Auction Algorithms for Market Equilibrium

Rahul Garg [*]   Sanjiv Kapoor [†]

ABSTRACT

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 items. The *market equilibrium problem* is to compute a price vector which ensures market clearing, i.e. the demand of a 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 the dual of these parametrized linear programs. We show that finding the market equilibrium is the same as finding a linear-program from the family of programs where the optimal dual solution satisfies certain properties. 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.

Categories and Subject Descriptors

F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity

General Terms

Algorithms, Economics.

Keywords

Auction Algorithms, Market Equilibrium, Approximation Algorithms.

[*]grahul@in.ibm.com, IBM India Research Lab., Block-I, IIT Campus, Hauz Khas, New Delhi, INDIA - 110016.
[†]kapoor@iit.edu, Illinois Institute of Technology, Chicago, USA

1. INTRODUCTION

In this paper we study algorithms for computing market equilibrium in markets with linear utility functions. Consider a market comprising n buyers and m items. Each buyer has an endowment given by a portfolio of items. A finite quantity of each item is available and is assumed to be divisible. Further, each (buyer, item) 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 Leon Walras (1894) proposed the notion of general equilibrium. Walras proposed that a general equilibrium could be achieved by a price-adjustment process called *tatonnement* [14]. 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 has been established by Arrow and Debreu [1]. The proof is non-constructive and the natural question is the existence of an efficient computation process which establishes equilibrium. However, as discussed in Devanur et al. [8], computationally efficient time algorithms had evaded researchers. A polynomial time algorithm for a specific case has been recently proposed in Devanur et al. [8]. In this case the set of traders is partitioned into buyers and sellers. Buyers initially have endowment of money while the sellers initially have items. Moreover, the buyers do not have any value for money and the sellers do not have any value for the items.

In this paper we consider a more general model of market which consists of a set of traders who have an initial endowment of a set of items (one of the items could be money). The traders have different utilities for these items. These utilities are assumed to be linear.

We formulate a family of linear programs parameterized by the price vector. We construct the dual of these programs. The optimal solution of the dual program corresponding to a market clearing price, has a special property. We show that whenever the dual program satisfies this property, the corresponding price vector achieves market clearing. The market clearing conditions arise naturally from the complementary slackness conditions of the parameterized linear program.

We then define an auction mechanism which computes equilibrium prices such that approximate market clearing is achieved, i.e. the surplus is cleared to within ϵ of the total final endowment and the items are almost all sold. The proposed algorithm resembles a primal-dual mechanism similar to Kuhn's methodology for bipartite matching [11] and provides an efficient tatonnement type process [13, 5] for computing approximate market clearing.

The importance of polynomial time schemes has been highlighted in a computer science context by Papadimitriou [12]. Approximation algorithms for the market equilibrium have been considered in [7] where a fixed number of buyers (agents) have been considered. After the development of this algorithm we have become aware of two approximation algorithms recently developed for this problem. The first algorithm [10] provides an approximation using the framework described in [8]. This algorithm achieves complete market clearing and approximate the optimality of the solution. The second algorithm [9] removes the dependence on the sizes of the numbers to achieve a strongly polynomial algorithm. The notion of the approximation achieved in the paper [9] is that of bounding the deficiency or surplus of the goods in terms of their money value. The final algorithm presented in our paper (Section 7) achieves market clearing in terms of the quantity of each available good (and hence in terms of their money value) and approximates the optimality conditions.

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 items, p_{max} is the largest price (assuming that the smallest price is unity), $a_{min} = \min_j a_j$ is the smallest quantity of an available item, n is the number of traders and m is the number of items in the market. The algorithm achieves approximate market clearing in terms of the quantity of each available good. We also give an improved approximation algorithm, which achieves market clearing, and is of complexity $O(\frac{1}{\epsilon}(nm^2 + mn^2)\log p_{max})$. The largest price p_{max} can be loosely bounded by v_{max}/v_{min}, where v_{max} and v_{min} are the largest and smallest values of the positive valued utilities, respectively. The algorithm is polynomial in all the parameters of the problem, except for the error tolerance ϵ.

In comparison, the algorithm presented in [8] solves a special case of this problem but requires $O(m^2(n\log(v_{max}/v_{min}) + \log Mm^2))$ *max-flow computations*, where M is the total money available to compute the exact market clearing prices. The algorithm in [10] solves the problem we address with complexity $\frac{m^4}{\epsilon}(\log m + m\log U + \log M)$ *max-flow computations* where U and M depend on the utilities and endowments, respectively. And finally the algorithm in [9] 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 (roughly $O(\frac{m^7}{\epsilon}\log\frac{m}{\epsilon})$ (assuming max-flow computations are $O((m^3))$ 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. Moreover, if we aim to satisfy the approximation conditions of [9] then the complexity of our method is also strongly polynomial, i.e. $O(\frac{1}{\epsilon}(nm^2 + mn^2)\log\frac{m}{\epsilon})$.

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 discuss how our algorithm can be used to find approximate market equilibrium while allowing some of the utilities to be zero. In Section 7 we show that, under a weaker definition of approximate market clearing [9] the time complexity of our algorithm becomes strongly polynomial. 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 $a_j = \sum_{i=1}^{n} a_{ij}$. 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. The traders exchange their goods so as to maximize their individual utilities. For this presentation, we will assume that the utilities are positive. Note that as a consequence, all prices are positive. This assumption can be removed by a variety of methods. In the conclusion we outline a perturbation method which resolves this issue. Utilities which are 0 can create arbitrary price rises. This has also been observed in the algorithm in [10]. An alternate approach is to detect high prices during the execution of the algorithm.

Let the prices of the goods be represented in terms of an abstract currency, which serves just as a medium of exchange. Given the prices $p_1, p_2, \ldots, p_m$ of the m goods, a trader would like to buy goods with high utility per unit money and sell goods with low utility per unit money. Thus, in equilibrium, trader i will keep only those goods that maximize v_{ij}/p_j, where v_{ij} represents per-unit utility of trader i on good j. Let x_{ij} represent the amount of good j available with trader i. Let $\underline{P}$ represent the $m \times 1$ vector of prices and $\underline{X}$ represent the $n \times m$ matrix of the assignments $x'_{ij}s$. The pair $(\underline{X}, \underline{P})$ forms a market equilibrium iff (a) there is neither a surplus nor a deficiency of any good (including money); (b) all the traders get goods that maximize their utility per unit money spent. The prices $\underline{P}$ are called market clearing prices and $\underline{X}$ is called equilibrium assignment.

The condition for market equilibrium can be mathematically represented as:

$$\forall j : \sum_{i=1}^{n} x_{ij} = a_j \qquad (1)$$

$$\forall i : \sum_{j=1}^{m} x_{ij}p_j = \sum_{j=1}^{m} a_{ij}p_j \qquad (2)$$

$$x_{ij} > 0 \Rightarrow v_{ij}/p_j \geq v_{ik}/p_k \forall k \qquad (3)$$

$$x_{ij} \geq 0, p_j \geq 0$$

Equation (1) implies that there is no deficiency or surplus of any good. Equation (2) implies that there is no deficiency or surplus of money. Equation (3) implies that every trader gets only those goods that maximizes its utility gained per unit money spend on the good.

It must be noted that the model described by Devanur et al. [8] 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 (1), (2) and (3) translate into the market clearing conditions for the model considered by Devanur [8].

3. A PARAMETRIZED 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:

$$\forall j : \sum_{i=1}^{n} x_{ij} = a_j \quad (4)$$

$$\forall i : \sum_{j=1}^{m} x_{ij}p_j = \sum_{j=1}^{m} a_{ij}p_j \quad (5)$$

$$x_{ij} \geq 0$$

If the prices p_j are 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 (4) and α_i for (5)) $DP(\underline{P})$:

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

Subject to:

$$\forall i,j : \alpha_i p_j + \beta_j \geq v_{ij} \quad (6)$$

This gives a family of primal-dual linear programs $LP(\underline{P})$ and $DP(\underline{P})$, one for each value of the price vector $\underline{P}$. It is easy to show that 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:

$$x_{ij} > 0 \quad \Rightarrow \quad \alpha_i p_j + \beta_j = v_{ij} \quad (7)$$

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

LEMMA 1. *A price vector $\underline{P} \geq 0$ forms market clearing prices if the program $DP(\underline{P})$ has an optimal solution with $\beta_j = 0$.*

PROOF. Consider an optimal primal and an optimal dual with $\beta_j = 0$. The optimal primal satisfies the conditions (4) and (5) which are same as (1) and (2). Complementary slackness conditions (7) and the fact $\beta_j = 0$ and the dual feasibility conditions (6) give:

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

□

The following result is immediate from the definition of market clearing prices.

LEMMA 2. *Existence of market clearing prices is equivalent to the existence of a program $DP(\underline{P})$ which has an optimal solution with $\beta = 0$.*

4. AN AUCTION ALGORITHM FOR MARKET CLEARING PRICES

We now present an approximate algorithm for discovering the 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.

This approach has a similarity with the Hungarian method of Kuhn [11] for soving the assignment problem. Unlike the Hungarian method which raises the price of all the goods in a *minimal over-demanded set* by a specific amount, our algorithm raises the price of one good at a time 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. Prior to this, auction algorithms have been proposed for maximum weight matching in bipartite graphs and network flow problems [6, 3, 2].

The dual variables α_i are chosen such that the dual feasibility condition (6) is satisfied. During the course of the algorithm, the variables x_{ij} are successively modified by a bidding process such that the following relaxed primal and relaxed complementary slackness conditions are always satisfied:

$$\forall j : \text{If } p_j > 1 \text{ then } \textstyle\sum_{i=1}^{n} x_{ij} = a_j \text{ else } \sum_{i=1}^{n} x_{ij} \leq a_j \quad (8)$$

$$\forall i : \textstyle\sum_{j=1}^{m} x_{ij}p_j \leq (1+\epsilon)\sum_{j=1}^{m} a_{ij}p_j \quad (9)$$

$$x_{ij} > 0 \Rightarrow v_{ij} \leq \alpha_i p_j \leq (1+\epsilon)v_{ij} \quad (10)$$

The bidding process raises the prices. As these prices increase, the inequalities (8) and (9) become tighter. These become very close to (4) and (5) when the algorithm terminates. 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 demand set D_i of trader i as:

$$D_i = \arg\max_j v_{ij}/p_j \quad (11)$$

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

$$r_i = \sum_{j=1}^{m} a_{ij}p_j - \sum_{j=1}^{m} y_{ij}\frac{p_j}{1+\epsilon} - \sum_{j=1}^{m} h_{ij}p_j \quad (12)$$

and the total surplus $r = \sum_{i=1}^{n} r_i$. Let $a_{min} = \min_j a_j$ and $a = \sum_{j=1}^{m} a_j$. 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 $\sum_{i=1}^{n} h_{ij} < a_j$.

At the beginning of the algorithm, the prices (p_j) are initialized to 1 and the variables y_{ij}, h_{ij}, x_{ij} are initialized to zero. 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. If the good j is available at its current price p_j, it is acquired by outbidding another trader 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 $(1+\epsilon)$ and hence, the good is made available. This process continues until either the surplus of the traders becomes sufficiently small or all the goods are assigned. The details of the algorithm are given in Figure 1.

```
algorithm main
$\forall i, j : x_{ij} = y_{ij} = h_{ij} = 0$;  $p_j = 1$;
$r_i = \sum_{j=1}^{m} a_{ij} p_j$;
$\alpha_i = \max_j v_{ij}/p_j$;
repeat
        pick $i$ s.t.  $r_i > 0$
        $\alpha_i = \max_j v_{ij}/p_j$
        pick $j \in D_i$
        if $\sum_{k=1}^{n} x_{kj} < a_j$ then assign($i$, $j$)
        else if $\exists k$ s.t.  $y_{kj} > 0$ then outbid($i$, $j$, $k$)
        else raise_price($j$)
until $\forall i : r_i < \frac{\epsilon}{n(1+\epsilon)} a_{min}$ or $\forall j : \sum_{i=1}^{n} x_{ij} = a_j$
end algorithm main

procedure outbid($i$, $j$, $k$)
        $t = \min(y_{kj}, \frac{r_i}{p_j})$
        $h_{ij} = h_{ij} + t$
        $y_{kj} = y_{kj} - t$
        $r_i = r_i - tp_j$
        $r_k = r_k + t\frac{p_j}{1+\epsilon}$
end procedure

procedure assign($i$, $j$)
        $t = \min(a_j - \sum_{k=1}^{n} x_{kj}, \frac{r_i}{p_j})$
        $h_{ij} = h_{ij} + t$
        $r_i = r_i - tp_j$
end procedure

procedure raise_price($j$)
        $\forall k : y_{kj} = h_{kj}$
        $\forall k : h_{kj} = 0$
        $\forall i : r_i = r_i + \epsilon a_{ij} p_j$
        $p_j = (1+\epsilon) p_j$
end procedure
```

Figure 1: An auction algorithm for discovering market clearing prices

Lemma 3. *During the progress of the auction algorithm, conditions (8), (9), (10) and (6) are always satisfied.*

Proof. Condition (8) is satisfied after the initialization step. Note that procedure outbid() does not change the values of p_j and $\sum_{i=1}^{n} x_{ij}$. So, if condition (8) is satisfied before the entry to the procedure, it is also satisfied after its exit. Procedure raise_price() can only be entered if $\sum_{k=1}^{n} x_{kj} = a_j$. It does not change the value of x_{ij}. Therefore (8) will be satisfied at the end of the procedure. In the procedure assign(), p_j is unchanged and x_{ij} is only increased. The variable t is chosen such that (8) remains satisfied.

For condition (9) 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 $r_i \geq 0$. Procedure raise_price() does not change the value of $\sum_{j=1}^{m} y_{ij} p_j/(1+\epsilon) + \sum_{j=1}^{m} h_{ij} p_j$ for any i. As a result r_i as defined in (12), can only increase in this procedure. Therefore (9) remains satisfied throughout the algorithm.

The variables α_i's are set during initialization step and the update step in such a way that (6) is satisfied. Procedure raise_price(j) only increases p_j. Therefore (6) remains satisfied throughout the algorithm.

Condition (10) is satisfied after the initialization step. Since (6) is satisfied throughout the algorithm $v_{ij} \leq \alpha_i p_j$. We just need to show that:

$$x_{ij} > 0 \Rightarrow \alpha_i p_j \leq (1+\epsilon) v_{ij}. \tag{13}$$

Since p_j's can only increase as the algorithm progresses, α_i can only decrease. Therefore, if (13) 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 (13). It remains to be seen that (13) continues to be satisfied after raise_price() is called.

Note that when assign() or outbid() is called, $j \in D_i$ i.e. $v_{ij} = \alpha_i p_j$. 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 (13) remains satisfied. So, it is sufficient to show that between two successive calls to raise_price(j), outbid(i, j, k) is called for every i 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 if $\forall k, y_{kj} = 0$. outbid(i, j, k) is the only place where 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. $\square$

4.1 Analysis

We first show that when the algorithm terminates, conditions (4) and (5) are approximately satisfied.

Lemma 4. *When algorithm main terminates, the following conditions are satisfied:*

$$\forall j : \frac{a_j}{1+\epsilon} \leq \sum_{i=1}^{n} x_{ij} \leq a_j \tag{14}$$

$$\forall i : \sum_{j=1}^{m} x_{ij} p_j \leq \sum_{j=1}^{m} a_{ij} (1+\epsilon) p_j \tag{15}$$

$$\frac{1}{1+\epsilon} \sum_{j=1}^{m} a_j p_j \leq \sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij} p_j \leq \sum_{j=1}^{m} a_j p_j \tag{16}$$

Proof. Condition (15) follows from (12) and the fact that $r_i \geq 0$ when the algorithm terminates. The condition (16) follows from multiplying (14) with p_j and summing them up for all j. The second inequality of (14) follows from the invariant (8). We now show that the first inequality of (14) 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 (14) is trivially satisfied. In the other case we have, $r_i \leq \frac{\epsilon}{n(1+\epsilon)} a_{min}$. From (12) we have:

$$\begin{aligned} r_i &= \sum_{j=1}^{m}(a_{ij}p_j - x_{ij}p_j) + \frac{\epsilon}{(1+\epsilon)}\sum_{j=1}^{m} y_{ij}p_j \quad (17) \\ \Rightarrow \sum_{i=1}^{n} r_i &= \sum_{j=1}^{m}\sum_{i=1}^{n}(a_{ij} - x_{ij})p_j + \frac{\epsilon}{(1+\epsilon)}\sum_{j=1}^{m}\sum_{i=1}^{n} y_{ij}p_j \end{aligned}$$

From (8) it follows that if $\sum_{i=1}^{n} x_{ij} < a_j$ then $p_j = 1$. Since $r_i \leq \frac{\epsilon}{n(1+\epsilon)} a_{min}$, $\sum_{i=1}^{n} r_i \leq \frac{\epsilon}{1+\epsilon} a_{min}$. Therefore we have,

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

Since $y_{ij} \geq 0$ and $\sum_{i=1}^{n} a_{ij} \geq \sum_{i=1}^{n} x_{ij}$,

$$\begin{aligned} \Rightarrow \forall j : a_j - \sum_{i=1}^{n} x_{ij} &\leq \frac{\epsilon}{(1+\epsilon)} a_{min} \\ \Rightarrow \forall j : \frac{a_j}{(1+\epsilon)} &\leq \sum_{i=1}^{n} x_{ij} \end{aligned}$$

□

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 $m \log_{(1+\epsilon)} p_{max} = O(\frac{m}{\epsilon} \log p_{max})$. In order to show convergence, we first need to bound p_{max}.

LEMMA 5. *Let $v_{min} = \min_{i,j} v_{ij}$. Let $v_{max} = \max_{i,j} v_{ij}$. For all j, $p_j \leq (1+\epsilon)v_{max}/v_{min}$.*

PROOF. We first show that there is at least one good that stays at price 1. The price of a good can increase only when it is completely assigned. The algorithm terminates if all the goods are completely assigned. Therefore, there is at least one good at unit price during the course of the algorithm main. Let this good be k.

Consider good j s.t. $p_j > 1$. Using (8) one can always pick i such that $x_{ij} > 0$. From (10) we get $\alpha_i p_j \leq (1+\epsilon)v_{ij}$. From (6) we have $\alpha_i p_k \geq v_{ik}$. Since $p_k = 1$, we have $p_j \leq (1+\epsilon)v_{ij}/v_{ik} \leq (1+\epsilon)v_{max}/v_{min}$. □

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. However, if the bidding is organized in rounds with each trader exhausting its surplus in every round then the algorithm can be shown to terminate in polynomial time.

LEMMA 6. *If each trader exhausts its surplus once in a round, then either there is a price rise in the round, or the total surplus (r) reduces by a factor $1/(1+\epsilon)$ in the round.*

PROOF. Assume that raise_price() is never called in the round. Let r represent the value of the total surplus at the beginning of the round, and r_i represent the surplus of trader i at the beginning of the round. We now put a lower bound on the surplus reduction in the round.

Note that a call to assign() decreases the value of the total surplus by t (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 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: $r' \leq r/(1+\epsilon)$. □

THEOREM 1. *If the bidding is organized in rounds, and each trader exhausts its surplus once in every round, then the algorithm main terminates in $O(\frac{1}{\epsilon^2} m \log(\frac{p_{max}a}{\epsilon a_{min}}) \log p_{max})$ rounds.*

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 surplus increase by a single call to raise_price() is bounded by ap_{max} where $a = \sum_{j=1}^{m} a_j$. The algorithm terminates 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. □

Further note that in each round, where each trader is considered once, a trader, say i, having a surplus r_i and chosen for processing either exhausts the surplus on a particular good j or y_j becomes zero. The first event is charged to the trader i and the second to a price rise (which happens for good j when y_j is exhausted). Thus in every round n charges corresponding to the first type of events are generated. This leads to the bound of $O(n(\frac{1}{\epsilon^2} m \log(\frac{p_{max}a}{\epsilon a_{min}}) \log p_{max})$

We next present a modification to the above basic algorithm that gives better convergence and a stronger market clearing condition.

5. A FASTER 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 and (c) bidding is organized in rounds such that every trader exhausts its surplus at least once in every round. These changes are incorporated in the algorithm main2 using procedures outbid2() as shown in Figure 2.

Before moving further into the analysis, we introduce some notations. Consider a directed bipartite graph $G =$

```
algorithm main2
$\forall i,j : x_{ij} = y_{ij} = h_{ij} = 0$; $p_j = 1$;
$r_i = \sum_{j=1}^{m} a_{ij}p_j$;
$\alpha_i = \max_j v_{ij}/p_j$;
repeat
    compute demand sets $D_i$; $\alpha_i = \max_j v_{ij}/p_j$
    repeat
        if there is no trader with $r_i > 0$ then done
        if there is no unassigned good then done
        for all traders $i$ $incomplete_i = FALSE$
        while $\exists i$ s.t.  $r_i > 0$, $incomplete_i = FALSE$
          and raise_price() is not called
            pick $j \in D_i$
            if $\sum_{k=1}^{n} x_{kj} < a_j$ then assign($i$, $j$)
            else if $y_{ij} > 0$ then outbid2($i$, $j$, $i$)
            else if $\exists k$ s.t.$y_{kj} > 0$ then outbid2($i$, $j$, $k$)
            else raise_price($j$)
            if $r_i = 0$ then $incomplete_i = TRUE$
        end while
    until (raise_price() is called) or (done)
until done
end algorithm main2

procedure outbid2($i$, $j$, $k$)
    if $j \notin D_k$ and $i \neq k$ then
        $t = \min(y_{kj}, \frac{r_i}{p_j})$
        $h_{ij} = h_{ij} + t$
        $y_{kj} = y_{kj} - t$
        $r_i = r_i - tp_j$
        $r_k = r_k + tp_j/(1+\epsilon)$
    else
        $t = \min(\frac{\epsilon}{(1+\epsilon)}y_{kj}, \frac{r_i}{p_j})$
        $h_{kj} = h_{kj} + t/\epsilon$
        $y_{kj} = y_{kj} - t(1+\epsilon)/\epsilon$
        $h_{ij} = h_{ij} + t$
        $r_i = r_i - tp_j$
    endif
end procedure
```

Figure 2: A faster auction algorithm

(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.

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

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

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 times edges are 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(). This is dependent on the number of times r_i becomes positive after it has been set to zero. Note that, in the process of bidding (when outbid2(i, j, k) is called), the surplus (r_i) may be transferred to another trader (r_k). It can be seen from the definition of procedure outbid2() that, surplus may the 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 that establishes that the surplus from a trader i cannot cycle back to itself without a price rise.

LEMMA 8. *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 $y_{u2v1} > 0$ and $v1 \notin D_{u2}$. 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 will be 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 9. *After d rounds of bidding either there is a price rise, or surplus of at least d traders is guaranteed to be 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. Let us assign a unique rank between 1 and n to each trader using a topological sort of the graph G. Trader with rank 1 is a trader with no incoming edge.

LEMMA 10. *If there is no price rise till 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. 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. Trader k has no incoming edge in G. 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 k is a maximal trader in G.

Note that as the bidding progresses without a price rise, D remains unchanged. Moreover, the sets Y and B shrink. 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. Therefore, it will have a zero surplus at the end of round d and thereafter. $\square$

LEMMA 11. *The algorithm main2 terminates in* $O(\frac{1}{\epsilon}m\log p_{max})$ *iterations of the outer loop.*

PROOF. From Lemma 9 it follows in each iteration of the outer loop, either there is a price rise within n rounds of bidding or the total surplus r goes to zero. The algorithm terminates if the total surplus goes to zero. There can be at most $O(\frac{1}{\epsilon}m\log p_{max})$ price rises. Hence the result. $\square$

THEOREM 2. *The algorithm main2 terminates in* $O(\frac{1}{\epsilon}(nm^2+mn^2)\log(v_{max}/v_{min}))$ *steps.*

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

From Lemma 7, 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. $\square$

LEMMA 12. *When the algorithm main2 terminates conditions (4), (15) and the following*

$$\sum_{i=1}^{n}\sum_{j=1}^{m}a_{ij}p_j = \sum_{i=1}^{n}\sum_{j=1}^{m}x_{ij}p_j \tag{18}$$

are satisfied.

PROOF. Condition (15) is satisfied because $r_i \geq 0$. The algorithm main2 terminates either when all the goods are assigned or when there is no surplus left. In the former case condition (4) will be satisfied and hence (18) will also be satisfied. In the latter $\forall i : r_i = 0$. Summing (17) 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. Therefore conditions (15) and (18) will be satisfied. $\square$

6. PERTURBATION BOUNDS

Non-zero utility functions can be obtained by perturbing the valuations by small amounts. 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.

The zero co-efficients of the utility function are modified to be $\delta = \epsilon/(na)$. Now, the error on the sum of all the individual objective functions of the traders is bounded by utmost ϵ. Further, the relaxed complementary slackness constraints (10) is satisfied to within an error of ϵ. The complexity of the algorithm remains bounded since the perturbation modifies the term $\log p_{max}$ which is bounded by $\log v_{max} + \log(na) + \log(1/\epsilon)$.

7. A WEAKER MODEL FOR APPROXIMATE MARKET CLEARING

In this section we show that with the model of [9], our algorithm becomes strongly polynomial. For the purpose of this section we will assume that each a_j, the amount available of good j is unity. This can be achieved by scaling [9]. We modify our terminating condition so that the algorithm stops when there is an item with price greater than m/ϵ. An analysis of the excess money demand/supply gives the approximation bound and a polynomial algorithm with improved time complexity.

Consider the following approximating condition [9]: An allocation satisfies the ϵ-approximate market clearing condition if neither deficiency nor surplus is too high in value i.e.,

$$|\xi(p) - p| \leq \epsilon, ||p|| = 1 \tag{19}$$

where $\xi()$ is the demand in terms of money, $\xi_j(p) = \Sigma_{i\in T}x_{ij}p_j$ and p is the price vector.

We show that when $p_j \geq m/\epsilon$ for any j then our algorithm could be stopped satisfying the condition (19).

If $p_j > 1$ in our algorithm then the good numbered j is cleared, i.e. $\xi_j(p) = p_j$. If $p_j = 1$ then $p_j - \xi(p) = p_j - \sum_{i=1}^{n}x_{ij}p_j = 1 - \sum_{i=1}^{n}x_{ij} \leq 1$. This implies that $|p - \xi(p)| \leq m$. If there is a j such that $p_j \geq m/\epsilon$ then

$$\frac{1}{\sum_{j=1}^{m}p_j}|p - \xi(p)| \leq \epsilon$$

Now the normalized price vector $\hat{p} = p/\sum_{j=1}^{m}p_j$ satisfies:

$$|\hat{p} - \xi(\hat{p})| \leq \epsilon, ||\hat{p}|| = 1$$

Thus the algorithm main2 terminates in $O(\frac{1}{\epsilon}(nm^2 + mn^2)\log p_{max}))$ where $p_{max} = m/\epsilon$ and is strongly polynomial.

8. CONCLUSIONS

In this paper we described a parametrized 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.

Improving the complexity of the approximations should be possible by a careful choice of the bidding order.

9. REFERENCES

[1] K. Arrow and G. Debreu. Existance of an equilibrium for a competitive economy. *Econometrica*, 22:265–290, 1954.

[2] V. Bansal and R. Garg. Simultaneous Independent Online Auctions with Discrete Bid Increments. *Electronic Commerce Research Journal: Special issue on Dynamic Pricing Policies in Electronic Commerce*, To Appear.

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

[7] 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.

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

[9] N. R. Devanur and V. Vazirani. An improved approximation scheme for computing the arrow-debreu prices for the linear case. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2003)*, 2003.

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

[11] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[12] 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.

[13] P. Samuelson. Foundations of economic analysis. Harward University Press, Cambridge, Mass., 1947.

[14] L. Walras. Elements of pure economics, or the theory of social wealth (in French). Lausanne, Paris, 1874.