the union of their categories. For example, a horizontal contraction on the tree of Figure 1(c) results in the tree of Figure 1(d). In the market in Table 3, $N_2 \cup (N_3 \wedge N_4)$ contains the values $\{-2, -4, -5, -7, -8, -10, -11\}$.

The rationale is that, for every path from the root to one leaf there exists a path from the root to the other leaf, and vice-versa. Therefore, in any PS that contains an agent of one leaf-category, this agent can be replaced with an agent from the other leaf-category. In economic terms, these categories are *substitutes*. Therefore, uniting them leads to a market with the same optimal GFT.

Given a tree with two or more vertices, we consider a leaf with a maximum depth (that is, a leaf farthest from the root). If this leaf is a single child, we apply vertical contraction. If it has a sibling, the sibling must also be a leaf, so we apply horizontal contraction. By repeating this process, we can contract any tree to a single node. For example, a vertical contraction on the tree of Figure 1(d), in the market of Table 3, yields: $\{17-2, 14-4, 13-5, 9-7, 6-8, 2-10\}$. The optimal trade in such a market is just the set of all deals with positive values, which in this case contains four deals with values $\{15, 10, 8, 2\}$. This corresponds to an optimal trade with $k = 4$ deals:

- Buyer 17, A-producer -1, B-producer -1;
- Buyer 14, seller -4;
- Buyer 13, seller -5;
- Buyer 9, A-producer -3, B-producer -4.

If the forest has two or more trees, all contracted trees can be further combined using a horizontal contraction to a single node. The process is summarized as **Algorithm 5**.

B.2 Ascending Auction Mechanism

The ascending-price auction is a randomized sequential mechanism. The general scheme is presented as **Algorithm 6**. For each category g, the auctioneer maintains a price p_g, and a subset $M_g \subseteq N_g$ of all agents that are "in the market" (that is, their value is higher than the current price

Algorithm 6 Ascending prices mechanism for binary recipes — main loop.

Input: A market N, a set of categories G and a recipe-forest R.
Output: Strongly-budget-balanced trade.
1. *Initialization:* Let $M_g := N_g$ for each $g \in G$.
   Using Equation (2), set an initial price p_g for each category $g \in G$.
2. Using Algorithm 7, select a set $G^* \subseteq G$ of categories.
3. For each $g^* \in G^*$, ask each agent in $i \in M_{g*}$ whether $v_i > p_{g*}$.
   (a) If an agent $i \in M_{g*}$ answers "no", then
      Remove i from M_{g*} and go back to step 1.
   (b) If all agents in M_{g*} for all $g^* \in G^*$ answer "yes", then
      for all $g^* \in G^*$, let $p_{g*} := p_{g*} + 1$.
   (c) If after the increase $\sum_{g \in G} p_g \cdot r_g = 0$ for some $\mathbf{r} \in R$, then
      go on to step 4.
   (d) else go back to step 3.
4. Determine final trade using Algorithm 8.

Algorithm 7 Finding a set of prices to increase — binary recipes.

Input: A set of categories G, a set of remaining traders M_g for
   all $g \in G$, and a recipe-forest R based on a forest T.
Output: A subset of G denoting categories for price-increase.
0. *Initialization*: For each category $g \in G$, let $m_g := |M_g|$.
1. If T contains two or more trees,
   Recursively run Algorithm 7 on each individual tree T';
      Denote the outcome by $I_{T'}$.
   Return $\bigcup_{T' \in T} I_{T'}$.
2. Let g_0 be the category at the root of the single tree.
   Let $c_{g_0} := \sum_{g' \in \text{CHILDREN}(g_0)} m_{g'}$.
3. If $m_{g_0} > c_{g_0}$ [or g_0 has no children at all],
   then return the singleton $\{g_0\}$.
4. Else ($c_{g_0} \geq m_{g_0}$), for each child g' of g_0:
   Recursively run Algorithm 7 on the sub-tree rooted at g';
      Denote the outcome by $I_{g'}$.
   Return $\bigcup_{g' \in child(g_0)} I_{g'}$.

of their category). At each iteration, the auctioneer chooses a subset of the prices, and increases each price p_g in this subset by 1. After each increase, the auctioneer asks each agent in turn, in a pre-specified order (e.g. by their index), whether their value is still higher than the price. An agent who answers "no" is permanently removed from the market. After each increase, the auctioneer computes the sum of prices of the categories in each recipe, defined as: Prices-sum$(\mathbf{r}) := \sum_{g \in G} p_g$. When this sum equals 0, the auction ends and the remaining agents trade in the final prices. Throughout the execution, we ensure that the sum of prices is the same for all recipes $\mathbf{r} \in R$, so that the price-sum crosses 0 for all recipes simultaneously, and all deals are simultaneously SBB.

To flesh out this scheme, we need to explain (a) how the prices are initialized, (b) how the set of prices to increase is selected, and (c) how the final trade is determined.

(a) The prices are initialized as follows:

$$p_g := \begin{cases} -V & g \text{ is not a leaf} \\ -V \cdot (\text{MAXDEPTH} - \text{DEPTH}(g) + 1) & g \text{ is a leaf} \end{cases} \tag{2}$$

Algorithm 8 Determining a feasible trade — binary recipes.

Input: A set of categories G,
a set of remaining traders M_g for all $g \in G$,
and a recipe-forest R based on a forest T.
Output: A set of PSs with remaining traders,
each of which corresponds to a recipe in R.
1. If T has a single vertex g:
Return M_g — the set of traders remaining in category g.
2. If T has two roots without children g_l and g_s:
Do a horizontal contraction of g_l into g_s. Go back to step 1.
3. Otherwise, pick an arbitrary deepest leaf category $g_l \in T$.
4. If g_l is a single child of its parent $g_p \in T$:
Perform a randomized vertical contraction of g_l and g_p.
Go back to step 1.
5. Otherwise, g_l has a sibling $g_s \in T$:
Perform a horizontal contraction of g_l and g_s.
Go back to step 1.

This guarantees that the initial price-sum in any path from the root to a leaf is the same: $-V \cdot (\text{MAXDEPTH} + 1)$. Additionally, the price in each category is lower than the lowest possible value of an agent in this category, which we denoted by $-V$.

(b) The set of prices to increase is selected by **Algorithm 7**. It is a recursive algorithm: if the forest contains only a single category (a root with no children), then this category is necessarily selected. Otherwise, in each tree, either its root category or some of its descendants are selected for increase. The selection is based on the number of agents of each category g who are currently in the market. We denote this number by $m_g := |M_G|$.

We denote the root category of a tree by g_0. The algorithm first compares m_{g_0} to the sum of the $m_{g'}$ over all g' that are children of g_0 (this sum is denoted by c_{g_0}). If m_{g_0} is larger, then the price selected for increase is the price of g_0; Otherwise (c_{g_0} is larger or equal), the prices to increase are the prices of some of its descendants' categories: for each child category, Algorithm 7 is used recursively to choose a subset of prices to increase, and all returned sets are combined. It is easy to prove by induction that the resulting subset contains exactly one price for each path from a root to a leaf. Therefore, if all prices in the subset are increased simultaneously by one unit, then the price-sum in all recipes remains equal.

Algorithm 7 always selects exactly one price to increase in every recipe in R. This guarantees that the equality of price-sums is preserved by the price-increases. The price never skips any agent's integer value, because the initial category price was a big negative integer number $(-V)$ and the increment is done always by 1 so the category price visits every integer from $-V$ to the current category price. at some point the price-sum is exactly 0, and the auction stops.

(c) Once the auction ends, the final trade is computed using **Algorithm 8**. At this stage, it is possible that in some recipes, the numbers of traders remaining in the market are not balanced. In order to construct an integer number of procurement-sets of each recipe, some agents must be removed from the trade. The traders to remove must be selected at random and not by their value, since selecting traders by value would make the mechanism non-truthful.

The computation of the final trade is similar to that of the optimal trade, except that the vertical contraction is replaced with a *randomized vertical contraction*. A leaf that is a single child is combined with its parent in the following way. Denote the leaf and parent category by l and p respectively, and let M_i be the set of traders remaining in category i. Let $m_{\min} := \min(|M_l|, |M_p|) =$ the number of procurement-sets that can be constructed from the agents in both categories. For each

$g \in \{l, p\}$ if $|M_g| > m_{\min}$ then choose $|M_g| - m_{\min}$ agents uniformly at random and remove them from M_g. Then perform a vertical contraction with the remaining agents. The horizontal contractions can be performed deterministically, as no traders should be removed.

B.3 Example Run

We illustrate Algorithm 6 using the example in Table 3, where the recipe set is $R = \{(1, 1, 0, 0), (1, 0, 1, 1)\}$ and the recipe-forest contains the single tree shown in Figure 1(b). The execution is shown in Table 4.

Step 1 Since MAXDEPTH $= 2$, the initial prices determined by (2) are $-V, -2V, -V, -V$. The price-sum in each recipe is $-3V$.

Step 2 The categories whose price should be increased are determined using Algorithm 7. Initially, the numbers of remaining traders in the four categories are $6, 4, 3, 3$. The algorithm compares $m_1 = 6$ to $m_2 + m_3 = 4+3$. Since $6 < 4+3$, category 1 (buyers) is not selected, the algorithm recursively checks the subtrees rooted at categories 2 and 3. In the former, there is only one category (sellers), so it is selected. In the latter, there is one child category 4. The algorithm compares $m_3 = 3$ with $m_4 = 3$. Since $3 \geq 3$, the algorithm selects the child category (B-producers). Therefore, the chosen set G^* is $\{2, 4\}$ = {seller, B-producer}.

Step 3 The auctioneer increases the prices of each category $g^* \in G^*$ by 1, until one agent of some category $g^* \in G^*$ indicates that his value is not higher than the price, and leaves the trade. The first agent who answers "no" is B-producer -6. While p_4 has increased to -6, p_2 has increased to $-V - 6$, so the price-sum in all recipes remains the same: $-2V - 6$. After B-producer -6 is removed, we return to step 2 to choose a new set of prices to increase. The algorithm keeps executing steps 2 and 3 as described in Table 4. Finally, while the algorithm increases p_1, and before buyer 9 exits the trade, the price-sum in all recipes becomes 0 and the loop ends.

Step 4 The final trade is determined by Algorithm 8. First, a randomized vertical contraction is first done between the A-producers and B-producers. Since there is one A-producer -1 and one B-producer -1, none of them has to be removed, and the combined category now has a single element. Next, a horizontal contraction is done between the pair of producers and the remaining two sellers. This results in a combined category of size 3. Finally, a randomized vertical contraction is done between this combined category and the buyers' category. Since there are 4 remaining buyers, but only 3 sets in the child category, one of the buyers is chosen at random and removed from trade. Finally, three deals are made: two deals follow the recipe $(1, 1, 0, 0)$ and involve a buyer and a seller, and one deal follows the recipe $(1, 0, 1, 1)$ and involves a buyer, an A-producer and a B-producer.

B.4 Proof of Algorithm Properties

A crucial feature of our mechanism is that the price-sum along each path from the same node to a leaf is constant.

Lemma B.1. *Throughout Algorithm 6, for any category $g \in G$, the price-sum along any path from g to a leaf is the same for all paths.*

Proof. After the initialization step, the price-sum in all paths from g to a leaf is equal: $-V \cdot (\text{MAXDEPTH} - \text{DEPTH}(g) + 1)$. The selection of prices to increase (Algorithm 7) guarantees that, for any $g \in G$, one of the following holds: either (a) no descendant of g is selected, or (b) exactly one node is selected in any path from g to a leaf. Algorithm 6 increases all selected prices simultaneously by the same amount of one unit; therefore the price-sum remains equal. □

Table 4: Execution of Algorithm 6 on market from Table 3

Category counts	G^*	**Price-increase stops when**	**New prices**	**Price-sum**
$6, 4, 3, 3$	$2, 4$	B-producer -6 exits	$-V, -V-6, -V, -6$	$-2V-6$
$6, 4, 3, 2$	$2, 3$	A-producer -5 exits	$-V, -11, -5, -6$	$-V-11$
$6, 4, 2, 2$	$2, 4$	seller -10 exits	$-V, -10, -5, -5$	$-V-10$
$6, 3, 2, 2$	1	buyer 2 exits	$2, -10, -5, -5$	-8
$5, 3, 2, 2$	$2, 4$	B-producer -4 exits	$2, -9, -5, -4$	-7
$5, 3, 2, 1$	$2, 3$	seller -8 exits	$2, -8, -4, -4$	-6
$5, 2, 2, 1$	1	buyer 6 exits	$6, -8, -4, -4$	-2
$4, 2, 2, 1$	$2, 3$	A-producer -3 exits	$6, -7, -3, -4$	-1
$4, 2, 1, 1$	1	price-sum crosses zero	$7, -7, -3, -4$	0

The strategic and economic properties of the auction are summarized in the following theorems.

Theorem B.2. *Algorithm 6 is universally strongly-budget-balanced, individually-rational and obviously truthful.*

Proof. Given a fixed priority-ordering on the agents, consider the deterministic variant of the algorithm in which, in step 4 of Algorithm 8, instead of the randomized vertical contraction, the removed agents in each category are selected deterministically by the fixed agent ordering. Algorithm 6 is a lottery on such deterministic mechanisms, where the agent ordering is selected uniformly at random. Therefore, to prove that the randomized mechanism satisfies a property universally, it is sufficient to prove that each such deterministic variant satisfies this property.

Strong budget balance holds since by Lemma B.1 (applied to the root category), the price-sum for all recipes remains the same throughout the execution, and the algorithm stops whenever this sum becomes 0.

Individual rationality holds since an agent $i \in N_g$ may remain in the market only if $v_i \geq p_g$. [12]

To prove obvious-truthfulness, we consider an agent $i \in N_g$ who is asked whether $v_i > p_g$, and check the two possible cases:

- Case 1: $v_i > p_g$. If the agent answers truthfully "yes", then his lowest possible utility is 0, since the mechanism is IR. If the agent answers untruthfully "no", then his highest possible utility is 0 since he is immediately removed from trade and cannot return.
- Case 2: $v_i \leq p_g$. If the agent answers truthfully "no", then his lowest possible utility is 0, since he is removed from trade immediately. If the agent answers untruthfully "yes", then his highest possible utility is 0, since the utility is $v_i - p_g$ and the price can only increase.

In both cases, the lowest possible utility of a truthful agent is at least the highest possible utility of a non-truthful agent. □

We now show that the ascending auction attains an asymptotically optimal GFT. The analysis assumes that the valuations are *generic* — the sum of valuations in every subset of agents is unique. In particular, the optimal trade is unique. This is a relatively mild assumption, since every instance can be modified to have generic valuations with negligible impact on the gains from trade, as explained by [2].

First, choose a sufficiently large constant $W \geq n+1$ and replace each value v_i by $2^W \cdot v_i$. This scaling obviously has no effect on the optimal or the actual trade. Then, arbitrarily assign a unique integer index $i \in \{1, \ldots, n\}$ to every agent, and set $v'_i := 2^W \cdot v_i + 2^i$.

[12] If there are no two agents with the same value, then agent i remains only if $v_i > p_g$; in case of ties, agent i may remain also when $v_i = p_g$, since only one agent is removed in each iteration.

Now the sum of valuations in every agent subset is unique, since the n least significant bits in its binary representation are unique. Moreover, for every subset $I \subseteq N$, $\sum_{i \in I} v'_i \approx 2^W \sum_{i \in I} v_i$ plus some "noise" smaller than $2^{n+1} \leq 2^W$.

Therefore, the optimal trade in the new instance corresponds to one of the optimal trades in the original instance, with the GFT multiplied by 2^W. If the constant W is sufficiently large, the "noise" has a negligible effect on the GFT.

Definition B.3. (a) The number of deals in the optimal trade is denoted by k.

(b) For each recipe $\mathbf{r} \in R$, the number of deals in the optimal trade corresponding to $\mathbf{r}$ is denoted by $k_{\mathbf{r}}$ (so $k = \sum_{\mathbf{r} \in R} k_{\mathbf{r}}$).

(c) The smallest positive number of deals of a single recipe in the optimal trade is denoted by $k_{\min} := \min_{\mathbf{r} \in R, k_{\mathbf{r}} > 0} k_{\mathbf{r}}$.

(d) For each recipe $\mathbf{r} \in R$, The GFT of all deals corresponding to $\mathbf{r}$ is denoted by GFT_r (so $GFT = \sum_{r \in R} GFT_r$).

Theorem B.4. *For every $r \in R$ the expected GFT_r of the ascending-price auction of Section B.2 is at least $1 - 1/k_r$ of the optimal GFT_r.*

As a corollary, The GFT of the ascending-price auction of Section B.2 is at least $1 - 1/k_{\min}$ of the optimal GFT.

Before proving the theorem, we remark on the dependence on $k_{\min}$. This dependence may appear weak, but it is the best possible. Consider a recipe-tree with 5 categories and 2 recipes: (dummy, buyer1, seller1) and (dummy, buyer2, seller2). The dummy category contains infinitely-many agents with value 0; the (buyer1, seller1) categories contain $k_{\max}$ pairs with a GFT of 1; the (buyer2, seller2) categories contain $k_{\min}$ pairs with a GFT of ${k_{\max}}^2$. Here, $OPT = k_{\max} + k_{\min} \cdot {k_{\max}}^2$. It is clearly equivalent to two independent two-sided markets: the (buyer1, seller1) market with $OPT = k_{\max}$ and the (buyer2, seller2) market with $OPT \gg k_{\max}$. The approximation ratio of any mechanism is dominated by the ratio on the (buyer2, seller2) market, which by Myerson-Satterthwaite theorem is at most $1 - 1/k_{\min}$. When there is only one optimal deal, $k_{\min} = 1$, the only way to satisfy the truthfulness requirement of the mechanism is to remove that only deal, so the approximation ratio is zero. Theorem 2 of [3] provides a similar guarantee for their WBB auction, and they too present an example showing that the ratio must depend on the recipe with the least number of PSs.

When there is a single recipe, $k_{\min} = k$, so Theorem B.4 provides the same guarantee as [32].

The proof of Theorem B.4 uses several definitions. For every category $g \in G$:

(*) $k_g :=$ the number of deals in the optimal trade containing an agent from N_g (equivalently: the number of deals whose recipe-path passes through g). If g is the root category then $k_g = k$. If g is any non-leaf category then

$$k_g = \sum_{g' \text{ is a child of } g} k_{g'}. \tag{3}$$

In the market in Table 4, k_g for categories 1,2,3,4 equals $4, 2, 2, 2$ respectively.

(*) $v_{g,k_g} :=$ the value of the k_g-th highest trader in N_g — the lowest value of a trader who participates in the optimal trade. In the market in Table 4, v_{g,k_g} for categories 1,2,3,4 equals $9, -5, -3, -4$ respectively. In any path from the root to a leaf, the sum of v_{g,k_g} is positive — otherwise we could remove the PS composed of the agents corresponding to this path, and get a trade with a higher GFT.

(*) $v_{g,k_g+1} :=$ the highest value of a trader who does not participate in the optimal trade (or $-V$ if no such trader exists). In the market in Table 4, v_{g,k_g+1} for categories 1,2,3,4 equals $6, -8, -5, -6$ respectively. In any path from the root to a leaf, the sum of v_{g,k_g+1} is at most 0 — otherwise we could add the corresponding PS and get a trade with a higher GFT.

Recall that, during the auction, $m_g := |M_g|$ = the number of agents of category g currently in the market (whose value is larger than p_g), and

$$c_g := \sum_{g' \text{ is a child of } g} m_{g'}. \tag{4}$$

When the algorithm starts, $m_g \geq k_g$ for all $g \in G$, since all participants of the optimal trade are in the market. Similarly, $c_g \geq k_g$. In contrast to equation (3), m_g and c_g need not be equal. By adding dummy agents with value $-V + 1$ to some categories, we can guarantee that, when the algorithm starts, $m_g = c_g$ for all non-leaf categories $g \in G$. For example, in the market in Table 4 it is sufficient to add a buyer with value $-V + 1$. This addition does not affect the optimal trade, since no PS in the optimal trade would contain agents with such low values. It does not affect the actual trade either, since the price-sum is negative as long as there are dummy agents in the market. Once $m_g = c_g$, we show that these values remain close to each other throughout the algorithm:

Lemma B.5. *For all non-leaf categories $g \in G$,*

$$c_g \leq m_g \leq c_g + 1.$$

Proof. The proof is by induction on the algorithm rounds. Before the first round, $m_g = c_g$ by the addition of dummy agents, so the claim holds.

In each round, if $m_g = c_g$ then Algorithm 7 never selects p_g for increase. Hence, Algorithm 6 never removes agents from M_g, so $c_g \leq m_g$ still holds. It may remove an agent from a child of g, but since at most one agent is removed in each round, $m_g \leq c_g + 1$ still holds after the removal.

If $m_g = c_g + 1$, then the algorithm never increases prices and never removes agents from children of g, so $m_g \leq c_g + 1$ still holds; it may remove at most one agent from M_g, so $c_g \leq m_g$ holds. □

Definition B.6. Given a price-vector $\mathbf{p}$, a subset $G' \subseteq G$ is called:
(a) *Cheap* — if $p_g \leq v_{g,k_g+1}$ for all $g \in G'$;
(b) *Expensive* — if $p_g \geq v_{g,k_g}$ for all $g \in G'$.

We apply Definition B.6 to paths in trees in the recipe-forest T. Intuitively, in a cheap path, the prices are sufficiently low to allow the participation of agents not from the optimal trade. In an expensive path, the prices are sufficiently high to allow the participation of agents only from the optimal trade.

Lemma B.7. *Let g_1, g_2 be two children of the same parent node $g_p \in T$. There cannot be simultaneously a cheap path from g_1 to a leaf and an expensive path from g_2 to a leaf.*

Proof. Let q_1 be the price-sum along the cheap path from g_1 to a leaf, and q_2 the price-sum along the expensive path from g_2 to a leaf. By definition of cheap and expensive paths, q_1 is the GFT of a part of non-optimal PS, and q_2 is the GFT of a part of an optimal PS; therefore $q_1 < q_2$. But both paths are children of the same node g, contradicting Lemma B.1. □

Lemma B.8. *If $m_g \leq k_g - 1$ for some $g \in G$, then there is an expensive path from g to a leaf.*

Proof. The fact that $m_g \leq k_g - 1$ means that $p_g \geq v_{g,k_g}$, so the condition for an expensive path holds for g itself. To show that it holds for a path from g to a leaf, we apply induction on $\text{HEIGHT}(g)$. If $\text{HEIGHT}(g) = 0$ (i.e., g itself is a leaf), then the claim is obvious. Otherwise, by Lemma B.5,

$$\sum_{g' \text{ is a child of } g} m_{g'} = c_g \leq m_g \leq k_g - 1 = \left(\sum_{g' \text{ is a child of } g} k_{g'} \right) - 1$$

Therefore, there is at least one child g' of g for which $m_{g'} \leq k_{g'} - 1$. Since $\text{HEIGHT}(g') < \text{HEIGHT}(g)$, by the induction assumption there is an expensive path from g' to a leaf. Prepending g to this path yields an expensive path from g to a leaf. □

Lemma B.9. *If $m_g \geq k_g + 1$ for some $g \in G$, then there is a cheap path from g to a leaf.*

Proof. The fact that $m_g \geq k_g + 1$ means that $p_g \leq v_{g,k_g+1}$, so the condition for a cheap path holds for g itself. To show that it holds for a path from g to a leaf, we apply induction on $\text{HEIGHT}(g)$. If $\text{HEIGHT}(g) = 0$ then the claim is obvious. Otherwise, there are two cases.

Case #1: g has a child g' for which $m_{g'} \geq k_{g'} + 1$. Then by the induction assumption there is a cheap path from g' to a leaf; prepending g to this path yields a cheap path from g to a leaf.

Case #2: $m_{g'} \leq k_{g'}$ for all children g' of g. Then,

$$c_g = \sum_{g' \text{ is a child of } g} m_{g'} \leq \sum_{g' \text{ is a child of } g} k_{g'} = k_g \leq m_g - 1.$$

Lemma B.5 implies that $m_g - 1 \leq c_g$, so all these inequalities are in fact equalities. In particular, $\sum_{g'} m_{g'} = \sum_{g'} k_{g'}$, where the sums are on all children g' of g. Together with $m_{g'} \leq k_{g'}$, this implies $m_{g'} = k_{g'}$ for all children g' of g. Now, let us look back at the history of price-increases made by the algorithm, and identify the most recent price-increase in a descendant of g (a category in the subtree below g). Before this price-increase, $c_g = m_g$ had necessarily held, since otherwise Algorithm 7 would have chosen g rather than a descendant of g. After the price-increase, we have $c_g = m_g - 1$. This means that the price-increase must have been in a child g' of g, and it caused $m_{g'}$ to decrease by one. So before this increase, this child had $m_{g'} = k_{g'} + 1$. Since $\text{HEIGHT}(g') < \text{HEIGHT}(g)$, by the induction assumption there was a cheap path from g' to a leaf. The price-increase of g' stopped at the moment when agent $k_{g'} + 1$ was removed from $M_{g'}$, i.e., it stopped at $p_{g'} = v_{g',k_{g'}+1}$; therefore, the same path from g' to a leaf is still cheap. Prepending g yields a cheap path from g to a leaf. □

Lemma B.10. *If $m_g \geq k_g + 1$ for some $g \in G$, then there is a cheap path from the root to a leaf (through g).*

Proof. By Lemma B.9 there is a cheap path from g to a leaf. Therefore, it is sufficient to prove that there is a cheap path from the root to g. The proof is by induction on $\text{DEPTH}(g)$. If $\text{DEPTH}(g) = 0$ (i.e., g itself is the root), then the claim is obvious. Otherwise, let g_p be the parent of g.

By Lemma B.7, since there is a cheap path from g to a leaf, there cannot be an expensive path from any other child of g_p to a leaf. So by Lemma B.8, $m_{g'} \geq k_{g'}$ for any child g' of g_p. Summing over all children of g_p (and adding 1 for the child g) gives:

$$c_{g_p} = \sum_{g' \text{ is a child of } g_p} m_{g'} \geq 1 + \sum_{g' \text{ is a child of } g_p} k_{g'} = k_{g_p} + 1.$$

Since $m_{g_p} \geq c_{g_p}$ by Lemma B.5, this implies $m_{g_p} \geq k_{g_p} + 1$. Since $\text{DEPTH}(g_p) < \text{DEPTH}(g)$, by the induction assumption there is a cheap path from the root to g_p; appending g to this path yields a cheap path from the root to g. □

Lemma B.11. *If $m_g \leq k_g - 1$ for some $g \in G$, and Algorithm 7 decides to increase the price of g or a descendant of it, then there is an expensive path from the root to a leaf (through g).*

Proof. By Lemma B.8, $m_g \leq k_g - 1$ implies that there is an expensive path from g to a leaf. Therefore, it is sufficient to prove that there is an expensive path from the root to g. The proof is by induction on $\text{DEPTH}(g)$. If $\text{DEPTH}(g) = 0$ (i.e., g itself is the root), then the claim is obvious. Otherwise, let g_p be the parent of g.

By Lemma B.7, since there is an expensive path from g to a leaf, there cannot be a cheap path from any other child of g_p to a leaf. So by Lemma B.9, $m_{g'} \leq k_{g'}$ for any child g' of g_p. Summing over all children of g_p (and subtracting 1 for child g) gives:

$$c_{g_p} = \sum_{g' \text{ is a child of } g_p} m_{g'} \leq -1 + \sum_{g' \text{ is a child of } g_p} k_{g'} = k_{g_p} - 1.$$

The fact that Algorithm 7 decides to increase the price of g or a descendant of it implies that $m_{g_p} \leq c_{g_p}$. Therefore, $m_{g_p} \leq k_{g_p} - 1$. Since $\text{DEPTH}(g_p) < \text{DEPTH}(g)$, by the induction assumption there is an expensive path from root to g_p; appending g to this path yields an expensive path from root to g. □

Lemma B.12. *When Algorithm 6 ends, $m_g \in \{k_g, k_g - 1\}$ for all $g \in G$.*

Proof. The proof is by contradiction.

First, suppose that $m_g \geq k_g + 1$ for some $g \in G$. By Lemma B.10, there is a cheap path from the root to a leaf; denote the set of categories along this path by G'. The sum of prices of categories $g \in G'$ is the GFT of a non-optimal PS, which is negative. As long as the price-sum is negative, the algorithm does not terminate.

Second, suppose that $m_g \leq k_g - 2$ for some $g \in G$. Since at most a single agent is removed in each iteration, this means that the algorithm decided to increase the price of g while m_g was equal to $k_g - 1$. By Lemma B.11, at that point there existed an expensive path from the root to a leaf; denote the set of categories along this path by G'. The sum of prices of categories $g \in G'$ is the GFT of an optimal PS, which is positive. However, the price-sum increases by a single unit each round, and the algorithm terminates when the price-sum hits zero, so the price-sum can never be positive. □

Proof of Theorem B.4. By Lemma B.12, each recipe $\mathbf{r} \in R$ with $k_{\mathbf{r}} = 0$ does not participate in the trade at all, since the leaf category g_l of $\mathbf{r}$ has $k_{g_l} = 0$ and therefore $m_{g_l} = 0$. For each recipe $\mathbf{r} \in R$ with $k_{\mathbf{r}} > 0$, for each category g in $\mathbf{r}$, all k_g optimal traders of g, except maybe the lowest-valued one, participate in the final trade. Therefore, in the random selection of the final traders (Algorithm 8), for each path with a corresponding recipe $r \in R$, at least $k_r - 1$ random deals out of the k_r optimal deals are executed. Hence, the expected GFT coming from recipe $\mathbf{r}$ alone is at least $(1 - 1/k_{\mathbf{r}})$ times the optimum.

Taking the minimum over all recipes yields the ratio claimed in the corollary. □

C Proof of Algorithm Properties of Integer Recipes

Analogously to the binary case, we first ensure that the weighted price-sum along each path from the same node to a leaf is constant. We also ensure that it is an integer.

C.1 Lemma 3.1

Lemma 1. *Throughout Algorithm 2, for any category $g \in G$, the weighted price-sum along any path from g to a leaf is an integer, and it is the same for all these paths.*

Proof. We first show that the lemma holds for the initial prices (1). Consider a path from g to some leaf g_l. The price for all non-leaf categories in this path is $-V$, so

$$\sum_{g' \in \text{PATH}(g \to g_l), g' \neq g_l} p_{g'} \cdot r_{g'} = -V \cdot \sum_{g' \in \text{PATH}(g \to g_l), g' \neq g_l} r_{g'}.$$

The price of g_l is determined such that

$$p_{g_l} \cdot r_{g_l} = -V \cdot (\text{MAXWD} - \text{WEIGHTD}(g_l) + r_{g_l}).$$

The total weighted price-sum is the sum of the above two expressions, which is

$$
\begin{aligned}
&-V \cdot \left(\textsc{MaxWD} - \textsc{WeightD}(g_l) + \sum_{g' \in \textsc{Path}(g \to g_l)} r_{g'} \right) \\
&= -V \cdot \left(\textsc{MaxWD} - \sum_{g' \in \textsc{Path}(root \to g_l)} r_{g'} + \sum_{g' \in \textsc{Path}(g \to g_l)} r_{g'} \right) \\
&= -V \cdot \left(\textsc{MaxWD} - \sum_{g' \in \textsc{Path}(root \to \textsc{Parent}(g))} r_{g'} \right) \\
&= -V \cdot (\textsc{MaxWD} - \textsc{WeightD}(g) + r_g),
\end{aligned}
$$

which is an integer and is independent of the selection of g_l.

The selection of prices to increase (Algorithm 3) guarantees that, for any $g \in G$, one of the following holds: either (a) no descendant of g is selected, or (b) exactly one node is selected in any path from g to a leaf. Algorithm 2 simultaneously increases the price of each selected category g^* by $1/r_{g*}$. Therefore, all the terms $p_{g*} \cdot r_{g*}$ increase simultaneously by 1. Therefore, the weighted price-sum in all paths from g to a leaf either does not change, or increases by 1. So the sum remains an integer, and remains equal. □

The strategic and economic properties of the auction are summarized in the following theorem.

C.2 Theorem 3.2

Theorem. *Algorithm 2 is universally strongly-budget-balanced, individually-rational and obviously truthful.*

The proof is identical to Theorem B.2 and we omit it.

To Analyze the gain-from-trade, we again assume that the valuations are *generic* — the sum of valuations in every subset of agents is unique. In particular, the optimal trade is unique. This is a relatively mild assumption, since every instance can be modified to have generic valuations, as explained in Section B.

To analyze the gain-from-trade, we define for every category $g \in G$:

(*) $k_g :=$ the number of deals in the optimal trade containing agents from N_g (equivalently: the number of deals whose recipe-path passes through g). If g is the root category then $k_g = k$. If g is any non-leaf category then

$$k_g = \sum_{g' \in \textsc{Children}(g)} k_{g'}. \tag{5}$$

In the market of Table 1, k_g for categories 1,2,3,4 equals $4, 2, 2, 2$ respectively.

(*) $V_{g,k_g} :=$ the set of values of the k_g-th highest set of traders in N_g after performing the internal contraction, i.e. the lowest set of values of traders that participate in the optimal trade. In the market in Table 1, V_{g,k_g} for categories 1,2,3,4 equals $(13), (-3,-4), (-3), (-3,-4)$ respectively. Note that, in any path from the root to a leaf, the sum of all $v_i \in V_{g,k_g}$ is at least 0 — otherwise we could remove the PS composed of the agents corresponding to this path, and get a trade with a higher GFT.

(*) $V_{g,k_g+1} :=$ the set of values of the k_{g+1}-th highest set of traders in N_g after performing the internal contraction, i.e. the highest set of values of traders that do not participate in the optimal trade. In the market in Table 1, V_{g,k_g+1} for categories 1,2,3,4 equals $(6), (-5,-8), (-5), (-6,-8)$ respectively. Note that, in any path from the root to a leaf, the sum of all $v_i \in V_{g,k_g+1}$ is negative — otherwise we could add the corresponding PS and get a trade with a higher GFT.

Recall that, during the auction, $m_g := |M_g|/r_g$ and

$$c_g := \sum_{g' \in \text{CHILDREN}(g)} \lfloor m_{g'} \rfloor . \tag{6}$$

When the algorithm starts, $m_g \geq k_g$ for all $g \in G$, since all participants of the optimal trade are in the market. Similarly, $c_g \geq k_g$. Similarly to the binary case, we add dummy agents with value $-V+1$ to some categories, such that, when the algorithm starts, $m_g = c_g$ for all non-leaf categories $g \in G$. For example, in the market of Table 1 it is sufficient to add a buyer and two B-producers with values $-V+1$. Once $m_g = c_g$, we show that these values remain close to each other throughout the algorithm:

C.3 Lemma 3.3

Lemma 2. *For all non-leaf categories $g \in G$,*

$$c_g \leq m_g \leq c_g + 1.$$

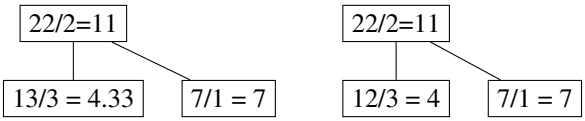
Proof. The proof is by induction on the algorithm rounds. Before the first round $m_g = c_g$ (thanks to the addition of dummy agents), so the claim holds.

In each round, if $m_g = c_g$ then Algorithm 3 never selects p_g for increase. Hence, Algorithm 2 never removes agents from M_g, so $c_g \leq m_g$ still holds. It may remove an agent from a child of g, but since at most one agent is removed in each round, $m_g \leq c_g + 1$ still holds after the removal.

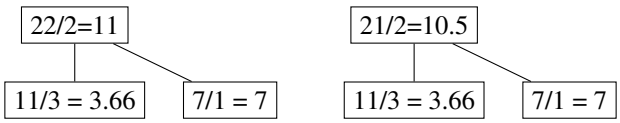
if $m_g > c_g$ then the algorithm never increases prices and never removes agents from children of g, so $m_g \leq c_g + 1$ still holds. It may remove at most one agent from M_g, which decreases the value of m_g by $1/r_g$. Since $c_g, r_g, -V$ are integers, and m_g is an integer multiple of $1/r_g$, the value of m_g does not go below c_g, so $c_g \leq m_g$ still holds. □

The illustrations below show some possible states of a tree subset during the execution. The top node is g and it has two child nodes. The numbers in the nodes are in the following format: $|M_g|/r_g = m_g$.

Initially, $m_g = 11$ and $c_g = \lfloor 4.33 \rfloor + \lfloor 7 \rfloor = 11$ too. Then, an agent from the left child is removed, and we still have $m_g = c_g = 11$:



Then, another agent from the left child is removed, and $m_g = c_g + 1$; Then an agent from g is removed, and $c_g < m_g < c_g + 1$:



The next agent will be removed from g again (since $m_g > c_g$), and at that point we will have $m_g = c_g = 10$.

Similarly to the binary case, in a cheap path, the prices are sufficiently low to allow the participation of agents not from the optimal trade (those in V_{g,k_g+1}), while in an expensive path, the prices are sufficiently high to allow the participation of agents only from the optimal trade (not including those in V_{g,k_g}).

C.4 Lemma 3.5

Lemma 3. *Let g_1, g_2 be two children of the same node $g_p \in T$. There cannot be simultaneously a cheap path from g_1 to a leaf and an expensive path from g_2 to a leaf.*

The proof is identical to that of Lemma B.7 so we omit it.

Recall that $\text{LVS}(g)$ is the set of leaf nodes that are descendants of g.

C.5 Lemma 3.6

Lemma 4. *If $m_g < k_g - |\text{LVS}(g)|$ for some category $g \in G$, then there is an expensive path from g to a leaf.*

Proof. The proof is by induction on $\text{HEIGHT}(g)$.

The base is $\text{HEIGHT}(g) = 0$, i.e., g is a leaf. The fact that $m_g < k_g - |\text{LVS}(g)| = k_g - 1$ implies that $|M_g| < k_g r_g - r_g$. This means that at least r_g agents from category g, who participate in the optimal trade, have already left the market due to price-increase. This means that $p_g \geq \max_{v_i \in V_{g,k_g}}(v_i)$, so the condition for an expensive path holds for g.

Suppose now that $\text{HEIGHT}(g) > 0$, i.e., g is not a leaf. By Lemma 3.3, $m_g \geq c_g$, so

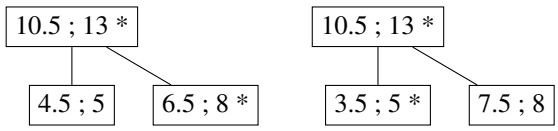
$$\sum_{g' \in \text{CHILDREN}(g)} \lfloor m_{g'} \rfloor = c_g \leq m_g < k_g - |\text{LVS}(g)| = \\ = \sum_{g' \in \text{CHILDREN}(g)} (k_{g'} - |\text{LVS}(g')|)$$

Since the sigmas on both sides are integers, it follows that:

$$\sum_{g' \in \text{CHILDREN}(g)} \lfloor m_{g'} \rfloor \leq \sum_{g' \in \text{CHILDREN}(g)} (k_{g'} - |\text{LVS}(g')|) - 1$$

Therefore, there is at least one child g' of g for which $\lfloor m_{g'} \rfloor \leq k_{g'} - |\text{LVS}(g')| - 1$. Since $k_{g'} \in \mathbb{Z}_+$, it follows that $m_{g'} < k_{g'} - |\text{LVS}(g')|$. Since $\text{HEIGHT}(g') < \text{HEIGHT}(g)$, by the induction assumption there is an expensive path from g' to a leaf. Prepending g to this path yields an expensive path from g to a leaf. □

Two possible subtrees are illustrated below. The top node is g and it has two child leaf nodes. The numbers in the nodes are in the format m_g ; k_g. The nodes in the expensive path are denoted by *.



$$\sum_{g' \in \text{CHILDREN}(g)} \lfloor m_{g'} \rfloor = 10 \leq 13 - 2 - 1 = \sum_{g' \in \text{CHILDREN}(g)} (k_{g'} - |\text{LVS}(g')|) - 1$$

For the top node we have $|\text{LVS}(g)| = 2$, so $m_g < k_g - |\text{LVS}(g)|$, and the lemma indicates that there should be an expensive path from g to a leaf. To identify this path, we should find a child of g in which the same inequality holds. For the leaf nodes, we have $|\text{LVS}(g')| = 1$. The inequality $m_{g'} < k_{g'} - |\text{LVS}(g')|$ holds for the rightmost leaf in the leftmost tree, and for the leftmost leaf in the rightmost subtree, so these are the leaves in the expensive path.

C.6 Lemma 3.7

Lemma 5. *If $m_g \geq k_g + 1$ for some $g \in G$, then there is a cheap path from g to a leaf.*

Proof. The fact that $m_g \geq k_g + 1$ implies that $|M_g| \geq k_g \cdot r_g + r_g$. This means that at least one set of r_g agents from category g, who do not participate in the optimal trade, is still in the market. This means that $p_g \leq \min_{v_i \in V_{g,k_g+1}}(v_i)$, so the condition for a cheap path holds for g itself. To show that it holds for a path from g to a leaf, we apply induction on $\text{HEIGHT}(g)$. If $\text{HEIGHT}(g) = 0$ then the claim is obvious. Otherwise, there are two cases.

Case #1: g has a child g' for which $m_{g'} \geq k_{g'} + 1$. Then by the induction assumption there is a cheap path from g' to a leaf; prepending g to this path yields a cheap path from g to a leaf.

Case #2: $m_{g'} < k_{g'} + 1$ for all children g' of g. Since $\lfloor m_{g'} \rfloor, k_{g'}$ are integers, it follows that $\lfloor m_{g'} \rfloor \leq k_{g'}$ for all children g' of g, therefore:

$$\sum_{g' \in \text{CHILDREN}(g)} k_{g'} \geq \sum_{g' \in \text{CHILDREN}(g)} \lfloor m_{g'} \rfloor = c_g$$

By Lemma 3.3 we have $c_g + 1 \geq m_g$, so

$$c_g \geq m_g - 1 \geq k_g = \sum_{g' \in \text{CHILDREN}(g)} k_{g'} \geq c_g$$

Therefore, in the expression above, all inequalities collapse to equalities. In particular, $m_g = k_g + 1$ and $c_g = m_g - 1$ (which implies that m_g is an integer).

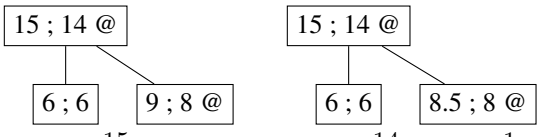
Now, let us look back at the history of price-increases made by the algorithm, and identify the most recent price-increase in a descendant of g (a category in the subtree below g). Before this price-increase, $c_g \geq m_g$ had necessarily held, since otherwise Algorithm 3 would not have chosen a descendant of g for increase. After the price-increase, $c_g = m_g - 1$ holds. This means that the price-increase must have been in some child g^* of g, and it caused $\lfloor m_{g^*} \rfloor$ and c_g to decrease by one. Before the increase, we had

$$\sum_{g' \in \text{CHILDREN}(g)} \lfloor m_{g'} \rfloor = c_g \geq m_g = k_g + 1 = \left(\sum_{g' \in \text{CHILDREN}(g)} k_{g'} \right) + 1.$$

Since for all other children g' of g except for g^*, $\lfloor m_{g'} \rfloor \leq k_{g'}$ is still true, therefore before the increase, $\lfloor m_{g^*} \rfloor \geq k_{g^*} + 1$ held.

Since $\text{HEIGHT}(g^*) < \text{HEIGHT}(g)$, by the induction assumption there was a cheap path from g^* to a leaf. The price-increase of g^* stopped at the moment when an agent from set $k_{g^*} + 1$ was removed from M_{g^*}, i.e., it stopped at $p_{g^*} \leq \min_{v_i \in V_{g^*,k_{g^*}+1}}(v_i)$; therefore, the same path from g^* to a leaf is still cheap. Prepending g to this path yields a cheap path from g to a leaf. □

This is illustrated below, where the left subtree is before and the right subtree is after the price-increase mentioned in the proof. The nodes in the cheap path are denoted by @.



$c_g = 15 = m_g$ $\quad$ $c_g = 14 = m_g - 1$

$\sum_{g' \in \text{CHILDREN}(g)} \lfloor m_{g'} \rfloor = 15 \geq 14 + 1 = \sum_{g' \in \text{CHILDREN}(g)} (k_{g'}) + 1$

For the top node $m_g \geq k_g + 1$, so the lemma indicates that there should be a cheap path from g to a leaf. The leftmost subtree illustrates Case #1: $m_{g'} \geq k_{g'} + 1$ holds for the rightmost leaf g', so it is the leaf in the cheap path. The rightmost subtree illustrates Case #2: $m_{g'} < k_{g'} + 1$ for all children of g. We have $c_g = 14 = m_g - 1$, and indeed the previous price-increase was in a child of g (the rightmost leaf).

C.7 Lemma 3.8

Lemma 6. *If $m_g \leq k_g - 1$ for some $g \in G$, and there is an expensive path from g to a leaf, and Algorithm 3 decides to increase the price of g or a descendant of g, then, even before the increase, there is an expensive path from the root to g.*

Proof. The proof is by induction on $\text{DEPTH}(g)$. If $\text{DEPTH}(g) = 0$ (i.e., g itself is a root), then the claim is obvious. Otherwise, let g_p be the parent of g. We will show that $m_{g_p} \leq k_{g_p} - 1$. Then, by the induction assumption there is an expensive path from the root to g_p; appending g to this path yields an expensive path from the root to g.

Aassume for contradiction that $m_{g_p} > k_{g_p} - 1$. The fact that Algorithm 3 decides to increase the price of g or a descendant of g, implies that $c_{g_p} \geq m_{g_p}$. Hence,

$$\sum_{g' \in \text{CHILDREN}(g_p)} \lfloor m_{g'} \rfloor = c_{g_p} \geq m_{g_p} > k_{g_p} - 1 = \sum_{g' \in \text{CHILDREN}(g_p)} k_{g'} - 1$$

Since $\lfloor m_g \rfloor \leq m_g$ and $m_g \leq k_g - 1$, we have $\lfloor m_g \rfloor \leq k_g - 1$. Removing from both sides the term corresponding to g (which is a child of g_p) yields:

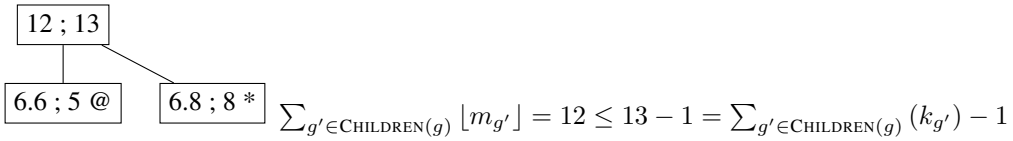
$$\left(\sum_{g \neq g' \in \text{CHILDREN}(g_p)} \lfloor m_{g'} \rfloor \right) > \left(\sum_{g \neq g' \in \text{CHILDREN}(g_p)} k_{g'} \right).$$

Since both sides are integers, it follows that:

$$\left(\sum_{g \neq g' \in \text{CHILDREN}(g_p)} \lfloor m_{g'} \rfloor \right) \geq \left(\sum_{g \neq g' \in \text{CHILDREN}(g_p)} k_{g'} \right) + 1.$$

Therefore, there is at least one child g' of g_p for which $\lfloor m_{g'} \rfloor \geq k_{g'} + 1$. Since $m_{g'} \geq \lfloor m_{g'} \rfloor$, we have $m_{g'} \geq k_{g'} + 1$. By Lemma 3.7, there is a cheap path from g' to a leaf. But by the assumption of the present Lemma, there is an expensive path from g — which is a sibling of g' — to a leaf. By Lemma 3.5, these two paths cannot exist simultaneously. □

An example of such an impossible subtree is illustrated below. The top node is g_p, the left child is g', and the right child is g.



For the right child g, we have $m_g \leq k_g - 1$, and we assume there is an expensive path from g to a leaf, and Algorithm 3 decides to increase the price of g or a descendant of g (since $m_{g_p} = 12 \leq 12 = c_{g_p}$), then, even before the increase, there is an expensive path from the root to g. And it is not possible to have a cheap path on the left child g'.

C.8 Lemma 3.9

Lemma 7. *If $m_g \geq k_g + |R| - |\textsc{Lvs}(g)| + 1$ for some $g \in G$, then there is a cheap path from the root to a leaf (through g).*

Proof. Since $|R| = \textsc{Lvs}(\text{root}) \geq |\textsc{Lvs}(g)|$ for all $g \in G$, the lemma assumption $m_g \geq k_g + |R| - |\textsc{Lvs}(g)| + 1$ implies $m_g \geq k_g + 1$. By Lemma 3.7 there is a cheap path from g to a leaf. Therefore, it is sufficient to prove that there is a cheap path from the root to g. The proof is by induction on $\textsc{Depth}(g)$. If $\textsc{Depth}(g) = 0$ (i.e., g itself is a root), then the claim is obvious.

Otherwise, let g_p be the parent of g. We will show that $m_{g_p} \geq k_{g_p} + |R| - |\textsc{Lvs}(g_p)| + 1$. Then, by the induction assumption there is a cheap path from the root to g_p; appending g to this path yields a cheap path from the root to g.

By Lemma 3.3, $m_{g_p} \geq c_{g_p}$. By the present lemma assumption, $\lfloor m_g \rfloor \geq k_g + |R| - |\textsc{Lvs}(g)| + 1$, since $k_g, |R|, |\textsc{Lvs}(g)|$ are integers.

As there is a cheap path from g to a leaf, by Lemma 3.5 there cannot exist an expensive path from any sibling of g to a leaf. By Lemma 3.6, all siblings g' of g must have $m_{g'} \geq k_{g'} - |\textsc{Lvs}(g')|$, and since the right-hand side is an integer, $\lfloor m_{g'} \rfloor \geq k_{g'} - |\textsc{Lvs}(g')|$ must hold too. Therefore:

$$
\begin{aligned}
& m_{g_p} \geq c_{g_p} \\
& = \lfloor m_g \rfloor + \sum_{g \neq g' \in \textsc{Children}(g_p)} \lfloor m_{g'} \rfloor \\
& \geq (k_g + |R| - |\textsc{Lvs}(g)| + 1) + \sum_{g \neq g' \in \textsc{Children}(g_p)} (k_{g'} - |\textsc{Lvs}(g')|) \\
& = (k_g + |R| - |\textsc{Lvs}(g)| + 1) + (k_{g_p} - k_g) - (|\textsc{Lvs}(g_p)| - |\textsc{Lvs}(g)|) \\
& = k_{g_p} + |R| - |\textsc{Lvs}(g_p)| + 1,
\end{aligned}
$$

which concludes the proof by induction. □

C.9 Lemma 3.10

Lemma 8. *When Algorithm 2 ends,*

$$k_g - |\textsc{Lvs}(g)| \leq m_g \leq k_g + |R|$$

for all $g \in G$.

Proof. The proof is by contradiction.

First, suppose that $m_g > k_g + |R|$ for some $g \in G$. Then $m_g > k_g + |R| - \textsc{Lvs}(g) + 1$. By Lemma 3.9, there is a cheap path from the root to a leaf; denote the set of categories along this path by G'. By definition of a cheap path, $p_g \leq \min_{v_i \in V_{g,k_g+1}}(v_i)$ for all $g \in G'$. So the sum of prices of categories $g \in G'$ is at most the GFT of a non-optimal PS, which is negative. As long as the price-sum is negative, the algorithm does not terminate.

Second, suppose that $m_g < k_g - |\textsc{Lvs}(g)|$ for some $g \in G$. Since at most a single agent is removed in each iteration, this means that the algorithm decided to increase the price of g while m_g was equal to $k_g - |\textsc{Lvs}(g)|$. By Lemma 3.6 and Lemma 3.8, at that point there existed an expensive path from the root to a leaf; denote the set of categories along this path by G'. By definition of expensive path, $p_g \geq \max_{v_i \in V_{g,k_g}}(v_i)$ for each $g \in G$', so the sum of prices of categories $g \in G'$ is at least the GFT of an optimal PS, which is positive. However, the price-sum increases by a single unit each round, and the algorithm terminates when the price-sum hits zero, so the price-sum can never be positive. □

Table 5: Notations

Variable	Description	Equation
N	Set of agents	
G	Set of agent categories	
N_g	Set of agents in category $g \in G$	$\sqcup_{g \in G} N_g$
PS	Procurement-set: a subset of agents that can perform a single deal	
r_g	number of agents of category g that should be in each PS	
$\mathbf{r}$	Vector of number of agents of each category that should be in each PS	$(r_g)_{g \in G}$
$v_i \in \mathbb{Z}$	Represents the material gain of an agent i from participating in a PS	
V	Publicly known bounds on the possible valuations	$\forall i \in N : -V < v_i < V$
T	A forest in which, in each tree, one vertex is denoted as its *root*	
R	Recipe-forest: a rooted forest T in which the set of nodes is G	
P	Path in some tree T	
$\textsc{Children}(g)$	Child nodes of the node g in its tree	
$\textsc{Lvs}(g)$	Leaf descendants of the node g in its tree (if g is a leaf then $\textsc{Lvs}(g) = \{g\}$)	
$\textsc{Path}(g_1 \to g_2)$	Nodes in the unique path from g_1 to g_2, inclusive	
$\textsc{Height}(g)$	Largest distance between the node g and a leaf of its tree. The height of a leaf is 0	
$\textsc{Depth}(g)$	Unique distance between the node g and the root of its tree. The depth of a root is 0	
MaxDepth	Maximum depth of forest T	$\max_{g \text{ is a leaf in } T} \textsc{Depth}(g)$
$GFT(S)$	*Gain-from-trade* of a procurement-set S	$\sum_{i \in S} v_i$
$k_{\mathbf{r}}$	Number of deals in the optimal trade corresponding to $\mathbf{r}$	
k	The number of deals in the optimal trade	$\sum_{\mathbf{r} \in R} k_{\mathbf{r}}$
$k_{\min}$	The smallest positive number of deals of a single recipe in the optimal trade	$\min_{\mathbf{r} \in R, k_{\mathbf{r}} > 0} k_{\mathbf{r}}$
$\text{GFT}(S_1, \ldots, S_k)$	Sum of the GFT of all procurement-sets participating in the trade	$\sum_{j=1}^{k} \text{GFT}(S_j)$
Prices-sum($\mathbf{r}$)	The sum of prices of the categories in each recipe	$\sum_{g \in \mathbf{r}} p_g$
M_g	Agents of category g who are currently in the market	$M_g \subseteq N_g$

Table 6: Notations for binary recipes

Variable	Description	Equation
m_g	Number of agents of category g who are currently in the market	$\lvert M_g \rvert$
c_g	Sum of m'_g of children g' of g	$\sum_{g' \in \textsc{Children}(g)} m_{g'}$
v_{g,k_g+1}	The highest value of a trader that does not participate in the optimal trade	
Cheap	The prices are sufficiently low to allow the participation of agents not from the optimal trade	if $p_g \le v_{g,k_g+1}, \forall g \in G'$
v_{g,k_g}	The lowest value of a trader that participates in the optimal trade	
Expensive	The prices are sufficiently high to allow the participation of agents only from the optimal trade	if $p_g \ge v_{g,k_g}, \forall g \in G'$

D Experiments

We evaluated the performance of our ascending auction using simulation experiments. [13]

For these experiments, we used the recipe-forests $\mathbf{R} = \{(1,1,0,0),(1,0,1,1)\}$ and $\mathbf{R} = \{(1,2,0,0),(1,0,1,2)\}$, each of which each contains a single tree with two paths ($N_1 \longrightarrow N_2$ and $N_1 \longrightarrow N_3 \longrightarrow N_4$).

For several values of $n \le 2000$, we constructed a market with $n \cdot r_g$ agents of each category g, such that the potential number of procurement-sets is n. We chose n to be a number divisible by $|\textsc{Children}(g_{root})|$ (= number of children of the root category), and at most 2000. The value of each trader was selected randomly as described in Section D.1 below. For each n, we made 10,000 runs and averaged the results. We split the values among the categories uniformly at random, so each category has n values.

D.1 Agents' Values

We conducted two sets of experiments. In the first experiment set, the value of each buyer (root category) was selected uniformly at random from $[1, 1000]$, and the value of each trader from the other three categories was selected uniformly at random from $[-1, -1000]$.

[13] The code used for the experiments and the experiment results are available at https://github.com/dvirg/auctions.

Table 7: Notations for integer recipes

Variable	Description	Equation
$\textsc{Weight}(g)$	Distance between g and the root of its tree (including the root), weighted by the $r_{g'}$	$\sum_{g' \in \textsc{Path}(g \to root)} r_{g'}$
$\textsc{MaxWD}$	Maximum weighted depth of forest T	$\max_{g \text{ is a leaf in } T} \textsc{Weight}(g)$
m_g	Number of agents of category g who are currently in the market divided by category size r_g	$\lvert M_g \rvert / r_g$
c_g	Sum of rounded down of m'_g of children g' of g	$\sum_{g' \in \textsc{Children}(g)} \lfloor m_{g'} \rfloor$
V_{g,k_g+1}	The highest set of values of traders that do not participate in the optimal trade	
Cheap	The prices are sufficiently low to allow the participation of agents not from the optimal trade	if $p_g \leq \min_{v_i \in V_{g,k_g+1}}(v_i), \forall g \in G'$
V_{g,k_g}	The lowest set of values of traders that participate in the optimal trade	
Expensive	The prices are sufficiently high to allow the participation of agents only from the optimal trade	if $p_g \geq \max_{v_i \in V_{g,k_g}}(v_i), \forall g \in G'$

Table 8: Results of experiment with stock-market prices and the recipe-forest $\mathbf{R} = \{(1,1,0,0),(1,0,1,1)\}$.

	Optimal					Ascending Price					
n	k	$k_{\min}$	$k_{\max}$	LB	$OGFT$	k'	${k_{\min}}'$	${k_{\max}}'$	$\%k'$	GFT	$\%GFT$
2	1.11	1.00	1.00	0.496	903.2	0.47	0.47	0.47	37.183	483.8	43.393
4	2.28	1.54	1.82	35.401	2003.9	1.50	1.34	1.37	64.908	1667.6	80.315
6	3.47	1.85	2.66	46.183	3092.3	2.60	1.90	2.22	74.594	2855.7	90.595
10	5.84	2.40	4.29	58.373	5163.3	4.89	2.52	3.87	83.634	5025.1	96.321
16	9.39	3.35	6.75	70.174	8415.4	8.42	3.32	6.34	89.593	8333.2	98.502
26	15.31	5.11	10.85	80.449	13683.2	14.34	4.89	10.45	93.603	13633.3	99.426
50	29.53	9.63	20.70	89.621	26223.9	28.55	9.24	20.31	96.655	26195.7	99.834
100	59.16	19.18	41.20	94.787	53135.7	58.18	18.79	40.81	98.340	53121.7	99.957
500	296.00	92.89	205.20	98.923	266371.5	295.00	94.80	204.78	99.661	266368.4	99.997
1000	592.02	182.49	410.16	99.452	532833.8	591.03	184.66	409.72	99.832	532832.4	99.999
2000	1184.19	364.14	820.05	99.725	1065649.7	1183.18	363.88	819.55	99.914	1065649.1	99.999

In the second experiment set, the values were selected based on real stocks prices on Yahoo's stock market site using 33 stocks. For each stock, we collected the prices from every day from the inception of the stock until September 2020. Every day the stock has 4 values: Open, Close, High and Low. All price values are multiplied by 1000, so they can be represented as integers, to avoid floating-point rounding errors. On each stock, we collected all the price values and used those price values as agents' values at random. For the non-root categories, the values were multiplied by -1. There were more than 5000 values for each category.

D.2 Number of Deals and Gain From Trade

In each run, we calculated k (the number of deals in the optimal trade), $k_{\min}$, $k_{\max}$ (recipe minimum and maximum number of deals in the optimal trade), LB (the theoretical lower bound ratio $\frac{k_{\min}-1}{k_{\min}}$ or $\frac{k_{\min}-|R|}{k_{\min}+|R|}$) and $OGFT$ (the optimal gain-from-trade). For the ascending-price mechanism, we calculated k' (the actual number of deals achieved by the mechanism), ${k_{\min}}'$, ${k_{\max}}'$ (the actual recipe minimum and maximum number of deals achieved by the mechanism) and the GFT (the actual gain-from-trade of deals achieved by the mechanism).

D.3 Results and Conclusions

The results from the stock-prices experiment are presented in Tables 8 and 10 and in Figures 2 and 4. The results from the uniform-random experiment are presented in Tables 9 and 11 and in Figures 3 and 5.

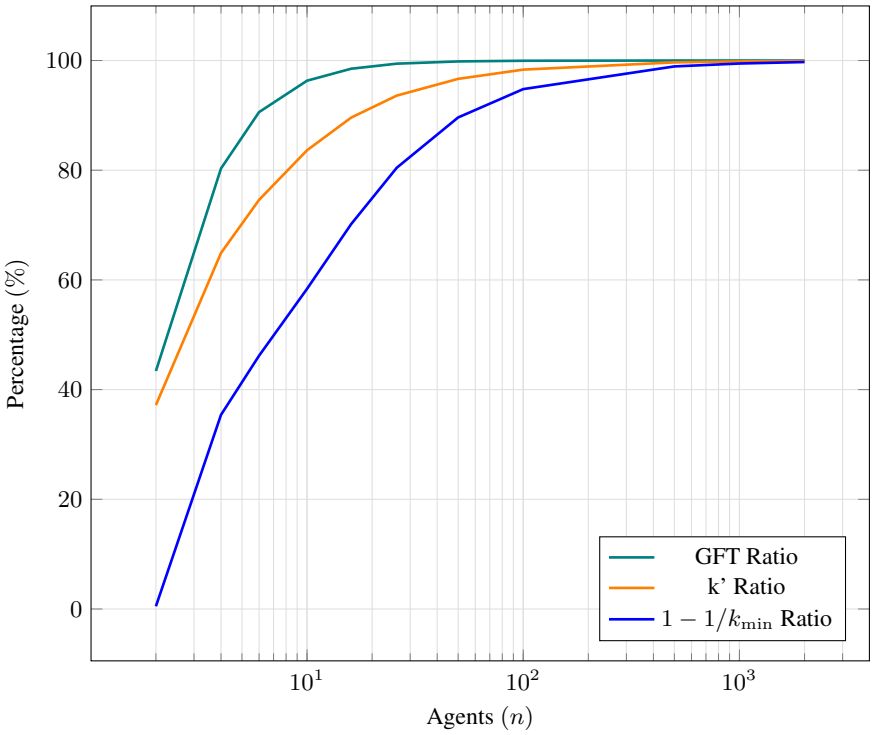


Figure 2: Graph of results from Table 8. GFT Ratio is the actual gain-from-trade of deals achieved by the mechanism divided by the optimal gain-from-trade. k' Ratio is the actual number of deals achieved by the mechanism divided by the number of deals in the optimal trade. $1 - 1/k_{\min}$ Ratio is the theoretical lower bound ratio (LB).

The highlights of both sets of experiments are similar. Below are some of the highlights:

- The actual number of trades (k') is very close to $k - 1$. Note that, theoretically, the mechanism might lose up to one optimal deal in the recipe-forest $\mathbf{R} = \{(1,1,0,0),(1,0,1,1)\}$ and up to two optimal deals in the recipe-forest $\mathbf{R} = \{(1,2,0,0),(1,0,1,2)\}$ (see the proof of Theorems B.4 and 3.11). But in practice, it loses about one optimal deal on average.

- The actual number of minimum and maximum trades (${k_{\min}}'$ and ${k_{\max}}'$) is very near optimal $k_{\min} - 0.5$ and $k_{\max} - 0.5$. Note that, theoretically in the recipe-forest $\mathbf{R} = \{(1,1,0,0),(1,0,1,1)\}$ and $\mathbf{R} = \{(1,2,0,0),(1,0,1,2)\}$, the mechanism might lose up to one and two optimal deals respectively for each recipe (see the proof of Theorems B.4 and 3.11). But in practice, it loses about half an optimal deal on average.[14]

- The actual GFT of the ascending auction is much higher than the theoretical lower bound (LB) of the optimum. For example, in the experiments with recipe-tree $\mathbf{R} = \{(1,1,0,0),(1,0,1,1)\}$ when $n = 10$ (and $k_{\min} \leq 3$), the theoretical lower bound is approximately 50%, but the ascending-price auction attains more than 95%. It surpasses 99.9% already for $n \geq 100$. In the experiments with recipe-tree $\mathbf{R} = \{(1,2,0,0),(1,0,1,2)\}$ when

[14] Since in our experiments $|R| = 2$ we may think that in one leaf there are $k_{\min}$ optimal deals and in the other leaf there are $k_{\max}$ optimal deals, which means by definition $k = k_{\min} + k_{\max}$. But for small values of n, sometimes there are optimal deals only in one path of the tree. In such cases $k_{\min} = k_{\max} = k$. Therefore, for small n the average of $k_{\min}$ plus the average of $k_{\max}$ may be larger than the average of k. For the same reason, ${k_{\min}}'$ may be greater than $k_{\min}$ for some values of n.

Table 9: Results of experiment with values chosen uniformly at random, and recipe-forest $\mathbf{R} = \{(1,1,0,0),(1,0,1,1)\}$.

	Optimal					Ascending Price					
n	k	$k_{\min}$	$k_{\max}$	LB	$OGFT$	k'	${k_{\min}}'$	${k_{\max}}'$	$\%k'$	GFT	$\%GFT$
2	1.13	1.00	1.00	0.279	457.2	0.47	0.47	0.47	41.743	258.2	56.490
4	2.32	1.44	1.77	30.963	1027.8	1.46	1.29	1.32	63.136	823.1	80.079
6	3.51	1.59	2.54	37.158	1616.0	2.57	1.72	2.10	73.105	1427.8	88.354
10	5.91	1.98	4.13	49.604	2803.3	4.92	1.98	3.71	83.209	2665.2	95.073
16	9.53	3.03	6.52	67.029	4615.1	8.53	2.58	6.11	89.500	4522.6	97.994
26	15.55	5.02	10.52	80.097	7604.2	14.55	4.43	10.11	93.576	7546.7	99.244
50	29.89	9.85	20.04	89.847	14782.6	28.91	9.26	19.64	96.695	14752.3	99.795
100	59.92	19.83	40.08	94.958	29754.2	58.92	19.26	39.65	98.330	29739.3	99.949
500	300.01	99.71	200.30	98.997	149526.3	299.01	99.21	199.80	99.666	149523.7	99.998
1000	600.13	199.45	400.68	99.498	299474.3	599.12	199.04	400.08	99.832	299473.2	99.999
2000	1200.40	399.33	801.06	99.749	598969.0	1199.40	399.13	800.26	99.917	598968.6	99.999

Table 10: Results of experiment with stock-market prices and the recipe-forest $\mathbf{R} = \{(1,2,0,0),(1,0,1,2)\}$.

	Optimal					Ascending Price					
n	k	$k_{\min}$	$k_{\max}$	LB	$OGFT$	k'	${k_{\min}}'$	${k_{\max}}'$	$\%k'$	GFT	$\%GFT$
2	0.41	0.40	0.40	0.000	511.0	0.05	0.05	0.05	5.258	113.1	5.601
4	1.15	0.85	0.87	0.000	1414.5	0.50	0.45	0.45	33.520	879.6	39.463
6	1.82	1.04	1.24	0.000	2284.3	1.10	0.81	0.86	53.199	1864.5	64.838
10	3.14	1.40	2.02	0.000	4012.6	2.38	1.23	1.64	70.782	3728.2	84.945
16	5.11	2.11	3.16	2.754	6501.7	4.33	1.88	2.79	81.226	6302.7	93.121
26	8.35	3.42	5.04	26.227	10630.7	7.56	3.12	4.65	88.208	10501.3	96.838
50	16.14	6.75	9.47	54.304	20675.6	15.34	6.38	9.09	93.824	20600.8	98.990
100	32.41	13.80	18.70	74.693	41537.5	31.60	13.42	18.31	96.905	41498.5	99.793
500	162.54	70.75	91.88	94.502	208332.9	161.72	70.47	91.50	99.338	208323.6	99.978
1000	325.07	141.99	183.10	97.222	417308.0	324.25	141.63	182.73	99.672	417303.1	99.993
2000	650.50	285.05	365.44	98.606	834387.5	649.70	284.65	365.06	99.837	834384.9	99.998

$n = 26$ (and $k_{\min} \leq 4$), the theoretical lower bound is lower than 30%, but the ascending-price auction attains more than 98%. It surpasses 99.9% already for $n \geq 500$.

- We performed an experiment to compare the performance of our mechanism on binary versus non-binary recipe-trees, by duplicating the values from the binary market to the non-binary market. The recipe-trees were: $\mathbf{R} = \{(1,1,0,0),(1,0,1,1)\}$ and $\mathbf{R} = \{(2,2,0,0),(2,0,2,2)\}$. We got the exact same results as described in Tables 8 and 9, even though the lower bound in the non-binary algorithm is lower than the binary algorithm lower bound.

- We performed an experiment to compare the optimal deals (k) and the actual deals (k') that our algorithm finds and how it is dependant on $|R|$ of non-binary "wide" recipe-trees (with many children). We used a market with a root that has 20 children, each with an agent count (r_g) of 20. The results are shown in Table 12 and Figure 6.

  Note that theoretically, the mechanism might lose up to $|R|$ optimal deals (the difference between k and k'). In our experiment, for low values of n, the mechanism loses approximately $|R|/2$ optimal deals on average. For higher values of n, the number of optimal deals lost on average, goes down to 0.5. The actual GFT of the algorithm is much higher than the theoretical lower bound (LB) of the optimum. It surpasses 99% already for $n \geq 500$.

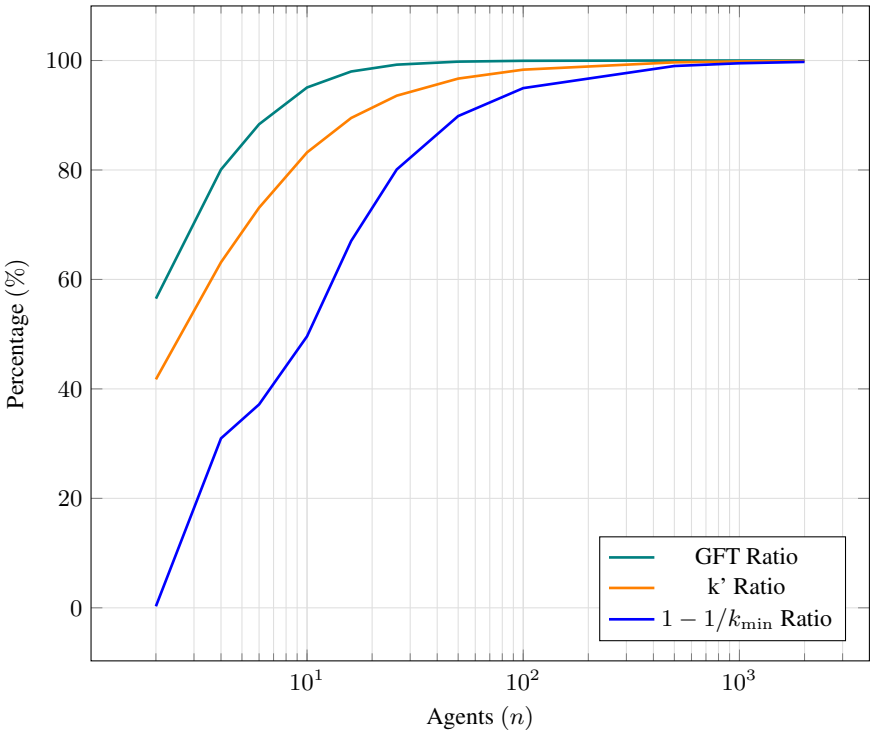


Figure 3: Graph of results from Table 11.

E Hardness of General Recipe Sets

To illustrate the difficulty of handling general recipe-sets, we prove that calculating the optimal trade, even without strategic considerations, is MAX-SNP-hard. This is means not only that it is NP-hard, but also that it does not have a PTAS unless P=NP. In other words, the best approximation algorithm that can be hoped for this problem is a constant-factor approximation. The theorem was already proven in [23] and we repeat it here for completeness.

Theorem E.1. *The following problem is MAX-SNP-hard. Given a set N of agents with known integer valuations, a set G of categories, a set R of recipes, and an integer C, decide whether there exists a feasible trade in which the GFT is at least C.*

Proof. The proof is by reduction from *3-dimensional matching*, which is the following decision problem: given a 3-uniform hypergraph $H = (V, E)$ (a hypergraph in which each edge in E contains 3 vertices of V) and a positive integer C, decide whether H has a matching that contains at least C edges. This problem is known to be NP-hard [29] and MAX-SNP-hard [28].

Given an instance $H = (V, E)$ of 3-D matching, construct an instance of the GFT problem as follows.

- There is a category for each vertex: $G = V$.
- Each category contains a single agent.
- The value of every agent is $1/3$.

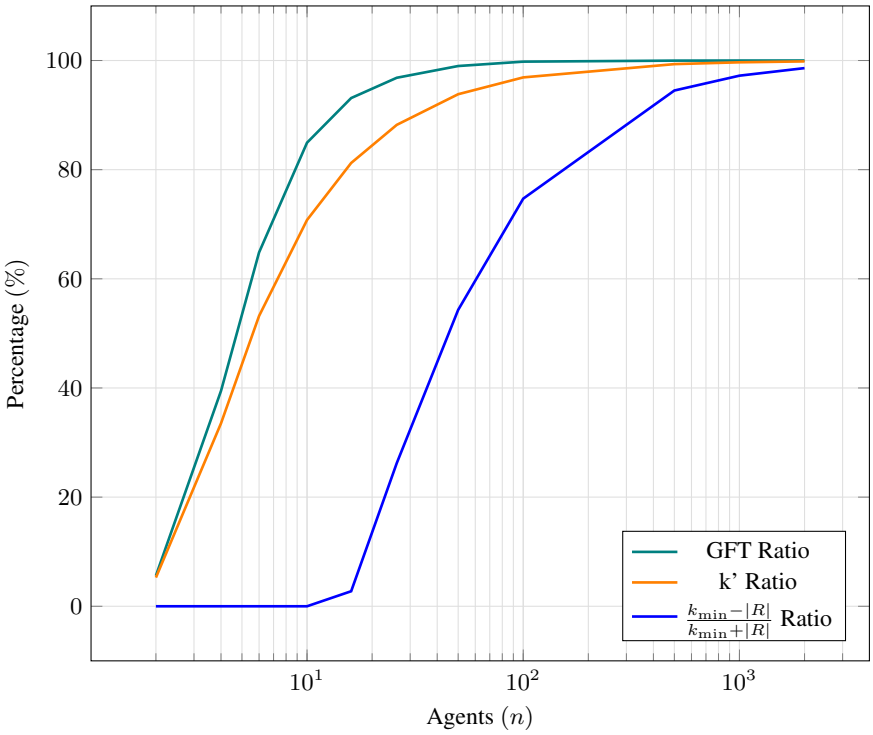


Figure 4: Graph of results from Table 10.

- There is a recipe $\mathbf{r}^e$ for each edge $e \in E$, defined as follows:

$$\mathbf{r}_i^e := \begin{cases} 1 & i \in e, \\ 0 & \text{otherwise.} \end{cases}$$

Since H is 3-uniform, each recipe contains exactly 3 ones and the other elements are zero. Therefore, the GFT of every PS is $3 \cdot 1/3 = 1$, and the GFT of every trade equals the number of trading PS. Since each category contains a single agent, each category must appear in at most one PS. Therefore, every feasible trade corresponds to a matching in H and vice-versa, so the problems are equivalent. □

Note that the best known polytime algorithm for 3-D matching attains $3/4$ of the optimum [15]; this illustrates the kind of approximation to the GFT that we can hope to obtain for general recipe-sets. Developing truthful mechanisms that attain such constant-factor approximations is another interesting future work direction.

F Limitations of our approach

The approximation ratio of our algorithm for general recipes (Theorem 3.11) depends on $|R|$. The reason is that Lemma 3.10, which bounds the difference between the optimal and the actual number of traders in each category, depends on $|R|$. In this section we show the tightness of Lemma 3.10. Specifically, we show examples in which the algorithm stops and the remaining agents in g are $m_g \geq k_g + |R| - 1$ or $m_g \leq k_g - |R| + 1$, which depends on the number of recipes.

We consider three variants of our approach, using three different ways of comparing c_g to m_g in Algorithm 3 step 2. Specifically, m_g can be rounded down (as in the original variant), or rounded

Table 11: Results of experiment with values chosen uniformly at random, and recipe-forest $\mathbf{R} = \{(1,2,0,0),(1,0,1,2)\}$.

	Optimal					Ascending Price					
n	k	$k_{\min}$	$k_{\max}$	LB	$OGFT$	k'	${k_{\min}}'$	${k_{\max}}'$	$\%k'$	GFT	$\%GFT$
2	0.32	0.31	0.31	0.000	83.1	0.04	0.04	0.04	13.001	20.9	25.190
4	1.08	0.86	0.87	0.000	385.6	0.42	0.40	0.40	39.032	225.1	58.374
6	1.74	1.06	1.17	0.000	711.7	0.98	0.81	0.82	56.138	537.7	75.547
10	2.99	1.24	1.90	0.000	1344.2	2.21	1.19	1.53	73.794	1213.1	90.244
16	4.84	1.90	2.96	0.000	2280.6	4.06	1.60	2.60	83.764	2193.3	96.172
26	7.96	3.24	4.72	23.685	3841.1	7.18	2.83	4.35	90.246	3787.2	98.595
50	15.40	6.55	8.84	53.249	7552.2	14.61	6.13	8.48	94.886	7524.0	99.627
100	30.90	13.49	17.41	74.185	15280.4	30.10	13.04	17.05	97.409	15265.6	99.902
500	154.89	68.72	86.16	94.344	77078.1	154.10	68.30	85.80	99.492	77075.5	99.996
1000	309.97	137.53	172.44	97.133	154394.8	309.18	137.11	172.07	99.744	154393.6	99.999
2000	620.14	275.38	344.76	98.557	309003.4	619.35	275.00	344.35	99.871	309002.9	99.999

Table 12: Results of experiment with values chosen uniformly at random, and wide recipe-tree (20 children, agent count of 20 each).

	Optimal					Ascending Price					
n	k	$k_{\min}$	$k_{\max}$	LB	$OGFT$	k'	${k_{\min}}'$	${k_{\max}}'$	$\%k'$	GFT	$\%GFT$
20	0.0	0.0	0.0	0.00	3	0.0	0.0	0.0	0.00	0	0.00
50	3.7	0.9	0.9	0.00	768	0.0	0.0	0.0	2.42	49	6.50
100	9.6	1.0	1.1	0.00	3949	4.6	1.0	1.0	48.34	2572	65.13
500	45.1	3.6	5.5	28.82	22369	41.1	3.0	5.1	91.16	22165	99.08
1000	90.3	7.7	10.3	58.84	44523	86.1	7.2	9.9	95.29	44420	99.76
2000	180.2	16.0	19.9	77.83	89245	175.9	15.5	19.5	97.61	89189	99.93
5000	451.5	42.1	48.1	90.93	223104	447.1	41.7	47.7	99.04	223082	99.99
10000	901.8	85.7	94.5	95.44	446385	897.7	85.3	94.1	99.55	446371	99.99
20000	1802.9	174.3	186.2	97.73	892285	1799.6	174.0	186.0	99.81	892273	99.99
50000	4512.8	441.1	461.2	99.09	2230206	4510.3	441.0	461.0	99.94	2230193	99.99
100000	9024.3	888.6	916.9	99.55	4456886	9022.9	888.4	917.1	99.98	4456876	99.99
200000	18072.0	1786.2	1828.3	99.77	8919549	18070.8	1785.7	1828.2	99.99	8919540	99.99
500000	45203.4	4483.9	4555.0	99.91	22294888	45202.8	4483.9	4554.6	99.99	22294881	99.99
1000000	90551.1	9007.5	9099.6	99.95	44590364	90550.7	9008.9	9095.6	99.99	44590363	99.99
2000000	180953.1	17996.1	18175.9	99.97	89176913	180952.6	17980.1	18180.1	99.99	89176909	99.99

up, or not rounded at all. We show that the dependence on $|R|$ exists in all variants. Removing this dependence (if at all possible) probably requires a different approach.

F.1 Rounding down

Consider a recipe-forest R with one tree, consisting of a root and $|R|$ children (leaves). All recipes require one agent from the root ($r_{root} = 1$) and two agents from a child ($r_g = 2$). The market contains the following agents:

- The root contains $|R|$ agents, each with the value 101.
- $|R| - 1$ children g' contain two agents, with values $-1, -90$.
- One child g contains $2 \cdot |R|$ agents, all with value -50.

The optimal trade has $|R|$ deals, with one deal per child. But Algorithm 3 removes all -90 agents in the first $|R| - 1$ rounds. Then the algorithm increases the price of child g until it reaches -50 and removes one agent from g. Now, $m_{root} = |R|$ and $c_{root} = \lfloor m_g/2 \rfloor = |R| - 1$, so the

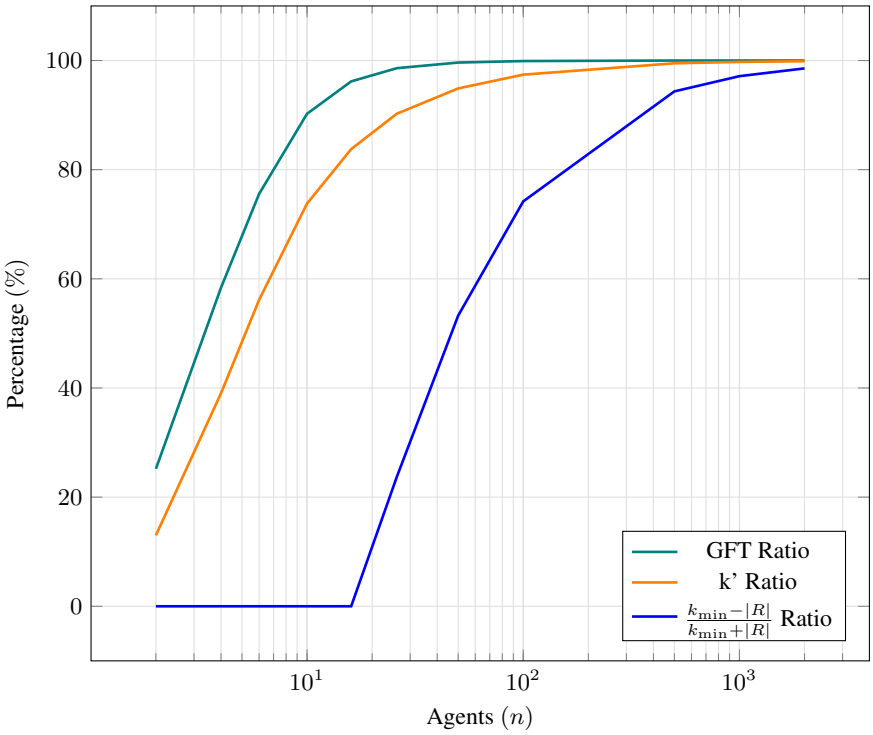


Figure 5: Graph of results from Table 11.

Table 13: Execution of Algorithm 2 on the market described in the first paragraph of subsection F.1 with $|R| = 3$

$\lvert M_i\rvert$	Compare	G^*	Price-increase stops when —	Updated prices	Price-sum
			Initial prices:	$-V, -V, -V, -V$	$-3V$
$3, 2, 2, 6$	$3/1 \leq \lfloor 2/2\rfloor + \lfloor 2/2\rfloor + \lfloor 6/2\rfloor$	$2, 3, 4$	Agent -90 exits from category 2	$-V, -90, -90, -90$	$-180 - V$
$3, 1, 2, 6$	$3/1 \leq \lfloor 1/2\rfloor + \lfloor 2/2\rfloor + \lfloor 6/2\rfloor$	$2, 3, 4$	Agent -90 exits from category 3	$-V, -90, -90, -90$	$-180 - V$
$3, 1, 1, 6$	$3/1 \leq \lfloor 1/2\rfloor + \lfloor 1/2\rfloor + \lfloor 6/2\rfloor$	$2, 3, 4$	Agent -50 exits from category 4	$-V, -50, -50, -50$	$-100 - V$
$3, 1, 1, 5$	$3/1 > \lfloor 1/2\rfloor + \lfloor 1/2\rfloor + \lfloor 5/2\rfloor$	1	price-sum crosses 0	$100, -50, -50, -50$	0

algorithm starts incrementing the price of the root category. It stops when p_{root} reaches 100. Now there are $|R| - 1$ deals in the whole tree, each deal contains one agent from the root and two agents of value -50 from child g. We have $k_g = 1$ optimal deal in this child, but the algorithm stops when there are $|M_g| = 2 \cdot |R| - 1$ agents which is $m_g = k_g + |R| - 1 - 1/r_g = |R| - 1/r_g$ deals, so $m_g \geq k_g + |R| - 1$. An example run is shown in Table 13 with $|R|$ = 3.

Consider now the same recipe-forest but the root category has $2 \cdot |R| - 1$ agents, each with the value 101. Each iteration removes one -90 agent from a child g' and the next iteration removes one agent from the root 101. The Algorithm loops until all -90 agents are removed from g' and $|R| - 1$ agents of 101 are removed from the root category. The algorithm then stops when the price reaches -50.5 in all children categories. In this case, the root has $k_{root} = 2 \cdot |R| - 1$ optimal deals, but the algorithm removes $|R| - 1$ agents from the root category. Now we have only $|R|$ agents left, which is $m_{root} = k_{root} - |R| + 1 = |R|$ deals instead of $2 \cdot |R| - 1$ optimal deals. So $m_g \leq k_g - |R| + 1$. An example run is shown in Table 14 with $|R|$ = 3.

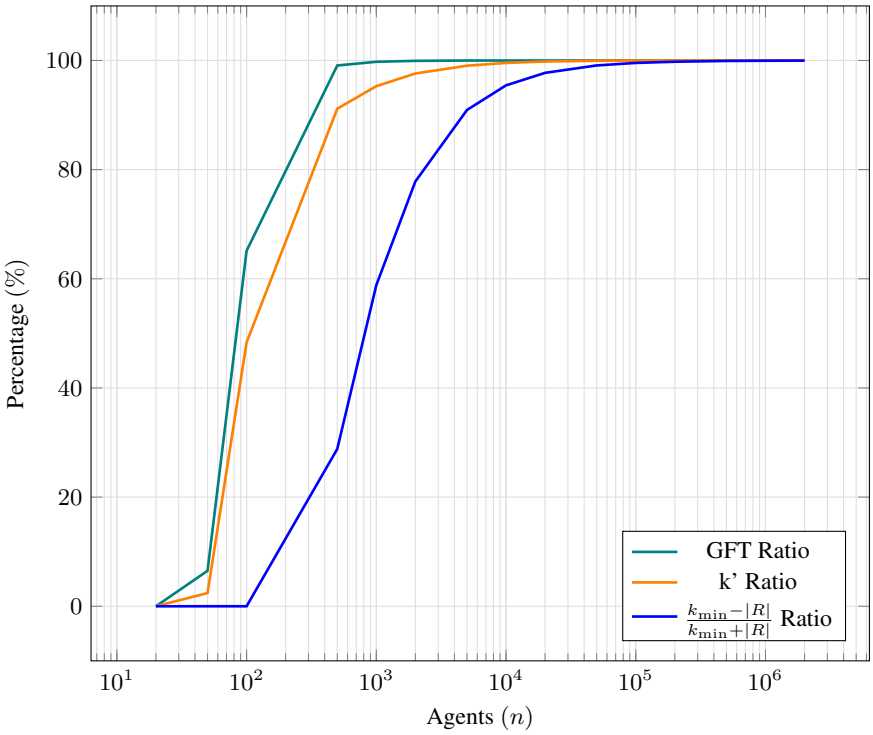


Figure 6: Graph of results from Table 12.

Table 14: Execution of Algorithm 2 on the market described in the second paragraph of subsection F.1 with $|R| = 3$

$\lvert M_i\rvert$	Compare	G^*	Price-increase stops when —	Updated prices	Price-sum
			Initial prices:	$-V, -V, -V, -V$	$-3V$
$5, 2, 2, 6$	$5/1 \le \lfloor 2/2\rfloor + \lfloor 2/2\rfloor + \lfloor 6/2\rfloor$	$2, 3, 4$	Agent -90 exits from category 2	$-V, -90, -90, -90$	$-180 - V$
$5, 1, 2, 6$	$5/1 > \lfloor 1/2\rfloor + \lfloor 2/2\rfloor + \lfloor 6/2\rfloor$	1	Agent 101 exits from root category 1	$101, -90, -90, -90$	-79
$4, 1, 2, 6$	$4/1 \le \lfloor 1/2\rfloor + \lfloor 2/2\rfloor + \lfloor 6/2\rfloor$	$2, 3, 4$	Agent -90 exits from category 3	$101, -90, -90, -90$	-79
$4, 1, 1, 6$	$4/1 > \lfloor 1/2\rfloor + \lfloor 1/2\rfloor + \lfloor 6/2\rfloor$	1	Agent 101 exits from root category 1	$101, -90, -90, -90$	-79
$3, 1, 1, 6$	$3/1 \le \lfloor 1/2\rfloor + \lfloor 1/2\rfloor + \lfloor 6/2\rfloor$	$2, 3, 4$	price-sum crosses 0	$100, -50.5, -50.5, -50.5$	0

F.2 Rounding up

Let us see what happens if in Algorithm 3 step 2 instead of rounding the $m_{g'}$ down, we round it up. Consider the same recipe-forest as the first one in subsection F.1. The root category has $|R|$ agents, each with the value 101. But each child g' has two agents with values $(-20, -90)$, The optimal trade has $|R|$ deals, all of them use the single category g. Algorithm 3 removes all -90 agents in the first $|R| - 1$ iterations. Then, the algorithm removes $2 \cdot |R|$ agents -50 from g. And then the algorithm stops when the price reaches 100 in the root category. For g we have $k_g = |R|$ optimal deals, but the algorithm stops when there are no agents left, $|M_g| = 0$ which is $m_g = k_g - |R| = 0$ deals. So $m_g \le k_g - |R| + 1$. An example run is shown in Table 15 with $|R| = 3$.

If we consider the same recipe-forest as before, but the root category has $2 \cdot |R| - 1$ agents, each with the value 101. Algorithm 3 removes all -90 agents in the first $|R| - 1$ iterations. Then, the algorithm removes two agents -50 from g. And then the algorithm stops when the price reaches 100 in the root category. In this case, the root has $k_{root} = |R|$ optimal deals, but the algorithm did not remove any agents from the root category. So we have $|M_{root}| = 2 \cdot |R| - 1$ agents left, which

Table 15: Execution of Algorithm 2 on the market described in the first paragraph of subsection F.2 with $|R| = 3$

$\lvert M_i\rvert$	Compare	G^*	Price-increase stops when —	Updated prices	Price-sum
			Initial prices:	$-V, -V, -V, -V$	$-3V$
$3, 2, 2, 6$	$3/1 \leq \lceil 2/2\rceil + \lceil 2/2\rceil + \lceil 6/2\rceil$	$2, 3, 4$	Agent -90 exits from category 2	$-V, -90, -90, -90$	$-180 - V$
$3, 1, 2, 6$	$3/1 \leq \lceil 1/2\rceil + \lceil 2/2\rceil + \lceil 6/2\rceil$	$2, 3, 4$	Agent -90 exits from category 3	$-V, -90, -90, -90$	$-180 - V$
$3, 1, 1, 6$	$3/1 \leq \lceil 1/2\rceil + \lceil 1/2\rceil + \lceil 6/2\rceil$	$2, 3, 4$	Agent -50 exits from category 4	$-V, -50, -50, -50$	$-100 - V$
$3, 1, 1, 5$	$3/1 \leq \lceil 1/2\rceil + \lceil 1/2\rceil + \lceil 5/2\rceil$	$2, 3, 4$	Agent -50 exits from category 4	$-V, -50, -50, -50$	$-100 - V$
$3, 1, 1, 4$	$3/1 \leq \lceil 1/2\rceil + \lceil 1/2\rceil + \lceil 4/2\rceil$	$2, 3, 4$	Agent -50 exits from category 4	$-V, -50, -50, -50$	$-100 - V$
$3, 1, 1, 3$	$3/1 \leq \lceil 1/2\rceil + \lceil 1/2\rceil + \lceil 3/2\rceil$	$2, 3, 4$	Agent -50 exits from category 4	$-V, -50, -50, -50$	$-100 - V$
$3, 1, 1, 2$	$3/1 \leq \lceil 1/2\rceil + \lceil 1/2\rceil + \lceil 2/2\rceil$	$2, 3, 4$	Agent -50 exits from category 4	$-V, -50, -50, -50$	$-100 - V$
$3, 1, 1, 1$	$3/1 \leq \lceil 1/2\rceil + \lceil 1/2\rceil + \lceil 1/2\rceil$	$2, 3, 4$	Agent -50 exits from category 4	$-V, -50, -50, -50$	$-100 - V$
$3, 1, 1, 0$	$3/1 > \lceil 1/2\rceil + \lceil 1/2\rceil + \lceil 0/2\rceil$	1	price-sum crosses 0	$100, -50, -50, -50$	0

Table 16: Execution of Algorithm 2 on the market described in the second paragraph of subsection F.2 with $|R| = 3$

$\lvert M_i\rvert$	Compare	G^*	Price-increase stops when —	Updated prices	Price-sum
			Initial prices:	$-V, -V, -V, -V$	$-3V$
$5, 2, 2, 6$	$5/1 \leq \lceil 2/2\rceil + \lceil 2/2\rceil + \lceil 6/2\rceil$	$2, 3, 4$	Agent -90 exits from category 2	$-V, -90, -90, -90$	$-180 - V$
$5, 1, 2, 6$	$5/1 \leq \lceil 1/2\rceil + \lceil 2/2\rceil + \lceil 6/2\rceil$	$2, 3, 4$	Agent -90 exits from category 3	$-V, -90, -90, -90$	$-180 - V$
$5, 1, 1, 6$	$5/1 \leq \lceil 1/2\rceil + \lceil 1/2\rceil + \lceil 6/2\rceil$	$2, 3, 4$	Agent -50 exits from category 4	$-V, -50, -50, -50$	$-100 - V$
$5, 1, 1, 5$	$5/1 \leq \lceil 1/2\rceil + \lceil 1/2\rceil + \lceil 5/2\rceil$	$2, 3, 4$	Agent -50 exits from category 4	$-V, -50, -50, -50$	$-100 - V$
$5, 1, 1, 4$	$5/1 > \lceil 1/2\rceil + \lceil 1/2\rceil + \lceil 4/2\rceil$	1	price-sum crosses 0	$100, -50, -50, -50$	0

is $m_{root} = k_{root} + |R| - 1 = 2 \cdot |R| - 1$ deals instead of $|R|$ optimal deals. So $m_g \geq k_g + |R| - 1$. An example run is shown in Table 16 with $|R|$ = 3.

F.3 No rounding

Let us see what happens if in Algorithm 3 step 2 instead of rounding the $m_{g'}$ to any direction, we do not round it. Consider a recipe-forest R with one tree, consisting of a root and $|R|$ children as leaves (assuming $|R| \geq 2$). Each deal contains one root agent $r_{root} = 1$, and $|R|$ agents from a child $r_g = r_{g'} = |R|$. The agents' values are:

- The root contains $|R|$ agents, each with the value $|R|^3$.
- $|R| - 1$ children g' contain $|R|$ agents each, $|R| - 1$ agents with the values $-|R|^2$ and one agent with the value -1. (each g' has a total value of $-|R|^3 + |R|^2 - 1$)

Table 17: Execution of Algorithm 2 on the market described in the first paragraph of subsection F.3 with $|R| = 4$

$\lvert M_i\rvert$	Compare	G^*	Price-increase stops when —	Updated prices	Price-sum
			Initial prices:	$-V, -V, -V, -V, -V$	$-5V$
$4, 4, 4, 4, 13$	$4/1 \leq 4/4 + 4/4 + 4/4 + 13/4$	$2, 3, 4, 5$	Agent -16 exits from category 2	$-V, -16, -16, -16, -16$	$-64 - V$
$4, 3, 4, 4, 13$	$4/1 \leq 3/4 + 4/4 + 4/4 + 13/4$	$2, 3, 4, 5$	Agent -16 exits from category 2	$-V, -16, -16, -16, -16$	$-64 - V$
$4, 2, 4, 4, 13$	$4/1 \leq 2/4 + 4/4 + 4/4 + 13/4$	$2, 3, 4, 5$	Agent -16 exits from category 2	$-V, -16, -16, -16, -16$	$-64 - V$
$4, 1, 4, 4, 13$	$4/1 \leq 1/4 + 4/4 + 4/4 + 13/4$	$2, 3, 4, 5$	Agent -16 exits from category 3	$-V, -16, -16, -16, -16$	$-64 - V$
$4, 1, 3, 4, 13$	$4/1 \leq 1/4 + 3/4 + 4/4 + 13/4$	$2, 3, 4, 5$	Agent -16 exits from category 3	$-V, -16, -16, -16, -16$	$-64 - V$
$4, 1, 2, 4, 13$	$4/1 \leq 1/4 + 2/4 + 4/4 + 13/4$	$2, 3, 4, 5$	Agent -16 exits from category 3	$-V, -16, -16, -16, -16$	$-64 - V$
$4, 1, 1, 4, 13$	$4/1 \leq 1/4 + 1/4 + 4/4 + 13/4$	$2, 3, 4, 5$	Agent -16 exits from category 4	$-V, -16, -16, -16, -16$	$-64 - V$
$4, 1, 1, 3, 13$	$4/1 \leq 1/4 + 1/4 + 3/4 + 13/4$	$2, 3, 4, 5$	Agent -16 exits from category 4	$-V, -16, -16, -16, -16$	$-64 - V$
$4, 1, 1, 2, 13$	$4/1 \leq 1/4 + 1/4 + 2/4 + 13/4$	$2, 3, 4, 5$	Agent -16 exits from category 4	$-V, -16, -16, -16, -16$	$-64 - V$
$4, 1, 1, 1, 13$	$4/1 \leq 1/4 + 1/4 + 1/4 + 13/4$	$2, 3, 4, 5$	Agent -15 exits from category 5	$-V, -15, -15, -15, -15$	$-60 - V$
$4, 1, 1, 1, 12$	$4/1 > 1/4 + 1/4 + 1/4 + 12/4$	1	price-sum crosses 0	$60, -15, -15, -15, -15$	0

Table 18: Execution of Algorithm 2 on the market described in the second paragraph of subsection F.3 with $|R| = 4$

$\lvert M_i\rvert$	Compare	G^*	Price-increase stops when —	Updated prices	Price-sum
			Initial prices:	$-V, -V, -V, -V, -V$	$-5V$
$7, 4, 4, 4, 16$	$7/1 \leq 4/4 + 4/4 + 4/4 + 16/4$	$2, 3, 4, 5$	Agent -4 exits from category 2	$-V, -4, -4, -4, -4$	$-16 - V$
$7, 3, 4, 4, 16$	$7/1 > 3/4 + 4/4 + 4/4 + 16/4$	1	Agent 14 exits from category 1	$14, -4, -4, -4, -4$	-9
$6, 3, 4, 4, 16$	$6/1 \leq 3/4 + 4/4 + 4/4 + 16/4$	$2, 3, 4, 5$	Agent -4 exits from category 2	$14, -4, -4, -4, -4$	-9
$6, 2, 4, 4, 16$	$6/1 \leq 2/4 + 4/4 + 4/4 + 16/4$	$2, 3, 4, 5$	Agent -4 exits from category 2	$14, -4, -4, -4, -4$	-9
$6, 1, 4, 4, 16$	$6/1 \leq 1/4 + 4/4 + 4/4 + 16/4$	$2, 3, 4, 5$	Agent -4 exits from category 3	$14, -4, -4, -4, -4$	-9
$6, 1, 3, 4, 16$	$6/1 \leq 1/4 + 3/4 + 4/4 + 16/4$	$2, 3, 4, 5$	Agent -4 exits from category 3	$14, -4, -4, -4, -4$	-9
$6, 1, 2, 4, 16$	$6/1 > 1/4 + 2/4 + 4/4 + 16/4$	1	Agent 14 exits from category 1	$14, -4, -4, -4, -4$	-9
$5, 1, 2, 4, 16$	$5/1 \leq 1/4 + 2/4 + 4/4 + 16/4$	$2, 3, 4, 5$	Agent -4 exits from category 3	$14, -4, -4, -4, -4$	-9
$5, 1, 1, 4, 16$	$5/1 \leq 1/4 + 1/4 + 4/4 + 16/4$	$2, 3, 4, 5$	Agent -4 exits from category 4	$14, -4, -4, -4, -4$	-9
$5, 1, 1, 3, 16$	$5/1 \leq 1/4 + 1/4 + 3/4 + 16/4$	$2, 3, 4, 5$	Agent -4 exits from category 4	$14, -4, -4, -4, -4$	-9
$5, 1, 1, 2, 16$	$5/1 \leq 1/4 + 1/4 + 2/4 + 16/4$	$2, 3, 4, 5$	Agent -4 exits from category 4	$14, -4, -4, -4, -4$	-9
$5, 1, 1, 1, 16$	$5/1 > 1/4 + 1/4 + 1/4 + 16/4$	1	Agent 14 exits from category 1	$14, -4, -4, -4, -4$	-9
$4, 1, 1, 1, 16$	$4/1 \leq 1/4 + 1/4 + 1/4 + 16/4$	$2, 3, 4, 5$	price-sum crosses 0	$14, -3.5, -3.5, -3.5, -3.5$	0

- One child g has $|R|^2 - |R| + 1$ agents with the values $-|R|^2 + 1$. (each deal in g has a total value of $-|R|^3 + |R|$)

The optimal trade has $|R| - 1$ deals with the children g'. and one deal with the child g

The first $|R|^2 - |R|$ iterations remove all agents with values $-|R|^2$ from all children g'. Then the algorithm removes one agent from g when the price reaches $-|R|^2 + 1$. Then the algorithm stops when the price reaches $|R|^3 - |R|$ in the root's category. Now we have $|R| - 1$ deals in the whole tree. Each deal contains one agent from the root and random $|R|^2 - |R|$ agents from child g. For g we have $k_g = 1$ optimal deal, but the algorithm stops when there are $|M_g| = |R|^2 - |R|$ agents which is $m_g = k_g + (|R|^2 - |R|)/r_g - 1 = |R| - 1$ deals instead of one optimal deal in g. So $m_g \geq k_g + |R| - 1$. An example run is shown in Table 17 with $|R|$ = 4.

Consider now the same recipe-forest as before, but with the following values (assuming $|R| > 2$ and is even):

- The root contains $2 \cdot |R| - 1$ agents with value $|R|^2 - |R| + 2$.
- $|R| - 1$ children g' contain $|R|$ agents each, $|R| - 1$ agents with the values $-|R|$ and one agent with the value -1. (each g' has a total value of $-|R|^2 + |R| - 1$)
- One child g has $|R|^2$ agents with the values $-|R|/2$. (each deal in g has a total value of $-|R|^2/2$)

The first iteration removes an agent with the value $-|R|$ from any child g' and the next iteration removes one agent from the root. Afterwards, the algorithm does the following until all g' categories have only one agent left: it removes $|R|$ agents with the values $-|R|$ from any child g' and then removes one agent from the root. The algorithm does this until all agents with values $-|R|$ are removed from all g' and $|R| - 1$ agents are removed from the root category. Then the algorithm stops when the price reaches $-|R| + \frac{|R|-2}{|R|}$ in the children categories before removing agent $-|R|/2$ from category g. In this case, the root has $k_{root} = 2 \cdot |R| - 1$ optimal deals, but the algorithm removes $|R| - 1$ agents from the root category. So only $|M_{root}| = |R|$ agents are left, which is $m_{root} = k_{root} - |R| + 1 = |R|$ deals instead of $2 \cdot |R| - 1$ optimal deals. So $m_g \leq k_g - |R| + 1$. An example run is shown in Table 18 with $|R|$ = 4.

G Discussion and Future Work

Future research should explore the GFT approximation bounds and the gap with our current approximation if any.

Designing obviously-truthful, strongly-budget-balanced and approximately-efficient auctions is a challenging task even in a single-recipe market. This paper generalizes this difficult task to multiple-recipe markets. Nevertheless, our model does not capture all multiple-recipe market scenarios. We discuss next the challenges of extending our model further.

G.1 Beyond Recipe-Forests

Our mechanisms assume that the set of recipes can be arranged as a forest. In particular, it means that the structure is acyclic.[15] With cyclic recipe sets, the main challenge is maintaining budget balance. Our current approach attains budget balance in all recipes simultaneously. But in the cyclic recipe-set $(1,1,0),(0,1,1),(1,0,1)$,[16] to attain budget balance in all recipes simultaneously, we need $p_1+p_2 = p_2+p_3 = p_3+p_1 = 0$ (where p_g is the price in category g), and the only solution is $p_1 = p_2 = p_3 = 0$. Clearly, no good approximation of the GFT is possible when all prices are fixed in advance. As another example, consider the cyclic recipe-set $(1,1,0,0),(0,1,1,0),(1,0,1,1)$. To ensure that the price-sum in all recipes is the same throughout the algorithm, we would need to increment either p_2, p_4 or p_1, p_2, p_3 — these are the only sets of categories that contain the same number of categories in each recipe. This means that p_2 is always incremented, so all agents from this category might leave the market before the algorithm completes.

Moreover, our recipe forests do not capture all acyclic structures. For example, the recipe-sets $(1,1,0),(1,1,1)$ and $(1,1,0,0),(1,0,1,0),(0,0,1,1)$ are acyclic, but not a recipe-forest by our definition, since they cannot be arranged such that every recipe corresponds to a root-leaf path.

Developing efficient auctions for more complex markets would probably require new techniques.

G.2 Beyond Disjoint Categories

Our mechanisms assume that each agent belongs to a single category. Extending our approach to handle the setting where an agent can belong to several categories is challenging, as the market will not maintain the truthfulness property. We illustrate the challenge with the following example: the recipes are $(1,1,0)$ and $(1,0,1)$. Category 1 contains four buyers with a value of $+20$; category 2 contains sellers s_1 and s_2, while category 3 contains sellers s_1 and s_3, so that s_1 belongs to both seller categories. In each iteration, our algorithm increments either p_1 (the root category price) or p_2, p_3 (the leaf categories prices). Initially, since $c_{g_0} = 4 \geq m_{g_0}$, the algorithm increments p_2, p_3. The seller s_1 can lie and exit category 2 when both prices arrive at -15, for example, but remain in category 3. Then, since $c_{g_0} = 3 < m_{g_0}$, the algorithm will increment p_1 until it arrives at $+15$, as the price-sum equals 0. Then, s_1 sells a single unit at a price of 15 instead of selling two units for a price of 10, so his utility increases from 0 to $+5$ due to the manipulation.

G.3 Beyond Identical Multiplicities

Our ascending auction requires that every category $g \in G$ appears in all recipes with the same multiplicity (see Section 2.2). This assumption is used in our proofs in Section 3.

The main challenge in extending our ascending-prices auction to different multiplicities is attaining a high GFT. To illustrate, consider first a special case of our market, where there are m

[15] One could object to cyclic recipe-sets based on economic arguments. Consider a recipe-tree with one parent node and two child nodes. The tree represents two different recipes, each with a parent node category that complements a child node category. The two recipes have mutually substitutable child nodes. Introducing a cycle, i.e., connecting the two child nodes in the graph to form a recipe, would create categories that simultaneously complement and substitute each other. Economically, such a recipe model is unavailing.

[16] Note that with this recipe-set, the optimal trade can be computed efficiently as follows. Construct a graph in which each node is an agent, and there is an edge between every two agents in different categories, where the edge weight is the sum of the values of the two agents. The optimal trade corresponds to a maximum-weight matching in this graph.

sellers with value 0, and $n > m$ categories of buyers, where category g has a single buyer with value $v_g > 0$. There are n recipes, where recipe g contains one seller and one buyer of category g. The ascending price mechanism finds the optimal trade in a greedy way: it increases the prices, each time eliminating a low-value buyer until only the m high-value buyers remain. Now, suppose that in each recipe g, the sellers have a *different* multiplicity, r_g. Then, computing the optimal trade is equivalent to a knapsack problem: each recipe g is an item with weight r_g and value v_g, and the knapsack capacity is m. It is known that greedy algorithms do not attain a good approximation for the knapsack problem. Since an ascending-price auction selects buyers in a greedy way, we believe that such an auction will not attain a good approximation of the GFT.[17]

G.4 Transaction Costs

In general, each procurement set may have a different cost-of-transaction, depending on the geographic locations of the agents in the PS and other factors. Such transaction costs make the computation of the optimal trade difficult, even before strategic considerations and even when all transaction costs are common knowledge.

Given the above, it is likely that without any restrictions on the transaction costs, there might be no mechanism that satisfies all the desirable properties of Theorem 3.2. It would be interesting to see whether a mechanism with transaction costs can be found even under some natural restrictions on the transaction costs, such as those described by [12].

Acknowledgments

A preliminary version was presented in the EUMAS 2021 conference. We are grateful to three referees of EUMAS 2021 for their helpful comments.
The second author would like to thank the Ministry of Science, Technology and Space Binational Israel-Taiwan grant, number 3-16542.

Dvir Gilor
The Open University of Israel
Raanana, Israel
Email: dvir@gilor.com

Rica Gonen
The Open University of Israel
Raanana, Israel
Email: ricagonen@gmail.com

Erel Segal-Halevi
Ariel University
Ariel, Israel
Email: erelsgl@gmail.com

[17] Interestingly, [20] show that, for binary allocation problems, every obviously-truthful mechanism must use a greedy algorithm.