MATHEMATICS OF OPERATIONS RESEARCH
Vol. 00, No. 0, Xxxxxx 20xx, pp. xxx–xxx
ISSN 0364-765X | EISSN 1526-5471 |xx|0000|0xxx

DOI 10.1287/moor.xxxx.xxxx

Auction Algorithms for Market Equilibrium

Rahul Garg
IBM India Research Lab., Block-I, IIT Campus, Hauz Khas, New Delhi, INDIA - 110016.
email: grahul@in.ibm.com

Sanjiv Kapoor
Illinois Institute of Technology, Chicago, USA
email: kapoor@iit.edu

In this paper we study algorithms for computing market equilibrium in markets with linear utility functions. The buyers in the market have an initial endowment given by a portfolio of goods. The *market equilibrium problem* is to compute a price vector which ensures market clearing, i.e. the demand of a positively priced good equals its supply, and given the prices, each buyer maximizes its utility. The problem is of considerable interest in Economics. This paper presents a formulation of the market equilibrium problem as a parameterized linear program. We construct a family of duals corrresponding to these parameterized linear programs and show that finding the market equilibrium is the same as finding a linear-program from this family of linear programs. The market clearing conditions arise naturally from complementary slackness conditions. We then define an auction mechanism which computes prices such that approximate market clearing is achieved. The algorithm we obtain outperforms previously known methods.

Key words: Market Equilibrium, Auction Algorithm, Walrasian Model

MSC2000 Subject Classification: Primary: 91B50, 91B52; Secondary: 91B26, 68W40

OR/MS subject classification: Primary: Economics, Algorithms; Secondary: Bidding/auctions

History: Received: Xxxx xx, xxxx; revised: Yyyyyy yy, yyyy and Zzzzzz zz, zzzz.

1. Introduction In this paper we study algorithms for computing market equilibrium in markets with linear utility functions. Consider a market comprising n traders and m goods. Each trader has an endowment given by a portfolio of goods. A finite quantity of each good is available and is assumed to be divisible. Further, each (trader, good) pair has an associated utility function. This function is assumed to be non-negative and linear. The *market equilibrium problem* is to compute a price vector and a feasible assignment of goods to buyers such that no buyer is induced to change his assignments with respect to the given set of prices and market clearing is achieved, i.e. there is no surplus or deficit of the goods.

The problem is of considerable interest in Economics and was first proposed in 1891 by Fisher [4]. Independently, Walras (1894) proposed the notion of general equilibrium. Walras proposed that a general equilibrium could be achieved by a price-adjustment process called *tatonnement* [21]. Establishing that an equilibrium can be achieved is a problem of considerable interest in descriptive and normative economics. The existence of equilibrium prices in a general setting was established by Arrow and Debreu [2] in 1959. The proof is non-constructive and the natural question is the existence of an efficient computation process which establishes equilibrium. The model proposed by Fisher is more amenable to computational methods. In Fisher's model, buyers (traders) initially have endowment of money while the sellers initially have goods. Moreover, the buyers do not have any value for money and the sellers do not have any value for the goods. The more general model considered by us was proposed by Walras. In this model the traders have an initial endowment of goods (instead of money). The amount of money available for use by the trader is dependent on the final price of the goods.

Efficient computation of the market equilibrium is an important problem highlighted recently by Papadimitriou [18]. Computation of market equilibrium has been of considerable interest in the history of micro-economics. In 1891 Fisher presented a design of a hydraulic apparatus for calculating equilibrium prices when utilities are separable, additive, monotone and concave. Around the same time, Walras

suggested that market equilibrium prices may be discovered by a price adjustment process called *tatonnement*. In 1959 Arrow et al. [1], showed the stability of the price adjustment process for a large class of utility functions which satisfied the property of *gross substitutability*. They showed that

$$\underline{P}^* Z(\underline{P}) \geq 0, \ \forall \underline{P}$$

where $\underline{P}^* > 0$ is the vector of equilibrium prices, $\underline{P}$ is a price vector and $Z(\underline{P})$ is a homogeneous (invariant under scaling) excess demand function. Using this property, algorithms utilizing circumscribed and inscribed ellipsoids have been discussed in [17, 19]. The above characterization leads to an ellipsoid method for computation of market equilibrium in polynomial time. This approach has been surveyed in a recent note by Codenotti et al. [6].

Eaves [11] formulated the market equilibrium problem with linear utilities as a linear complementarity problem and used Lemke's algorithm to solve it. Eisenberg and Gale [12] have reduced the market equilibrium problem for the Fisher model with linear utilities, to an optimization problem. These results were derived for the Fisher model. In 1983, Nenakov and Primak showed how to solve the equilibrium problem in the general Walrasian model via a convex program [16]. Recently Jain [13] has also devised a similar convex inequality system for the general Walrasian model when the utilities are linear. The inequalities that characterize the solution are similar to the complementary slackness conditions that arise in the program of Eisenberg and Gale [12]. Using a similar approach and the model of Eisenberg and Gale [12], Ye [22] claims interior point methods for both the Fischer and the Walrasian models in the case of linear utility functions, resulting in finding the exact solution to the problem in $O(n^4 L)$ arithmetic operations. A bound on the precision required for executing these operations has not been provided (but should be better than the ellipsoid method).

Combinatorial methods to solve this problem are of importance as they avoid the numerical issues involved with the ellipsoid method and interior point algorithms. A polynomial time combinatorial algorithm for the Fischer model has been proposed in Devanur et al. [9] in 2002. In this paper we continue the combinatorial study and devise efficient approximation algorithms to solve the equilibrium problem in the general Walrasian model. The algorithms are based on an auction mechanism resulting in a *tatonnement* process. To model the problem, we formulate a family of linear programs parameterized by the price vector. We construct the dual of these programs. The dual program corresponding to a market clearing price has an optimal solution where dual variables corresponding to a family of inequalities in the primal program, are all equal to zero. We show that whenever these dual variables are zero, the corresponding price vector achieves market clearing. The market clearing conditions arise naturally from the complementary slackness conditions of the parameterized linear program.

We use the primal-dual formulation to define an auction mechanism which computes a $(1+\epsilon)$-approximate market equilibrium. In such an equilibrium, no buyer gains more than a factor of ϵ utility by switching from the currently assigned goods, the uncleared surplus of any buyer is less than ϵ times its total final endowment and at least a factor $1/(1+\epsilon)$ of all the goods are sold. The proposed algorithm resembles a primal-dual mechanism similar to Kuhn's methodology for bipartite matching [15]. This can also be viewed as an auction mechanism similar to [7, 3] or an efficient tatonnement type process [20, 5] for computing approximate market clearing.

After the development of this algorithm we have become aware of two approximation algorithms developed around the same time for this problem. The first algorithm [14] provides an approximation using the framework described in [9]. This algorithm achieves complete market clearing and approximate the optimality of the solution. The second algorithm [10] removes the dependence on the sizes of the numbers to achieve a strongly polynomial algorithm. The notion of the approximation achieved in the paper [10] is that of bounding the deficiency or surplus of the goods in terms of their monetary value.

The time complexity of our first algorithm is $O(\frac{1}{\epsilon^2} nm \log(\frac{p_{max} a}{\epsilon a_{min}}) \log p_{max})$, where $a = \sum_{j=1}^{m} a_j$ the sum of all available goods, p_{max} is the largest price (assuming that the smallest price is unity), $a_{min} = \min_{i,j} a_{ij} > 0$ is the smallest quantity of a good endowed to a trader, n is the number of traders and m is the number of goods in the market. The algorithm achieves approximate market clearing in terms of the quantity of each available good and in disposing the endowment of each trader while achieving approximate optimality. We also give another approximation algorithm, which achieves exact market clearing and approximate optimality and is of complexity $O(\frac{1}{\epsilon}(nm^2 + mn^2) \log p_{max})$, replacing a factor of $1/\epsilon$ by m, which would be an improvement whenever $1/\epsilon > m$. The algorithm is polynomial in all the parameters of the problem, except in the specification length of the error tolerance ϵ.

In comparison, the algorithm presented in [9] computes exact market clearing prices in the Fisher model, which is a special case of the model we consider in this paper, using $O(n^4(\log n + n\log U + \log M))$ *max-flow computations*, where M is the total amount of money available in the market and $U = \max_{ij} v_{ij}$, is the maximum valuation of an good by a trader. The algorithm in [14] solves the problem in the general Walrasian model with complexity $O(\frac{m^4}{\epsilon}(\log m + m\log U + \log M))$ *max-flow computations* where U and M depend on the utilities and endowments, respectively. And finally the algorithm in [10] solves the problem in $O(\frac{m^4}{\epsilon}\log\frac{m}{\epsilon})$ *max-flow computations*. While the time-complexity of these methods involves large polynomials ($O(\frac{m^4 e(m+n)}{\epsilon}\log\frac{m}{\epsilon})$ assuming max-flow computations are $O(e(m+n))$, where $e < mn$ is the number of edges in the graph), these solutions provide very interesting insights into this problem.

Apart from the improved complexity provided in this paper, the technique proposed is of particular interest due to its decentralized nature and a natural auction interpretation. Moreover, if we aim to satisfy approximation conditions similar to that of [10] then the complexity of our method becomes $O(\frac{1}{\epsilon}(nm^2 + mn^2)\log\frac{m}{\epsilon})$.

We begin with the market model in Section 2 and present a parameterized linear programming formulation in Section 3. We present our basic auction algorithm for approximating market equilibrium in Section 4 and the faster auction algorithm in Section 5. In Section 6 we show how the strong conditions of approximate market equilibrium (defined in Section 2) can be satisfied by adding another phase to the algorithms. In Section 7 we discuss how the largest price p_{max} can be bounded for different definitions of approximate market equilibrium. We conclude in Section 8.

2. Market Model We consider the generalized market model with linear utilities. The market consists of a set of m goods (S) and a set of n traders (T). Trader i has an initial endowment a_{ij} of good j. The total amount of good j available in the market is given by $a_j = \sum_{i=1}^{n} a_{ij}$. Assume w.l.o.g that $a_j > 0$ else the good can be removed from the market. The utilities of the traders on these goods are assumed to be linear. Let v_{ij} be the per-unit utility of trader i on good j. We make the standard assumption that $v_{ij} \geq 0$. The traders exchange their goods so as to maximize their individual utilities.

Let the prices of the goods be represented in terms of an abstract currency. Given the prices $p_1, p_2, \ldots, p_m$ of the m goods, the amount of money which trader i can get (by selling these goods) is given by $\sum_{j=1}^{m} a_{ij}p_j$. Using this money, the trader buys goods that maximize its utility. Let x_{ij} represent the amount of good j obtained by trader i. Define $\underline{X}_i = (x_{i1}, x_{i2}, \ldots, x_{im})$. Trader i gets an allocation $\underline{X}_i$ that solves the following programs (LP_i):

$$\begin{aligned} \max & \sum_{j=1}^{m} v_{ij}x_{ij} \\ \text{Subject to: } & \sum_{j=1}^{m} x_{ij}p_j \leq \sum_{j=1}^{m} a_{ij}p_j \\ & x_{ij} \geq 0 \end{aligned} \tag{1}$$

Let $\underline{P}$ represent the $m \times 1$ vector of prices and $\underline{X}$ represent the $n \times m$ matrix of allocations, where the $(i,j)^{th}$ entry is x_{ij}. The pair $(\underline{X}, \underline{P})$ forms a market equilibrium iff (a) there is neither a surplus nor a deficiency of any positively priced good; (b) all the traders buy goods that maximize their utility. The prices $\underline{P}$ are called market clearing prices and $\underline{X}$ is called an equilibrium allocation.

We assume that there exists a market equilibrium where all the prices are strictly positive. This assumption can be removed by a variety of methods. In Section 7 we outline some methods that resolve this issue.

The condition (a) for market equilibrium can now be mathematically written as:

$$\forall j : \sum_{i=1}^{n} x_{ij} = a_j \tag{2}$$

Since $v_{ij} \geq 0$ and $p_j > 0$, condition (b) can be equivalently written as follows:

$$\forall i : \sum_{j=1}^{m} x_{ij}p_j = \sum_{j=1}^{m} a_{ij}p_j \tag{3}$$

$$\begin{aligned} \forall i,j : x_{ij} > 0 \Rightarrow v_{ij}/p_j &\geq v_{ik}/p_k, \forall k \quad (4) \\ x_{ij} &\geq 0 \end{aligned}$$

Equations (3) and (4) can be derived from the dual program and complementary slackness conditions of the program (LP_i). Equations (4) imply that every trader i is allocated only those goods j that maximize the corresponding v_{ij}/p_j.

It may be noted that the Fisher's model used in [9] is a special case of the above model, where money is also assumed to be a "good". The traders are partitioned into buyers and sellers. Initially the buyers are endowed with money and the sellers are endowed with goods. The utility of the traders on money is zero, and the utility of the sellers on all the goods is zero. The sellers have unit utilities for money. With these assumptions it can be observed that the conditions (2), (3) and (4) translate into the market clearing conditions for the model considered in [9].

A solution may be defined as an approximate market equilibrium if conditions (a) and (b) are satisfied approximately (see e.g. [8]). To achieve this, we work with the following definition of approximate market equilibrium.

We call a pair $(\underline{X}, \underline{P})$ as $(1+\epsilon)$-*approximate market equilibrium* if each of the conditions (2), (3) and (4) are satisfied to within a factor of $(1+\epsilon)$ i.e.,

$$\forall j : \frac{a_j}{1+\epsilon} \leq \sum_{i=1}^{n} x_{ij} \leq a_j \quad (5)$$

$$\forall i : (1-\epsilon)\sum_{j=1}^{m} a_{ij}p_j \leq \sum_{j=1}^{m} x_{ij}p_j \leq (1+\epsilon)\sum_{j=1}^{m} a_{ij}p_j \quad (6)$$

$$\forall i,j : x_{ij} > 0 \Rightarrow v_{ij}/p_j \geq v_{ik}/((1+\epsilon)p_k),\ \forall k \quad (7)$$

Note that the above conditions ensure that the allocations $X_i/(1+\epsilon)$ solve (LP_i) to within $(1+\epsilon)^2/(1-\epsilon)$ of optimal. Also, the above conditions of approximate market equilibrium are stronger than those mentioned in [8]. In general, a solution satisfying (a) and (b) approximately may not satisfy (7).

We first use the following weaker version of (6) to develop our algorithm:

$$\forall i : \sum_{j=1}^{m} x_{ij}p_j \leq (1+\epsilon)\sum_{j=1}^{m} a_{ij}p_j \quad (8)$$

We subsequently modify our algorithm to obtain a solution that satisfies (6).

3. A Parameterized Linear Programming Formulation The market equilibrium conditions can be written as a solution to a specific primal-dual program. Consider the following program $LP(\underline{P})$:

$$\text{Maximize} \sum_{i=1}^{n}\sum_{j=1}^{m} v_{ij}x_{ij}$$

subject to:

$$\begin{aligned} \forall j : \sum_{i=1}^{n} x_{ij} &= a_j \quad (9) \\ \forall i : \sum_{j=1}^{m} x_{ij}p_j &= \sum_{j=1}^{m} a_{ij}p_j \quad (10) \\ x_{ij} &\geq 0 \end{aligned}$$

If the price p_j is assumed to be fixed for all j, then the above program ($LP(\underline{P})$) becomes linear. We consider the dual of this linear program (using Lagrangian multipliers β_j for (9) and α_i for (10)) $DP(\underline{P})$:

$$\text{Minimize} \sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}p_j\alpha_i + \sum_{j=1}^{m} a_j\beta_j$$

Subject to:

$$\forall i,j : \alpha_i p_j + \beta_j \geq v_{ij} \tag{11}$$

This gives a family of primal-dual linear programs $LP(\underline{P})$ and $DP(\underline{P})$, one for each value of the price vector $\underline{P}$. From the theory of duality, the value of a feasible primal solution is always less than or equal to that of a feasible dual solution. Moreover, these values are equal when the feasible primal and dual solutions satisfy the following complementary slackness conditions:

$$\forall i,j : x_{ij} > 0 \quad \Rightarrow \quad \alpha_i p_j + \beta_j = v_{ij} \tag{12}$$

The following result relates the optimal solution of the above programs to the market clearing prices.

LEMMA 3.1 *A price vector $\underline{P}$ such that $p_j > 0, \forall j$, forms market clearing prices if the program $DP(\underline{P})$ has an optimal solution with $\beta_j = 0, \forall j$.*

PROOF. Consider an optimal primal and an optimal dual with $\beta_j = 0, \forall j$. The optimal primal solution satisfies conditions (9) and (10) which are same as (2) and (3). Complementary slackness conditions (12) with $\beta_j = 0, \forall j$ together with the dual feasibility conditions (11) give:

$$\begin{aligned} \forall i,j : x_{ij} > 0 \quad &\Rightarrow \quad \alpha_i p_j + \beta_j = v_{ij} \\ &\Rightarrow \quad \alpha_i = v_{ij}/p_j \\ &\Rightarrow \quad \forall k : (v_{ij}/p_j)p_k + \beta_k \geq v_{ik} \\ &\Rightarrow \quad \forall k : v_{ij}/p_j \geq v_{ik}/p_k \end{aligned}$$

□

We now show the converse of this lemma.

LEMMA 3.2 *If a price vector $\underline{P}$ such that $p_j > 0, \forall j$, forms market clearing prices then the corresponding dual program $DP(\underline{P})$ has an optimal solution with $\beta_j = 0, \forall j$.*

PROOF. Since the vector $\underline{P}$ forms market clearing prices, there is an allocation $\underline{X}$ that satisfies (2), (3) and (4). Therefore, the vector $\underline{X}$ is a primal feasible solution for $LP(\underline{P})$. Define $\alpha_i = v_{ij}/p_j$ for some j such that $x_{ij} > 0$. Let $\beta_j = 0$, $\forall j$. From (4) it follows that $\alpha_i, 1 \leq i \leq n$, and $\beta_j, 1 \leq j \leq m$, is a dual feasible solution. Also note that these primal and dual feasible solutions satisfy the complementary slackness conditions (12). Therefore, they must be the optimal solutions for $LP(\underline{P})$ and $DP(\underline{P})$ and thus there is a dual optimal solution with $\beta_j = 0, \forall j$. □

4. An Auction Algorithm for Approximate Market Clearing Prices We now present an algorithm for discovering approximate market clearing prices. The algorithm is based on the primal-dual formulation for market clearing as described in the previous section. The variables β_j are set to zero throughout the algorithm. The variables x_{ij} are initialized to zero and are modified as the algorithm progresses. The prices p_j are initialized to 1 and are slowly and monotonically increased as the algorithm makes progress. Note that if $\underline{P}$ forms market clearing prices then $\alpha\underline{P}$ also forms market clearing prices for any scalar $\alpha > 0$. Since we assume that there exist clearing prices such that $p_j > 0$ for all j, there also exist market clearing prices such that $p_j \geq 1$ for all j.

At every step in the algorithm, an "over-demanded" good is picked and reassigned to another buyer after raising its price by a fixed multiplicative factor $(1+\epsilon)$, where $\epsilon > 0$ is a small quantity suitably chosen at the beginning of the algorithm. This algorithm has an auction interpretation, where traders outbid each other to acquire goods of their choice by submitting a bid that is a factor $(1+\epsilon)$ of the current winning bid. When the algorithm terminates, approximate market clearing conditions are satisfied. The algorithm is very similar to the auction approach of Bertsekas [3] and Demange et al. [7] for solving the

assignment problem. The key difference however, is that, in the assignment problem, price corresponds to the dual variables (β's), while in our algorithm β's are fixed at zero and price corresponds to variables that appear in primal as well as dual program.

The dual variables α_i are chosen such that the dual feasibility condition (11) (with $\beta_j = 0$ for all j) is satisfied. During the course of the algorithm, the variables x_{ij} are successively modified by a bidding process such that the approximate market equilibrium conditions (8) and the following relaxed primal and relaxed complementary slackness conditions are always satisfied:

$$\forall j : \sum_{i=1}^{n} x_{ij} \leq a_j \tag{13}$$

$$\forall i, j : x_{ij} > 0 \Rightarrow v_{ij} \leq \alpha_i p_j \leq (1+\epsilon) v_{ij} \tag{14}$$

The bidding process raises the prices. As these prices increase, the inequalities (13) become tighter and closer to (5). Eventually, when the algorithm terminates (5) is satisfied. This leads to approximate market clearing. We now present the algorithm in detail.

Let the price of good j be p_j. At any stage in the auction algorithm, each good j is sold at two prices: $p_j/(1+\epsilon)$ and p_j. Let y_{ij} be the amount of good j sold to trader i at price $p_j/(1+\epsilon)$ and h_{ij} be the amount sold to trader i at price p_j. Now, $x_{ij} = y_{ij} + h_{ij}$.

Define the demand set D_i of trader i as:

$$D_i = \arg\max_j v_{ij}/p_j \tag{15}$$

Define the surplus (r_i) left with trader i as:

$$r_i = \sum_{j=1}^{m} a_{ij} p_j - \sum_{j=1}^{m} y_{ij} \frac{p_j}{1+\epsilon} - \sum_{j=1}^{m} h_{ij} p_j \tag{16}$$

and the total surplus as $r = \sum_{i=1}^{n} r_i$. Let $a_{min} = \min_{ij}\{a_{ij} : a_{ij} > 0\}$, $a_{max} = \max_j a_j$ and $a = \sum_{j=1}^{m} a_j$. Note that $a_{min} > 0$ by definition. Further, a good j is defined to be *unassigned* if $\sum_{i=1}^{n} x_{ij} < a_j$ and *assigned* otherwise. Define a good j to be available at price p if its current price p_j is equal to p and $\sum_{i=1}^{n} h_{ij} < a_j$.

A formal description of the algorithm is given in Figure 1. At the beginning of the algorithm `main`, the prices (p_j) are initialized to 1 and the variables y_{ij} and h_{ij} are initialized to zero. Inside the `repeat` loop a trader (say i) with positive surplus acquires goods in its demand set. If a good (say j) in the demand set of trader i is still unassigned, it is acquired at unit price using procedure `assign`(i, j). If the good j has already been completely assigned and is available at its current price p_j, it is acquired by the procedure `outbid`(i, j, k) from another trader k, who has been assigned the good at a lower price ($p_j/(1+\epsilon)$). If good j is not available at its current price p_j, its price is increased by a factor $(1+\epsilon)$ using procedure `raise_price`(j) and hence the good is made available at its new price. This process continues until either the surplus of the traders becomes sufficiently small or until all the goods are assigned.

Note that the algorithm described above is highly distributed and asynchronous. The algorithm does not specify the order in which the procedures `outbid`(i, j, k), `assign`(i, j) and `raise_price`(j) are called. These may be called for any value of i, j, k that satisfy the corresponding entry conditions.

LEMMA 4.1 *During the progress of the auction algorithm, conditions (8), (11) (with $\beta_j = 0$ for all j), (13) and (14) are always satisfied.*

PROOF. Condition (13) is satisfied after the initialization step. Note that procedure `outbid` does not change the values $\sum_{i=1}^{n} x_{ij}$ for all j. So, if condition (13) is satisfied before the entry to the procedure, it is also satisfied after its exit. In the procedure `assign` the variable t is chosen such that (13) remains satisfied. Procedure `raise_price` does not change the value of x_{ij}. Therefore (13) remains satisfied throughout the algorithm.

For condition (8) it is sufficient to show that $r_i \geq 0$ throughout the auction. This is true after the initialization step. In procedures `outbid` and `assign`, the variable t is chosen in such a way that $r_i \geq 0$.

```
algorithm main
define x_ij ≡ y_ij + h_ij
define D_i ≡ arg max_j v_ij/p_j
∀i, j : y_ij = h_ij = 0;  p_j = 1;
∀i, r_i = Σ_{j=1}^{m} a_ij p_j;
∀i, α_i = max_j v_ij/p_j;
repeat
        pick i s.t.  r_i > 0
        α_i = max_j v_ij/p_j
        pick j ∈ D_i
        if Σ_{k=1}^{n} x_kj < a_j then assign(i, j)
        else if ∃k s.t.  y_kj > 0 then outbid(i, j, k)
        else raise_price(j)
until Σ_{i=1}^{n} r_i < ε/(1+ε) a_min or ∀j : Σ_{i=1}^{n} x_ij = a_j
end algorithm main

procedure outbid(i, j, k)
        t = min(y_kj, r_i/p_j)
        h_ij = h_ij + t
        y_kj = y_kj − t
        r_i = r_i − t p_j
        r_k = r_k + t p_j/(1+ε)
end procedure

procedure assign(i, j)
        t = min(a_j − Σ_{k=1}^{n} x_kj, r_i/p_j)
        h_ij = h_ij + t
        r_i = r_i − t p_j
end procedure

procedure raise_price(j)
        ∀k : y_kj = h_kj
        ∀k : h_kj = 0
        ∀i : r_i = r_i + ε a_ij p_j
        p_j = (1 + ε) p_j
end procedure
```

Figure 1: An auction algorithm for discovering market clearing prices

Procedure `raise_price` does not change the value of $\sum_{j=1}^{m} a_{ij}p_j - r_i = \sum_{j=1}^{m} y_{ij}p_j/(1+\epsilon) + \sum_{j=1}^{m} h_{ij}p_j$ for any i. As a result r_i can only increase in this procedure. Therefore (8) remains satisfied throughout the algorithm.

The variables α_i's are set during initialization step and the update step in such a way that (11) is satisfied. Procedure `raise_price`(j) only increases p_j. Therefore (11) remains satisfied throughout the algorithm.

Condition (14) is satisfied after the initialization step. Since (11) is satisfied throughout the algorithm $v_{ij} \leq \alpha_i p_j$. We just need to show that:

$$\forall i,j : x_{ij} > 0 \Rightarrow \alpha_i p_j \leq (1+\epsilon)v_{ij}. \tag{17}$$

For all j, p_j can only increase as the algorithm progresses. Thus, for all i, α_i can only decrease. Therefore, if (17) is satisfied before update of α_i, it will remain satisfied after the update as well. When x_{ij} is increased in procedure `outbid` or `assign`, $j \in D_i$ i.e. $\alpha_i = v_{ij}/p_j$, which satisfies (17). It remains to be seen that (17) continues to be satisfied after `raise_price` is called.

Note that when `assign` or `outbid` is called, $j \in D_i$ i.e. $v_{ij} = \alpha_i p_j$. A call to `raise_price`(j) increases p_j by a factor $(1+\epsilon)$. So, if `raise_price`(j) is called after a call to `assign`(i, j) or `outbid`(i, j, k), condition (17) remains satisfied. So, it is sufficient to show that between two successive calls to `raise_price`(j), `outbid`($*$, j, k) is called for all k such that $x_{kj} > 0$. To see this, observe that `raise_price`(j) sets $y_{kj} = x_{kj}, \forall k$. But, `raise_price`($j$) is called only when $\forall k, y_{kj} = 0$. `outbid`($*$, j, k) is the only place where the value of y_{kj} is reduced. Therefore, `outbid`(i, j, k) must be called for every k s.t. $y_{kj} > 0$. This completes the proof. □

This algorithm also satisfies the following invariant:

$$\forall j : p_j > 1 \Rightarrow \sum_{i=1}^{n} x_{ij} = a_j \tag{18}$$

To see this, note that $p_j = 1, \forall j$, at initialization. So (18) is satisfied after the initialization. Procedure `outbid`(i, j, k) does not change the value of $\sum_{i=1}^{n} x_{ij}$ and hence does not affect (18). Procedure `assign`(i, j) is called only when $p_j = 1$ and `raise_price`(j) is called only if $\sum_{i=1}^{n} x_{ij} = a_j$. Hence (18) remains satisfied throughout the execution of the algorithm.

LEMMA 4.2 *When algorithm* `main` *terminates, condition (5) is satisfied.*

PROOF. The second inequality of (5) follows from invariant (13). We now show that the first inequality of (5) is satisfied when the algorithm terminates.

There are two conditions for the algorithm to terminate. When all the goods get assigned ($\forall j : \sum_{i=1}^{n} x_{ij} = a_j$) then condition (5) is trivially satisfied. In the other case we have, $\sum_{i=1}^{n} r_i \leq \frac{\epsilon}{(1+\epsilon)} a_{min}$. From (16) we have:

$$r_i = \sum_{j=1}^{m}(a_{ij}p_j - x_{ij}p_j) + \frac{\epsilon}{(1+\epsilon)}\sum_{j=1}^{m} y_{ij}p_j \tag{19}$$

$$\begin{aligned} \Rightarrow \sum_{i=1}^{n} r_i &= \sum_{j=1}^{m}\sum_{i=1}^{n}(a_{ij} - x_{ij})p_j + \frac{\epsilon}{(1+\epsilon)}\sum_{j=1}^{m}\sum_{i=1}^{n} y_{ij}p_j \\ &= \sum_{j=1}^{m}\left(a_j - \sum_{i=1}^{n} x_{ij}\right)p_j + \frac{\epsilon}{(1+\epsilon)}\sum_{j=1}^{m}\sum_{i=1}^{n} y_{ij}p_j \end{aligned} \tag{20}$$

Since $\sum_{i=1}^{n} r_i \leq \frac{\epsilon}{1+\epsilon} a_{min}$, we have

$$\sum_{j=1}^{m}\left(a_j - \sum_{i=1}^{n} x_{ij}\right)p_j + \frac{\epsilon}{(1+\epsilon)}\sum_{j=1}^{m}\sum_{i=1}^{n} y_{ij}p_j \leq \frac{\epsilon}{(1+\epsilon)} a_{min}$$

Since $y_{ij} \geq 0$ and $p_j \geq 0$ for all i, j, the above inequality can be reduced to the following

$$\sum_{j=1}^{m}\left(a_j - \sum_{i=1}^{n} x_{ij}\right)p_j \leq \frac{\epsilon}{(1+\epsilon)} a_{min}$$

From (18) it follows that if $\sum_{i=1}^{n} x_{ij} < a_j$ then $p_j = 1$. Therefore,

$$\sum_{j=1}^{m} \left(a_j - \sum_{i=1}^{n} x_{ij} \right) \leq \frac{\epsilon}{(1+\epsilon)} a_{min}$$

From (13) it follows that, each term of the summation in the above inequality is non-negative. Therefore each term of the summation must be bounded by the right hand side i.e.

$$a_j - \sum_{i=1}^{n} x_{ij} \leq \frac{\epsilon}{(1+\epsilon)} a_{min}, \ \forall j$$

$$\Rightarrow \frac{a_j}{(1+\epsilon)} \leq \sum_{i=1}^{n} x_{ij}, \ \forall j$$

□

Lemma 4.1 and 4.2 imply than the algorithm `main` achieves approximate market clearing on termination. To see this, observe that (11) with $\beta_j = 0$ for all j and (14) imply (7).

4.1 Analysis We now analyze the time complexity of the algorithm. Let p_{max} be an upper bound on the prices discovered by this algorithm. Every time `raise_price`(j) is called, p_j is increased by a factor $(1+\epsilon)$. Therefore, the number of times `raise_price` can be called is bounded by $m \log_{(1+\epsilon)} p_{max} = O(\frac{m}{\epsilon} \log p_{max})$.

We partition the execution of the algorithm into rounds. The first round corresponds to the first k_1 sequence of iterations of the `repeat` loop in algorithm `main`, such that each trader i with $r_i > 0$ has r_i reduced to zero at least once (after calling `outbid`(i, j, k) or `assign`(i, j)). Round l corresponds to sequence of iterations from $k_{l-1} + 1$ to k_l, where k_l is the minimum integer greater than k_{l-1} ensuring that each trader i with $r_i > 0$ has r_i reduced to zero at least once in round l. Note that r_i may possibly become positive again in subsequent iteration of the same round.

LEMMA 4.3 *In every round, either there is a price rise, or the total surplus (r) reduces by a factor* $1/(1+\epsilon)$.

PROOF. Assume that `raise_price` is never called in the round. It is sufficient to show that the total surplus reduces by a factor $1/(1+\epsilon)$. Let r_i represent the surplus of trader i at the beginning of the round and let $r = \sum_{i=1}^{n} r_i$ represent the the total surplus at the beginning of the round. We now put a lower bound on the surplus reduction in a round.

Note that a call to `assign` decreases the value of the total surplus by tp_j (as calculated in procedure `assign`). Also note that, a call to `outbid`(i, j, k) decreases the surplus of trader i by tp_j and increases the surplus of trader k by $tp_j/(1+\epsilon)$. Therefore, it decreases the value of the total surplus by $tp_j\epsilon/(1+\epsilon)$. In every round, for each trader i, `outbid` is repeatedly called (possibly interleaved with calls to `outbid` and `assign` for other traders) until the surplus of trader i goes to zero. Biddings done earlier by other traders can only increase the surplus of trader i. Therefore, in calls made to `outbid` and `assign` by trader i the total surplus is guaranteed to reduce by at least $r_i\epsilon/(1+\epsilon)$. Adding this for every trader, we get that the total surplus reduces by at least $r\epsilon/(1+\epsilon)$ in every round where `raise_price` is not called. In other words the surplus r' after the round is bounded as: $r' \leq r/(1+\epsilon)$. □

LEMMA 4.4 *Algorithm* `main` *terminates in* $O(\frac{1}{\epsilon^2} m \log(\frac{p_{max} a}{\epsilon a_{min}}) \log p_{max})$ *rounds, where* $a = \sum_{j=1}^{m} a_j$.

PROOF. Note that r can be written as:

$$r = \sum_{j=1}^{m} \sum_{i=1}^{n} (a_{ij} - x_{ij}) p_j + \frac{\epsilon}{1+\epsilon} \sum_{j=1}^{m} \sum_{i=1}^{n} y_{ij} p_j$$

It is easy to see that, `raise_price` can increase the value of r. The maximum possible value of surplus r is ap_{max} where $a = \sum_{j=1}^{m} a_j$. The algorithm is guaranteed to terminate if the total surplus becomes less than $\frac{\epsilon}{(1+\epsilon)} a_{min}$. Therefore, the maximum number of rounds between two successive calls to `raise_price` is

bounded by $O(\frac{1}{\epsilon}\log(\frac{ap_{max}}{\epsilon a_{min}}))$. The number of times `raise_price` is called is bounded by $O(\frac{1}{\epsilon}m\log p_{max})$. This gives the required bound. □

To efficiently implement this algorithm, each trader is considered once in every round. When trader i is chosen for processing, it makes $r_i = 0$ by repeatedly calling procedure `outbid`. When `outbid`(i, j, k) is called, either y_{kj} becomes zero or r_i becomes zero. The first event is charged to a call to `raise_price`(j) while the second event is charged to the trader i. For every call to `raise_price`(j), there can be at most n events of the first type. In every round, there can be at most n events of the second type. This gives the following time complexity of the algorithm.

THEOREM 4.1 *Algorithm* `main` *computes a solution satisfying (5), (7) and (8) in* $O(\frac{1}{\epsilon^2}nm\log(\frac{p_{max}a}{\epsilon a_{min}})\log(p_{max}))$ *time.*

We next present a modification to the above basic algorithm that gives better convergence (for $\frac{1}{\epsilon} \geq (n+m)$) and a stronger market clearing condition.

5. An Improved Auction Algorithm With two minor variations on the bidding order, the upper bound on the running time can be significantly improved. These variations are: (a) A trader i with positive surplus raises its assignment of good j to the higher price (p_j) before outbidding another trader on the good; (b) A trader who is being outbid on a good which is still in its demand set, bids back immediately on the good and raises its assignment to the higher price. These changes, along with bidding in rounds as discussed earlier, are incorporated in the algorithm `main2` as shown in Figure 2.

Algorithm `main2` carries out bidding until all the goods have been assigned. One iteration of the `repeat` loop completes a round. It consists of a sequence of iterations of the inner `while` loop such that every trader i exhausts its surplus (r_i becomes 0) exactly once. In the description presented in Figure 2, traders are picked one by one and exhaust their surplus. Once a trader is picked (say i), it selects a good in its demand set (say j). If the good j is not completely assigned then the available amounts of good j is assigned to trader i. If good j is completely assigned and if good j is assigned to some trader at the lower price $p_j/(1+\epsilon)$, then there are two cases. If trader i already has good j assigned at the lower price , i.e. $y_{ij} > 0$ then the surplus is spent in acquiring the same good at a higher price using `outbid2`(i, j, i). Otherwise, trader i acquires the good from another trader k, who was allocated the good at price $p_j/(1+\epsilon)$. If all of good j has been assigned at the higher price, then `raise_price` is called. In this case, the demand sets are recomputed and the iteration resumes. This continues until the surplus of the trader has been exhausted ($r_i = 0$). After this, the next trader is picked and the iterations of the `while` loop continue with the next trader.

For simplicity of exposition, `main2` in Figure 2 imposes round-robin ordering for the traders to carry out bidding. It may be noted that such an ordering is not required. In a round, an iteration of the inner `while` loop may be carried out for any (i, j), as long as $j \in D_i$ and r_i has not become zero earlier in the current round.

Before moving further into the analysis on the algorithm, we introduce some notations. Consider a directed bipartite graph $G = (T, S, E)$ where T is the set of traders, S is the set of goods and E is a set of directed edges between S and T. Define D to be the set of demand edges, X the set of assignment edges, Y a subset of the set of assignment edges and B a subset of Y as follows.

$$\begin{aligned}
(i,j) \in D \quad &\text{iff} \quad j \in D_i \\
(j,i) \in X \quad &\text{iff} \quad x_{ij} > 0 \\
(j,i) \in Y \quad &\text{iff} \quad y_{ij} > 0 \\
(j,i) \in B \quad &\text{iff} \quad y_{ij} > 0 \text{ and } j \notin D_i
\end{aligned}$$

Note that when procedure `outbid2`(i, j, k) is called either r_i goes to zero or an edge from Y is removed (either y_{ij} goes to zero or y_{kj} goes to zero). Define a call to `outbid2` as complete when an edge in Y is removed and incomplete if r_i goes to zero.

LEMMA 5.1 *The number of complete calls to* `outbid2` *is bounded by* $O(\frac{1}{\epsilon}nm\log p_{max})$.

```
algorithm main2
define x_ij ≡ h_ij + y_ij
define D_i ≡ arg max_j v_ij/p_j
∀i, j : y_ij = h_ij = 0;  p_j = 1;
r_i = Σ_{j=1}^{m} a_ij p_j;
α_i = max_j v_ij/p_j;
∀i : compute demand sets D_i,  α_i = max_j v_ij/p_j
repeat
   for i = 1 to n
      while r_i > 0 do
         pick j ∈ D_i
         if Σ_{k=1}^{n} x_kj < a_j then assign(i, j)
         else if y_ij > 0 then outbid2(i, j, i)
         else if ∃k s.t. y_kj > 0 then outbid2(i, j, k)
         else raise_price(j); ∀i, recompute D_i and α_i
         if ∀j : Σ_{k=1}^{n} x_kj = a_j then goto done
      end while
   end for
until ∀j : Σ_{k=1}^{n} x_kj = a_j
done:
end algorithm main2

procedure outbid2(i, j, k)
   if j ∉ D_k and i ≠ k then
      t = min(y_kj, r_i/p_j)
      h_ij = h_ij + t
      y_kj = y_kj − t
      r_i = r_i − t p_j
      r_k = r_k + t p_j/(1 + ε)
   else
      t = min(ε/(1+ε) y_kj, r_i/p_j)
      h_kj = h_kj + t/ε
      y_kj = y_kj − t(1 + ε)/ε
      h_ij = h_ij + t
      r_i = r_i − t p_j
   endif
end procedure
```

Figure 2: An improved auction algorithm

PROOF. Initially, there is no edge in Y. Edges in Y are added (y_{kj} become non-zero) only through a call to `raise_price`. Each call to `raise_price` can add at most n edges in Y. Since the total number of price raises are bounded by $O(\frac{1}{\epsilon} m \log p_{max})$, the total number of edges added to Y is bounded by $O(\frac{1}{\epsilon} nm \log p_{max})$. Each complete call to `outbid2` removes one edge in Y. Hence the result. □

To bound the total running time of the algorithm, we need to bound the number of incomplete calls to `outbid2`. Consider an incomplete call to procedure `outbid2` by trader i. If r_k was zero before a call to procedure `outbid2`(i, j, k), it may become positive after the call. This may lead to another incomplete call to the procedure `outbid2`(k, j', l) by trader k transferring the surplus to another trader l with zero surplus. This surplus could eventually cycle back to the trader i, thus repeating the whole process again. We show that this is not the case.

It can be seen from the definition of procedure `outbid2` that, surplus may be transferred from trader i to a trader j using a sequence of calls to `outbid2` only through a directed path in the graph $G = (T, S, D \cup B)$. We now prove an important result which establishes that the surplus from a trader i cannot cycle back to itself without a price rise.

LEMMA 5.2 *The graph $G = (T, S, D \cup B)$ is acyclic.*

PROOF. Consider the graph G at time t. Assume for contradiction that G has a cycle of the form $(u_1, v_1, u_2, v_2 \ldots, u_k, v_k, u_1)$ where $u_i \in T, v_i \in S$. Let t_i be the time instant when the price of good v_i was raised to its current level.

The fact that $(v_1, u_2) \in B$ implies that $y_{u_2 v_1} > 0$ and $v_1 \notin D_{u_2}$. Since v_1 is currently not in the demand set of u_2, it must have been assigned to u_2 at time $t' < t_1$. When v_1 was being assigned to u_2, the price of v_2 must have been at its current level otherwise v_2 would have been in the demand set of u_2 instead of v_1. Therefore, the price raise of v_2 must have happened prior to that of v_1 i.e. $t_2 < t_1$. Continuing this argument we get $t_1 < t_k < t_{k-1} < \ldots < t_2 < t_1$ which is a contradiction to our assumption that there was a cycle in G. Therefore there is no cycle in G. □

LEMMA 5.3 *After d rounds of bidding either there is a price rise, or the surplus of at least d traders becomes zero. Moreover, the surplus of these traders cannot rise later until there is a price rise.*

This lemma is immediate from the following stronger statement. Assign a unique rank between 1 and n to each trader using a topological sort of the directed acyclic graph G, such that for each $i, j \in T$ if there is a path from i to j in G, i is assigned a rank less than j. Note that, in G no trader has a path to the trader of rank 1.

LEMMA 5.4 *If there is no price rise in any consecutive d rounds of bidding, then all the traders with rank less than or equal to d have zero surplus. Moreover, the surplus of these traders cannot increase later until there is a price rise.*

PROOF. Assume that there is no price rise in d consecutive rounds. We prove this result by induction on the number of rounds d. We first establish the base case.

Let k be the trader of rank 1. There is no path in G from any other trader to k. r_k becomes zero in the first round after a call to `assign` or an incomplete call to `outbid2`. r_k can become positive again only through a call to `outbid2`(i, j, k), such that $y_{kj} > 0$ and $j \notin D_k$ i.e. $(j, k) \in B$. Such a call will be made only if $j \in D_i$ i.e. $(i, j) \in D$. This is not possible since there is no path in G from any other trader to trader k.

Note that as the bidding progresses without a price rise, D remains unchanged. Moreover, the sets Y and B shrink in size. Therefore, the ranks defined at the beginning of the first round remain consistent with the modified G until there is a price rise.

Now consider round d. In every round every trader exhausts its surplus once. Therefore, in the dth round the trader of rank d will also exhaust its surplus. This traders can acquire surplus again only through the traders with rank less than d (from construction of G). However, by induction hypothesis, all the traders of rank less than d will have zero surplus after round $d - 1$. Therefore, the trader of rank

d cannot acquire a surplus in round d or later and thus will have zero surplus at the end of round d and thereafter. □

LEMMA 5.5 *The algorithm* `main2` *terminates in* $O(\frac{1}{\epsilon}nm\log p_{max})$ *rounds.*

PROOF. From Lemma 5.3 it follows in that, after n rounds of bidding, either there is a price rise, or the total surplus r goes down to zero. Consider the case when the total surplus goes to zero ($r_i = 0, \forall i$). Summing (16) (or equivalently (19)) over i and using the fact that $\sum_{i=1}^{n} x_{ij} \leq \sum_{i=1}^{n} a_{ij}$ for all j, and $y_{ij} \geq 0$ for all i and j, we get $y_{ij} = 0$ for all i, j and $\sum_{i=1}^{n} x_{ij} = a_j$ for all j. In this case, the algorithm terminates. There can be at most $O(\frac{1}{\epsilon}m\log p_{max})$ price rises. Hence the result. □

LEMMA 5.6 *The time complexity of algorithm* `main2` *is* $O(\frac{1}{\epsilon}(nm^2 + mn^2)\log(p_{max}))$.

PROOF. A call to `assign` either raises the price or sets $r_i = 0$. Therefore it may be called at most n times in every round. So, total number of calls to `assign` is bounded by $O(\frac{1}{\epsilon}mn^2\log p_{max})$.

From Lemma 5.1, there can be at most $O(\frac{1}{\epsilon}mn\log p_{max})$ complete calls to `outbid2`. There can be at most $O(n)$ incomplete calls (one for each trader) to `outbid2` in every round. Therefore, the total number of calls to `outbid2` is bounded by $O(\frac{1}{\epsilon}(mn + mn^2)\log p_{max})$.

Demand set computation take $O(nm)$ time and there can be at most $O(\frac{1}{\epsilon}m\log p_{max})$ such computations. Therefore the total time in demand set computations is bounded by $O(\frac{1}{\epsilon}nm^2\log p_{max})$. This gives the required bound. □

LEMMA 5.7 *When the algorithm terminates conditions (2), (5), (7) and (8) are satisfied.*

PROOF. Condition (8) is satisfied because $r_i \geq 0$. Algorithm `main2`, like the predecessor `main` satisfies $\alpha_i p_j \geq v_{ij}, \forall i, j$ throughout its execution. It also satisfies the approximate complementary slackness conditions (14). This gives (7). Algorithm `main2` terminates when all the goods are assigned Therefore (2) is satisfied which also implies (5). □

Using the above two lemmas, Lemma 5.6 and Lemma 5.7, we get the following result:

LEMMA 5.8 *Algorithm* `main2` *computes a solution satisfying (2), (5), (7) and (8) in* $O(\frac{1}{\epsilon}(nm^2 + mn^2)\log(p_{max}))$ *time.*

Note that the algorithms `main` and `main2` do not ensure that all the traders spend most of their money (6). However, inequality (5) ensures that all the money ($\sum_{j=1}^{m}\sum_{i=1}^{n} a_{ij}p_j$) in the system is spent to within $(1+\epsilon)$ of its value. To ensure (6) we add a second phase of the algorithm.

6. Phase 2. Achieving $(1+\epsilon)$-approximate Market Equilibrium The solution found by `main` and `main2` can be modified to satisfy (6) by adding a second phase (called `phase2`) to the algorithm. In this phase the same algorithms may be continued with two minor variations: (a) only traders i with $r_i > \epsilon\sum_{j=1}^{m} a_{ij}p_j$ carry out bidding, and (b) during the bidding traders do not exhaust the surplus all the way to zero. Trader i stops bidding (and acquiring goods) as soon as r_i reaches $\epsilon\sum_{j=1}^{m} a_{ij}p_j$. The phase ends when $r_i \leq \epsilon\sum_{j=1}^{m} a_{ij}p_j$ for all i, thereby satisfying (6). Note that in this phase, if the following inequality (21) is satisfied at any stage, then it remains satisfied for the rest of the phase.

$$r_i \geq \epsilon\sum_{j=1}^{m} a_{ij}p_j \tag{21}$$

Since `phase2` is very similar to the corresponding algorithms `main` and `main2`, we omit its formal description and leave the details to the reader. Note the key difference in `phase2` is that the residual surplus is not reduced to zero but to within a factor of $(1+\epsilon)$ of the endowment which is allowed to remain unspent at the end. This affects the error condition when phase2 follows `main` such that the error bound in (6) now becomes $(1+2\epsilon)$ since the residual r_i is not reduced to zero but to within $(1+\epsilon)$ factor of $\epsilon\sum_{j=1}^{m} a_{ij}p_j$. Executing `main` followed by `phase2` with an error bound $\epsilon' = \epsilon/2$ gives the desired error bound without affecting the complexity. The error bound when `phase2` follows `main2` remains unchanged.

We now prove some properties of `phase2`.

LEMMA 6.1 *In* `phase2`*, there is a good whose price doesn't rise.*

PROOF. Assume, for contradiction that the price of all the goods has risen in `phase2`. Consider any stage in `phase2` after prices of all the goods has risen. At this stage, (2) is satisfied (since $p_j > 1$ implies $\sum_{i=1}^{n} x_{ij} = a_j$). If the prices of all the goods rise by a factor $(1+\epsilon)$ or more, r_i (for all i) must also increase by at least $\epsilon \sum_{j=1}^{m} a_{ij}p_j$ at some stage. Therefore, (21) is satisfied for all i.

From the definition of r_i we have,

$$\sum_{i=1}^{n} r_i + \frac{1}{1+\epsilon}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}p_j + \sum_{i=1}^{n}\sum_{j=1}^{m} h_{ij}p_j = \sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}p_j$$

Since $x_{ij} = h_{ij} + y_{ij}$ we have

$$\sum_{i=1}^{n} r_i + \sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij}p_j - \frac{\epsilon}{1+\epsilon}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}p_j = \sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}p_j$$

Using (2), the above equation reduces to

$$\sum_{i=1}^{n} r_i = \frac{\epsilon}{1+\epsilon}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}p_j$$

Since (21) is satisfied for all i we have

$$\sum_{i=1}^{n} r_i \geq \epsilon \sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}p_j$$

The above two inequalities imply

$$\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}p_j \geq (1+\epsilon)\sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}p_j$$

However, $\sum_{i=1}^{n} y_{ij} \leq \sum_{i=1}^{n} a_{ij}$ for all j, which implies

$$\sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}p_j \geq \sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}p_j \geq (1+\epsilon)\sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}p_j$$

which is a contradiction. Therefore, there is a good whose price doesn't rise in `phase2`. □

7. Bounding p_{max} In this section we bound the value of p_{max} that arises in the complexity of each of the above algorithms. We define $p_{max}(\mathtt{A})$ be the maximum price of any good during the execution of algorithm `A` (which can be `main`, `main2` or `phase2`). We first bound $p_{max}(\mathtt{A})$ for the $(1+\epsilon)$-approximate market equilibrium model presented in Section 2. We subsequently show that better bounds can be achieved by relaxing the definition of approximate market equilibrium.

7.1 Case 1: $v_{ij} > 0$ for all i, j Let $v_{max} = \max_{i,j} v_{ij}$ and $v_{min} = \min_{i,j} v_{ij}$. Let $p_{min}(\mathtt{A})$ and $p_{max}(\mathtt{A})$ respectively be the minimum and maximum price of goods at the end of the algorithm `A`. Now we have the following bound:

LEMMA 7.1 *Any solution satisfying (7) also satisfies* $p_j \leq (1+\epsilon)(v_{max}/v_{min})p_{min}(\mathtt{A})$ *for all* j.

PROOF. If good j is not allocated to any trader ($x_{ij} = 0$ for all i) then p_j is equal to 1 thereby satisfying the above. Otherwise, let good j be possessed by trader k (i.e. $x_{kj} > 0$). Let l be the good with the minimum price. From (7) it follows that $v_{kj}/p_j \geq v_{kl}/((1+\epsilon)p_l)$. The proof follows immediately from this. □

Note that $p_{min}(\mathtt{main}) = p_{min}(\mathtt{main2}) = 1$. Lemma 6.1 implies that $p_{min}(\mathtt{phase2}) \leq p_{max}(\mathtt{main2})$ (or $p_{max}(\mathtt{main})$). Thus $p_{max}(\mathtt{phase2}) \leq ((1+\epsilon)v_{max}/v_{min})^2$. This leads to the following theorem.

THEOREM 7.1 *Algorithm* `main2` *followed by* `phase2` *computes a* $(1+\epsilon)$*-approximate market equilibrium in* $O(\frac{1}{\epsilon}(nm^2 + mn^2)\log(v_{max}/v_{min}))$ *time when* $v_{ij} > 0$ *for all* i *and* j.

7.2 Case 2: $v_{ij} = 0$ for some i, j. In this case some of the equilibrium prices could be zero. Since prices are invariant to scaling and $p_{min}(\mathtt{A}) \geq 1$ for all auction algorithms $\mathtt{A}$, $p_{max}(\mathtt{A})$ could potentially go to infinity. To handle this, we first discover zero priced goods using a technique similar to [13] and remove them from the market. Now all the prices will be positive in the modified market. The auction algorithms are then executed on the modified market. The running time of the algorithms is bounded by the ratio $p_{max}(\mathtt{A})/p_{min}(\mathtt{A})$ in the modified market.

Define an endowment-valuation (E-V) graph as follows: The graph is a directed bipartite graph $G(T, S, E)$, with E specified as follows: $E = \{(t_i, s_j)|v_{ij} > 0\} \cup \{(s_j, t_i)|a_{ij} > 0\}$. The following result characterizes equilibrium prices:

LEMMA 7.2 *There exist a market equilibrium such that all goods with positive prices belong to one strongly connected component, C, of the E-V graph.*

PROOF. Consider the strongly connected components of the E-V graph. When considered independently in isolation, each of these strongly connected components will have a market equilibrium with all goods at non-zero prices (see [13]). Note that there is a partial ordering induced by the edges of the E-V graph on these components. Set the price of all components except one (maximal) component (say C) that has no outgoing edge, to zero. The market equilibrium problem can be solved independently on C because (a) the other components with zero prices do not provide any money to traders in C; (b) all traders in C have zero value on goods outside C and therefore will not benefit by getting any good outside C and (c) all traders outside C will have zero money to spend and will therefore not get any good form C. □

Now the algorithm is run on the component C while ignoring all the goods and traders not in C. We now bound the ratio of maximum to minimum. Let $p_j^k(\mathtt{A})$ represent the price of good j at the end of round k of algorithm $\mathtt{A}$. Let $p_{max}^k(\mathtt{A}) = \max_j p_j^k(\mathtt{A})$ and $p_{min}^k(\mathtt{A}) = \min_j p_j^k(\mathtt{A})$.

LEMMA 7.3 $p_{max}^k(\mathtt{A})/p_j^{k+n}(\mathtt{A}) \leq \left[\frac{1+\epsilon}{1-\epsilon}\frac{m v_{max} a_{max}}{v_{min} a_{min}}\right]^n$ *for all j.*

PROOF. Let good j_1 be priced at $p_{max}^k(\mathtt{A})$ at the end of round k. Since the E-V graph is strongly connected, there is a path from j_1 to j. Let the goods and traders in this path respectively be $j_1, j_2, j_3, \ldots, j_l, j$ and $t_1, t_2, \ldots t_l$. At the end of round k the money with trader t_1 is at least $a_{min} p_{max}^k(\mathtt{A})$. At some stage in round $k+1$ this money will be spent within a factor of $(1-\epsilon)$ by the trader. So there exists a good j_2' on which at least $\frac{1}{m}(1-\epsilon)p_{max}^k(\mathtt{A})a_{min}$ money was spent by trader t_1 in round $k+1$. Therefore the price of this good will be at least $\frac{1}{m}(1-\epsilon)p_{max}^k(\mathtt{A})\frac{a_{min}}{a_{max}}$ at the time when t_1 acquired it. Since the auction algorithms satisfy (14) we will also have

$$\begin{aligned}
\frac{v_{t_1 j_2'}}{p_{j_2'}} &\geq \frac{v_{t_1 j_2}}{(1+\epsilon)p_{j_2}} \\
\Rightarrow p_{j_2} &\geq \frac{1}{(1+\epsilon)}\frac{v_{t_1 j_2}}{v_{t_1 j_2'}} p_{j_2'} \\
&\geq \frac{1}{m}\left(\frac{1-\epsilon}{1+\epsilon}\right)\frac{v_{min} a_{min}}{v_{max} a_{max}} p_{max}^k(\mathtt{A})
\end{aligned}$$

Since the price never decreases, we get

$$p_{j_2}^{k+1}(\mathtt{A}) \geq \frac{1}{m}\left(\frac{1-\epsilon}{1+\epsilon}\right)\frac{v_{min} a_{min}}{v_{max} a_{max}} p_{max}^k(\mathtt{A})$$

By a similar argument

$$p_{j_3}^{k+2}(\mathtt{A}) \geq \frac{1}{m}\left(\frac{1-\epsilon}{1+\epsilon}\right)\frac{v_{min} a_{min}}{v_{max} a_{max}} p_{j_2}^{k+1}$$

Continuing in this way along the path to j

$$p_j^{k+l}(\mathtt{A}) \geq \left[\frac{1}{m}\left(\frac{1-\epsilon}{1+\epsilon}\right)\frac{v_{min} a_{min}}{v_{max} a_{max}}\right]^l p_{max}^k(\mathtt{A})$$

This gives the required result (as $l \leq n$). □

In order to bound the running time of the algorithms in this case we will modify each of the algorithms $\mathtt{A} \in \{\mathtt{main},\ \mathtt{main2},\ \mathtt{phase2}\}$ so that traders stop bidding when the price of any good in their demand set exceeds a certain critical value $P_c(\mathtt{A})$. We set $P_c(\mathtt{main}) = P_c(\mathtt{main2}) = \left[m\left(\frac{1+\epsilon}{1-\epsilon}\right)\frac{v_{max}a_{max}}{v_{min}a_{min}}\right]^{n+1}$ and $P_c(\mathtt{phase2}) = [P_c(\mathtt{main2})]^2$. Note that with these changes Lemma 7.3 still remains valid.

Observe that if algorithm $\mathtt{A}$ terminates before $P_c(\mathtt{A})$ is reached, $p_{max}(\mathtt{A})$ is bounded by $P_c(\mathtt{A})$ and hence the running time of the algorithms maybe bounded. However, after $P_c(\mathtt{A})$ is reached Lemma 4.3 and 5.3 are not satisifed and earlier proofs of convergence do not remain valid. We now show that the algorithm $\mathtt{A}$ terminates soon after $P_c(\mathtt{A})$ is reached.

LEMMA 7.4 *Algorithm* $\mathtt{A} \in \{\mathtt{main},\ \mathtt{main2},\ \mathtt{phase2}\}$ *terminates in less than* $n+1$ *rounds after the price of a good reaches* $P_c(\mathtt{A})$.

PROOF. Assume for contradiction that the algorithm doesn't terminate till the specified round. Let $p^k_{max}(\mathtt{A}) = P_c(\mathtt{A})$ for some k. In `main` and `main2` there is a good at price 1 (else all the goods are fully assigned and the algorithm terminates). Let j be such a good. Using Lemma 7.3 we have

$$\begin{aligned} p_j^{k+n}(\mathtt{A}) &\geq \left[\frac{1}{m}\left(\frac{1-\epsilon}{1+\epsilon}\right)\frac{v_{min}a_{min}}{v_{max}a_{max}}\frac{a_{min}}{a_{max}}\right]^n p^k_{max}(\mathtt{A}) \\ &\geq \left[\frac{1}{m}\left(\frac{1-\epsilon}{1+\epsilon}\right)\frac{v_{min}a_{min}}{v_{max}a_{max}}\right]^n P_c(\mathtt{A}) \\ &> 1 \end{aligned}$$

which is a contradiction.

In `phase2`, there is a good whose price doesn't change (see Lemma 6.1). Let this good be j. We must have $p_j^1(\mathtt{phase2}) = p_j^2(\mathtt{phase2}) \ldots p_j^{k+n}(\mathtt{phase2}) \leq P_c(\mathtt{main})$. Again, using Lemma 7.3 we have

$$\begin{aligned} p_j^{k+n}(\mathtt{phase2}) &\geq \left[\frac{1}{m}\left(\frac{1-\epsilon}{1+\epsilon}\right)\frac{v_{min}a_{min}}{v_{max}a_{max}}\right]^n p^k_{max}(\mathtt{phase2}) \\ &\geq \left[\frac{1}{m}\left(\frac{1-\epsilon}{1+\epsilon}\right)\frac{v_{min}a_{min}}{v_{max}a_{max}}\right]^n P_c(\mathtt{phase2}) \\ &> \left[m\left(\frac{1+\epsilon}{1-\epsilon}\right)\frac{v_{max}a_{max}}{v_{min}a_{min}}\right]^{n+1} \\ &= P_c(\mathtt{main}) \end{aligned}$$

which is a contradiction. Thus, the algorithm $\mathtt{A} \in \{\mathtt{main},\ \mathtt{main2},\ \mathtt{phase2}\}$ must terminate in less than $n+1$ rounds after the price of a good reaches $P_c(\mathtt{A})$. □

THEOREM 7.2 *The algorithm* `main2` *followed by* `phase2` *on a strongly connected component of the E-V graph finds a* $(1+\epsilon)$*-approximate market equilibrium in* $O(\frac{1}{\epsilon}(nm^2+mn^2)n\log((mv_{max}a_{max})/(v_{min}a_{min})))$ *time.*

The time complexity of the auction algorithms can be significantly improved by relaxing the definition of approximate market equilibrium. We now show two such relaxations. In the first relaxation (which we call additive error model), we permit an ϵ additive error along with a factor $(1+\epsilon)$ error on maximization problem for each trader LP_i. In this case p_{max} may be bounded by $O(nav_{max}/\epsilon)$. The second relaxation (called aggregate clearing model), is same as in [10], does not require conditions (5) and (6). It only requires that a factor $(1-\epsilon)$ of the total money be spent on goods. In this case p_{max} may be bounded by m/ϵ.

7.3 Additive Error Model We first scale the endowments such that $a_j \geq 1$ for all j. We also scale the utility functions such that the smallest positive utility is at least unity. We perturb v_{ij} by a small amount to ensure that $v_{ij} > 0$ for all i, j.

Now, a new market equilibrium problem is constructed by replacing v_{ij} by δ for all i, j such that $v_{ij} = 0$. δ is chosen to be $\epsilon/(na_{max})$. The auction algorithms are executed on the perturbed market.

Using the results of Section 7.1, p_{max} can be bounded by $\frac{(1+\epsilon)}{\epsilon} n a v_{max}$. Let OPT_i be the optimal value of LP_i for the prices discovered by the algorithm. Since the final solution obtained satisfies (5), (6) and (7)

$$\frac{1-\epsilon}{1+\epsilon} OPT_i \leq \sum_{j=1}^{m} v'_{ij} x_{ij} \leq OPT_i$$

for all i, where v'_{ij} are the perturbed valuations. Substituting v'_{ij} and δ in the above gives the following bound

$$\frac{1-\epsilon}{1+\epsilon} OPT_i - \epsilon \leq \sum_{j=1}^{m} v_{ij} x_{ij} \leq OPT_i$$

.

7.3.1 Aggregate Clearing Model Another model for ϵ-approximate market clearing has been defined in [10]. According to this model, an allocation (and prices) is ϵ-approximate market clearing if it satisfies the conditions of *budget constraint*, *optimality* and *approximate market clearing* listed below.

$$\begin{aligned} \forall i : \sum_{j=1}^{m} x_{ij} p_j &\leq \sum_{j=1}^{m} a_{ij} p_j \\ \forall i, j : \alpha_i p_j &\geq v_{ij} \\ \forall i, j : x_{ij} > 0 \Rightarrow \alpha_i p_j &= v_{ij} \\ \sum_{j=1}^{m} \mid (\sum_{i=1}^{n} x_{ij} - a_j) \mid p_j &\leq \epsilon \sum_{j=1}^{m} a_j p_j \end{aligned} \tag{22}$$

Note that with this model, the approximate clearing conditions on goods (5) may not be satisfied. It can be shown that in some markets, goods at low prices will remain fully unsold. Our proposed algorithms `main` and `main2` achieve approximate clearing of goods (5) and traders (8). If we relax our approximate market clearing conditions (5) and (8) to (22), we get an algorithm polynomial in $\log(1/\epsilon)$.

Without loss of generality, scale the amount of goods available (and the valuations on them) such that $a_j = 1$ for all j. We show that when $p_j \geq m/\epsilon$ for any j then our algorithm could be stopped satisfying the condition (22).

If $p_j > 1$ in our algorithm then the good numbered j is cleared, i.e. $\sum_{i=1}^{n} x_{ij} = a_j$. If $p_j = 1$ then

$$\begin{aligned} |(\sum_{i=1}^{n} x_{ij} - a_j)| p_j &= |(\sum_{i=1}^{n} x_{ij} - a_j)| \\ &\leq a_j = 1 \end{aligned}$$

This gives

$$\sum_{j=1}^{m} |(\sum_{i=1}^{n} x_{ij} - a_j)| p_j \leq m$$

If there exists k such that $p_k \geq m/\epsilon$ then

$$\begin{aligned} \sum_{j=1}^{m} |(\sum_{i=1}^{n} x_{ij} - a_j)| p_j &\leq \epsilon p_k = \epsilon p_k a_k \\ &\leq \epsilon \sum_{j=1}^{m} p_j a_j \end{aligned}$$

Thus the algorithm `main2` finds a solution satisfying (7), (8) and (22) in $O(\frac{1}{\epsilon}(nm^2 + mn^2) \log p_{max}))$ time, where $p_{max} = m/\epsilon$.

8. Conclusions In this paper we described a parameterized linear-programming formulation for the market clearing problem. The formulation naturally leads to an auction algorithm which approximates market clearing efficiently. The formulation is of independent interest as it could lead to an efficient exact algorithm for the market clearing problem.

References

[1] K. Arrow, H. Block, and L. Hurwicz. On the Stability of the Competetive Equilibrium, II. *Economitrica*, 27:82–109, 1959.

[2] K. Arrow and G. Debreu. Existence of an Equilibrium for a Competitive Economy. *Econometrica*, 22:265–290, 1954.

[3] D. P. Bertsekas. Auction Algorithms for Network Flow Problems: A Tutorial Introduction. *Computational Optimization and Applications*, 1:7–66, 1992.

[4] W. C. Brainard and H. E. Scarf. How to Compute Equilibrium Prices in 1891. Cowles Foundation Discussion Paper (1272), 2000.

[5] J. Cheng and M. Wellman. A Convergent Distributed Implementation of General Equilibrium Outcomes. *Computational Economics*, 12(1):1–24, 1998.

[6] B. Codenotti, S. Pemmaraju, and K. Varadarajan. The computation of market equilibria. *SIGACT News*, 35(4):23–37, 2004.

[7] G. Demange, D. Gale, and M. Sotomayor. Multi-item Auctions. *Journal of Political Economy*, 94(4):863–872, 1986.

[8] X. Deng, C. Papadimitriou, and S. Safra. On the Complexity of Equilibria. In *34th ACM Symposium on Theory of Computing (STOC 2002)*, Montreal, Quebec, Canada, May 2002.

[9] N. R. Devanur, C. H. Papadimitriou, A. Saberi, and V. V. Vazirani. Market equilibrium via a primal-dual-type algorithm. In *Proceedings of the 43rd Symposium on Foundations of Computer Science (FOCS 2002)*, pages 389–395, 2002.

[10] N. R. Devanur and V. Vazirani. An Improved Approximation Scheme for Computing the Arrow-Debreu Prices for the Linear Case. In *Proceedings of Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2003)*, 2003.

[11] B. Eaves. A Finite Algorithm for the Linear Exchange Model. *Journal of Mathematical Economics*, 3(2):197–203, 1976.

[12] E. Eisenberg and D. Gale. Consensus of Subjective Probabilities: The Pari-Mutuel Method. *Annals of Mathematical Statistics*, 30:165–168, 1959.

[13] K. Jain. A polynomial time algorithm for computing the arrow-debreu market equilibrium for linear utilities. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2004)*, pages 286–294, 2004.

[14] K. Jain, M. Mahdian, and A. Saberi. Approximating Market Equilibrium. In *Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX 2003)*, 2003.

[15] H. W. Kuhn. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[16] E. I. Nenakov and M. E. Primak. One algorithm for finding solutions of the Arrow-Debreu model. *Kibernetica*, 3:127–128, 1983.

[17] D. Newman and M. Primak. Complexity of Circumscribed and Inscribed Ellipsod Methods for Solving Equilibrium Economical Models. *Applied Mathematics and Computations*, 52:223–231, 1992.

[18] C. H. Papadimitriou. On the Complexity of the Parity-argument and other Inefficient Proofs of Existence. *Journal of Computer and System Sciences*, 48(3):498–532, June 1994.

[19] M. Primak. A Converging Algorithm for a Linear Exchange Model. *Applied Mathematics and Computations*, 52:223–231, 1992.

[20] P. Samuelson. Foundations of Economic Analysis. Harward University Press, Cambridge, Mass., 1947.

[21] L. Walras. Elements of Pure Economics, or the Theory of Social Wealth (in French). Lausanne, Paris, 1874.

[22] Y. Ye. A Path to the Arrow-Debreu Competetive Market Equilibrium. *Mathematical Programming (to appear)*.