

Price Roll-Backs and Path Auctions: An Approximation Scheme for Computing the Market Equilibrium

Rahul Garg¹ and Sanjiv Kapoor²

¹ IBM India Research Lab., New Delhi.

² Illinois Institute of Technology, Chicago, USA

Abstract. In this paper we investigate the structure of prices in approximate solutions to the market equilibrium problem. The bounds achieved allow a scaling approach for computing market equilibrium in the Fisher model. Our algorithm computes an exact solution and improves the complexity of previously known combinatorial algorithms for the problem. It consists of a price roll-back step combined with the auction steps of [11]. Our approach also leads to an efficient polynomial time approximation scheme. We also show a reduction from a flow problem to the market equilibrium problem, illustrating its inherent complexity.

1 Introduction

We consider the *market equilibrium problem* with non-negative and linear utilities. The combinatorial structure of the problem has become of interest recently in the design of algorithms, though interest in the structure of this problem dates back further. Arrow and Debreu [1] showed the existence of equilibrium prices under weak assumptions, using a non-constructive argument. Arrow et al. [2], showed that the set of equilibrium prices is convex if the utility functions satisfy the *weak gross substitutability (WGS)* property.

Active interest in designing efficient algorithms to solve the market equilibrium problem dates back to the works of Fisher [3] who designed a hydraulic apparatus to compute market equilibrium. Eaves [8] formulated the market equilibrium problem with linear utilities as a linear complementarity problem and used Lemke's algorithm to solve it. Recently, polynomial-time algorithms have been designed for a number of special cases using a variety of algorithmic techniques.

The techniques may be broadly classified into combinatorial techniques and techniques using convex programming methods. The characterization of equilibrium prices by Arrow et al. [2] leads to an ellipsoid method for computation of market equilibrium in polynomial time (for WGS utilities). This approach has been surveyed in a recent note by Codenotti et al. [4]. Algorithms utilizing circumscribed and inscribed ellipsoids have been discussed in [16, 17].

Another convex programming approach is to transform the market equilibrium problem into a convex feasibility (or optimization) problem. This includes the works of Eisenberg and Gale [10, 9] who reduced the market equilibrium problem in the Fisher model with linear utilities into a convex optimization problem. Similarly, the market equilibrium problem for the general Walrasian model with linear utilities can be transformed into a convex feasibility problem [15, 14, 21].

The combinatorial techniques solve the problem for restricted cases and often give approximate solution to the problem. These may be classified into (a) primal-dual techniques algorithms based on maximum flows [5, 20, 7] and (b) the

auction based approaches [11, 12]. Devanur et al. [5] solve the market equilibrium problem for the Fisher model with linear utilities using a primal-dual technique based on maximum flows. Using this approach, an algorithm for approximate market equilibrium for the Walrasian model with linear utilities was presented in [13]. This algorithm was further improved in [6].

An efficient auction algorithm was designed to find approximate market equilibrium in the Walrasian model with linear utilities [11]. The auction approach is also applicable to separable gross substitute utilities in the Fisher model [12].

In this paper we study the structure of prices achieved via a typical approximation method. Realizing reasonable bounds on the difference between the approximate and optimal prices allows us to design an approximation scheme for efficient computation of market equilibrium in the Fisher model with linear utilities. We give a definition of approximate market equilibrium and establish a bound on the ratio of prices in approximate and exact equilibrium. We show that $\frac{1}{(1+\epsilon)^n} p_j^* \leq p_j \leq (1+\epsilon)^{n-1} p_j^*$ where p_j, p_j^* are the prices of item j achieved by the approximate and optimal schemes, respectively. Achieving these bounds appears challenging enough itself and achieving similar bounds for the general exchange model is more complicated due to a feedback effect of the error on the prices on the endowment.

Using the price bounds, we devise a price-roll back mechanism and design an algorithm, using the distributed auction mechanism [11, 12], to give an approximation scheme for efficiently computing the market equilibrium. This algorithm uses the auction approach to find an approximate solution. It then successively improves the approximation by a price roll-back and uses the auction approach again with a smaller bid increment (ϵ). This algorithm finds an $(1+\epsilon)$ approximate market equilibrium in $O((n^3m + n^2m \log M) \log(1/\epsilon))$ steps, where n and m are the number of buyers and items respectively and $M (\leq ev_{max} a_{max} / e_{min} v_{min} a_{min})$ is a bound on the ratio of initial to final prices. Note that the time complexity of our algorithm depends logarithmically on the approximation factor ϵ . Since the equilibrium prices are rational numbers (assuming that the input is rational), ϵ may be chosen sufficient small to give an exact market equilibrium. This leads to an exact algorithm of complexity $O((n^3m + n^2m \log M)L)$ (L is the bit complexity of the input data, the data being assumed to be rational), which improves the current best complexity, $O(n^4)$ max-flow computation, of the best known combinatorial algorithm ([5]) and matches the currently best known complexity of $O(n^4L)$ [21], which uses an interior point approach.

We further improve on the auction mechanism by designing *path-auction* mechanisms. In this mechanism, the amount of good j that a buyer bids for, is dependent on the re-allocation possible along a sequence of buyer-good pairs $(b_1, g_1, b_2, g_2 \dots b_k, g_k)$ where every even pair, (g_i, b_{i+1}) represents the return of good i to buyer $i+1$ and the pair (b_i, g_i) represents the bid for good i by buyer i . These path auctions are efficiently implemented using a variant of the dynamic tree data structure of Sleater and Tarjan [18]. The data structure is modified to handle multiplier flows. The path auctions are then carried out using multiplier flows on dynamic trees. This leads to a time complexity of $O((n^3 + n^2 \log M)L)$. In comparison to the best known algorithm in [21], which is based on interior point methods and presumably requires careful handling of numerical precision, our algorithm is combinatorial in nature, has a natural economic interpretation, improves the approximation in successive iterations and is $O(n)$ times faster.

Finally, we show a reduction from the maximum flow problem to the market equilibrium problem (in the Fisher model with linear utilities). This throws some light on the hardness of the market equilibrium problem.

The rest of the paper is organized as follows: In Section 2 we formally define the market equilibrium problem considered by us. This is followed by a discussion

of approximate market equilibrium in Section 3. In this section, we establish the bound on prices of approximate market equilibrium. In Section 4 we describe our price-rollback scheme, outline the proof of its correctness and time complexity. In section 5 we describe the algorithm based on the path-auction mechanism. We show how to use multiplier flows and dynamic trees to achieve the improved time bound. In Section 6 we present the reduction from the maximum flows to the market equilibrium problem.

2 Market Model and Preliminaries

We now review the market model of Fisher [3] with linear utilities. Consider a market consisting of a set B of n buyers and a set S of m divisible goods. Buyer i has an amount of money equal to e_i . The amount of good j available in the market is a_j . Assume that the utility of buyers on these goods is linear. Buyer i has a per-unit utility of v_{ij} on good j . Assume that the buyers have no utility for money. The buyers use their money to purchase the goods that maximize their utility.

Given prices p_1, p_2, \dots, p_m of these m goods, the buyers use their money to purchase goods that maximize their individual utilities. Thus a buyer i will select a good j that maximizes v_{ij}/p_j . Let x_{ij} represent the amount of good j purchased by buyer i . We say that the pair $(\underline{X}, \underline{P})$ forms a market equilibrium if (a) the buyers have spent all their money; (b) there is neither a surplus or a deficiency of any good; (c) all the buyers get items that maximize their utility per unit money spent. The prices \underline{P} are called market clearing prices and the allocation \underline{X} is called the equilibrium allocation.

The condition for market equilibrium can be mathematically represented as:

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

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

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

where $x_{ij} \geq 0, p_j \geq 0$. The equations (1) and (2) imply that all the goods are sold and all the buyers have exhausted their budget. Equation (3) implies that every buyer gets only those goods that maximize its total utility. It was shown in [10] that the market equilibrium price for the above model is unique.

The market equilibrium problem can be formulated as a solution to a specific primal-dual program (derived from a family of Linear Programs)[11,12]. The equations corresponding to the "restricted" dual program are given by:

$$\forall i, j : \alpha_i p_j \geq v_{ij} \quad (4)$$

and the corresponding complementary slackness conditions are given by:

$$\forall i, j : x_{ij} > 0 \Rightarrow \alpha_i p_j = v_{ij} \quad (5)$$

where $\alpha_i \geq 0$ are the dual variables.

Theorem 1. *Any solution $(\underline{X}, \underline{P}, \underline{\alpha})$ with $\underline{X} \geq 0, \underline{P} \geq 0$ and $\underline{\alpha} \geq 0$, satisfying the conditions (1), (2), (4) and (5), constitutes a market equilibrium.*

For the proof the reader is referred to [11, 12]

3 Approximate Market Equilibrium

Define a solution $(\underline{X}, \underline{P}, \underline{\alpha})$ to be $(1 + \epsilon)$ -approximate market equilibrium if it satisfies (1), (4) and the following “ ϵ -relaxation” of the conditions (2) and (5):

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

$$\forall i, j : x_{ij} > 0 \Rightarrow v_{ij} \leq \alpha_i p_j \leq (1 + \epsilon) v_{ij} \quad (7)$$

Theorem 2. *Let \underline{P}^* be the unique market equilibrium price. Let \underline{X}^* be corresponding equilibrium allocation. Let $(\underline{P}, \underline{X}, \underline{\alpha})$ be a $(1 + \epsilon)$ -approximate market equilibrium (satisfying (1), (4), (6) and (7)). Then, for all j the following must be true:*

$$\frac{p_j^*}{(1 + \epsilon)^n} \leq p_j \leq (1 + \epsilon)^{n-1} p_j^*.$$

For the proof of the above theorem, we first construct a directed weighted bipartite graph (called the assignment graph G_a) with the set of buyers B and the set of goods S as vertices on the two side. We use the market equilibrium $(\underline{X}^*, \underline{P}^*)$ and the approximate market equilibrium $(\underline{X}, \underline{P})$ to define the weight on the edges of G_a . We then construct a reduced (weighted) directed acyclic subgraph G_r by successively removing cycles in G_a . We prove some properties of the graph G_r and use these properties to prove Theorem 2.

Define:

$$\gamma_i = \frac{e_i}{\sum_{j=1}^m x_{ij} p_j}$$

Since the solution $(\underline{X}, \underline{P})$ satisfies (6), $1 \leq \gamma_i \leq 1 + \epsilon$ for all i . The weight function $w_a : ((B \times S) \cup (S \times B)) \rightarrow R_+$ of the graph G_a is defined as follows:

$$\begin{aligned} w_a(i, j) &= x_{ij} p_j \gamma_i \quad \forall (i, j) \in B \times S \\ w_a(j, i) &= x_{ij}^* p_j^* \quad \forall (j, i) \in S \times B \end{aligned}$$

The edge e is present in the graph G_a iff $w(e) > 0$. Note that, with this construction following is true:

$$\sum_{j \in S} w_a(i, j) = \sum_{j \in S} w_a(j, i) = e_i$$

Define the average price of an item j (\hat{p}_j) in the approximate market equilibrium as follows:

$$\begin{aligned} \hat{p}_j &= \frac{\sum_{i \in B} w_a(i, j)}{a_j} \\ &= \sum_{i \in B} \frac{\gamma_i x_{ij} p_j}{a_j} \end{aligned}$$

Since $1 \leq \gamma_i \leq 1 + \epsilon$, and $\sum_{i=1}^n x_{ij} = a_j$, we have:

$$p_j \leq \hat{p}_j \leq (1 + \epsilon) p_j \quad (8)$$

The reduced acyclic sub-graph G_r is constructed from G_a using a sequence of “cycle removal” steps leading to graphs $G_a \equiv G_0, G_1, G_2, \dots, G_k, G_{k+1} \equiv G_r$.

At step l , a cycle in the graph G_{l-1} is located. Let e_l be the minimum weight edge in this cycle. The graph G_l is obtained from G_{l-1} by subtracting $w(e_l)$ from all edges in the cycle and removing the zero weight edges (including the edge e_l). Every step removes at least one edge (e_l) of the graph G_{l-1} . Therefore, this procedure is guaranteed to terminate giving an acyclic graph G_r . We now prove some properties of G_r .

Lemma 1. *No maximal path in G_r can start or end at a buyer vertex $i \in B$.*

Lemma 2. *If a maximal path in G_r starts at an item $k \in S$ and ends at an item $k' \in S$ then $p_k^* > \hat{p}_k$ and $p_{k'}^* < \hat{p}_{k'}$.*

Lemma 3. *If an item $k \in S$ is disconnected in G_r then $p_k^* = \hat{p}_k$.*

Lemma 4. *If there is a path of length $2l$ from item k to item k' in the graph G_a then*

$$\frac{p_{k'}}{p_k} \leq (1 + \epsilon)^l \frac{p_{k'}^*}{p_k^*}$$

Proof of Theorem 2 Consider any item $j \in S$. If j is disconnected in G_r , then Lemma 3 gives $p_j^* = \hat{p}_j$. Using (8) we get:

$$\frac{p_j^*}{1 + \epsilon} \leq p_j \leq p_j^*$$

If j is connected in G_r , then consider any maximal path in G_r containing j . According to Lemma 1, this path must start at an item (say k) and end at an item (say k'). Using Lemma 2 and (8) we get:

$$p_k \leq \hat{p}_k < p_k^* \tag{9}$$

$$p_{k'}^* < \hat{p}_{k'} \leq (1 + \epsilon)p_{k'} \tag{10}$$

The length of the path from k to j is bounded by $2(n-1)$. Therefore, Lemma 4 can be used to get:

$$\begin{aligned} \frac{p_j}{p_k} &\leq (1 + \epsilon)^{n-1} \frac{p_j^*}{p_k^*} \\ \Rightarrow \frac{p_j}{p_j^*} &\leq (1 + \epsilon)^{n-1} \frac{p_k}{p_k^*} \end{aligned}$$

The above inequality alongwith with (9) gives:

$$p_j \leq (1 + \epsilon)^{n-1} p_j^*$$

Similarly, there is a path from item j to item k' of length at most $2(n-1)$. Again, using Lemma 4 we have:

$$\frac{p_{k'}}{p_{k'}^*} \leq (1 + \epsilon)^{n-1} \frac{p_j}{p_j^*}$$

Using (10) the above reduces to:

$$p_j \geq \frac{1}{(1 + \epsilon)^n} p_j^*$$

This proves that for all items $j \in S$,

$$\frac{1}{(1 + \epsilon)^n} p_j^* \leq p_j \leq (1 + \epsilon)^{n-1} p_j^*$$

4 Successive Approximations by Price Rollback

We now describe the algorithm `roll-back` (shown in Figure 1), which provides a polynomial method for solving the market equilibrium problem exactly (or to within any specified accuracy). This algorithm relies on algorithm listed as

```

algorithm roll-back
  initialize
   $\epsilon = \epsilon_0 = 1$ 
  call algorithm auction( $\epsilon$ )
  while ( $\epsilon > \delta$ ) do
     $\forall j : p_j \leftarrow p_j / (1 + \epsilon)^{2n}$ 
     $\forall i, j : y_{ij} = y_{ij} + h_{ij}; h_{ij} = 0;$ 
     $\epsilon \leftarrow \epsilon / 2$ 
     $\forall i : r_i = e_i - \sum_{j=1}^m y_{ij} p_j / (1 + \epsilon)$ 
    call algorithm auction( $\epsilon$ )
  end while
end algorithm roll-back

```

Fig. 1. The price rollback algorithm

algorithm `auction` in Figure 2. This algorithm is a simplification of `main` in [12].

The algorithm begins with $\epsilon = 1$ and the initial solution as in [12]. It then calls `algorithm auction` to get a 2-approximate market equilibrium. It then scales down the prices by a factor $(1 + \epsilon)^{2n}$ and reduces ϵ to half of its current value. It then calls `algorithm auction` with the new value of ϵ to give a $(1 + \epsilon)$ -approximate market equilibrium. This process continues until ϵ is sufficiently small.

Lemma 5. *For any $0 \leq \epsilon_1 \leq \epsilon_0$, if the algorithm `algorithm auction` is called with $\epsilon = \epsilon_1$ and an initial solution satisfying (1), (4), (7), (11) and (12)*

$$\forall i : \sum_{j=1}^m x_{ij} p_j \leq e_i \quad (11)$$

$$\forall j : p_j \leq \frac{p_j^*}{(1 + \epsilon)^{n+1}} \quad (12)$$

with $\epsilon = \epsilon_0$, then it terminates with $(1 + \epsilon_1)$ -approximate market equilibrium.

This leads to the following result.

Theorem 3. *Iteration k of procedure `roll-back` finds a $(1 + \epsilon_k)$ -approximate market equilibrium, where $\epsilon_k = \epsilon_0 / 2^k$.*

To prove the complexity of the algorithm, we need to bound the time taken by a call to `algorithm auction`. For this we recall the relevant portions of the analysis in [11]. An item is sold at two prices and h_{ij} , y_{ij} respectively represent the amounts of item j sold to buyer i at prices p_j and $p_j / (1 + \epsilon)$. For a buyer i its demand set is defined as $D_i = \arg \max_j v_{ij} / p_j$. Define \bar{D} to be the set of demand edges, X the set of assignment edges, Y a subset of the set of assignment edges and Z a subset of Y as follows.

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

```

procedure initialize
   $\forall i, \forall j : y_{ij} = 0; \forall i \neq 1, \forall j : h_{ij} = 0$ 
   $\forall j : h_{1j} = a_j$ 
   $\forall j : \alpha_{1j} = (\sum_j a_j v_{1j}) / e_1$ 
   $\forall j : p_j = v_{1j} / \alpha_{1j} ; \forall i : \alpha_i = \max_j v_{ij} / p_j$ 
   $\forall i \neq 1 : r_i = e_i ; r_1 = 0$ 
end procedure initialize

algorithm auction( $\epsilon$ );
  repeat
    forall buyers  $i$  do
       $D_i = \arg \max_j v_{ij} / p_j$ 
       $\alpha_i = \max_j v_{ij} / p_j$ 
      while  $r_i > 0$  do
        pick  $j \in D_i$ 
        if  $\exists k : y_{kj} > 0$ 
          outbid( $i, j, k$ )
        else
          raise_price( $j$ )
           $\forall k : D_k = \arg \max_j v_{kj} / p_j ; \alpha_k = \max_j v_{kj} / p_j$ 
        endif
      end while
    end for
  until  $\sum_i r_i > \epsilon (\sum_i e_i)$ 
end algorithm auction

procedure outbid( $i, j, k$ )
  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

procedure raise_price( $j$ )
   $\forall i : y_{ij} = h_{ij} ; h_{ij} = 0;$ 
   $p_j = (1 + \epsilon)p_j$ 
end procedure raise_price

```

Fig. 2. The basic auction mechanism

Define a directed bipartite graph $G = (B, S, D \cup Z)$ where B is the set of buyers, S the set of goods.

Lemma 6. *The graph G is acyclic.*

The proof can be carried on same lines as [11].

Note that procedure `outbid` transfers the surplus (unspent money) from one buyer to another only along paths in G . Moreover the surplus reduces by a factor $(1 + \epsilon)$ every time it travels from one node to another in G . When `outbid`(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 `outbid` as complete when an edge in Y is removed and incomplete if r_i goes to zero.

The steps performed by the algorithm `auction` are classified into three types (a) price rise of an item; (b) complete call to `outbid` (y_{kj} limits the bid); (c) incomplete call (surplus r_i goes to zero) to `outbid` and (d) computation of α_i and D_i for all i . The time complexity of the algorithm can be bounded as follows: steps of type (a) take $O(n)$ time; there can be atmost n steps of type (b) for every step of type (a). Using the directed acyclic graph argument of [11] it can be shown that there can be atmost n^2 steps of type (c) for every step of type (a). Steps of type (d) can be naively implemented in $O(nm)$ time and there can be atmost one step of type (d) for every step of type (a).

This gives a time complexity of $O((n^2 + nm)W)$ where W is the number of steps of type (a) (i.e., the number of price rises).

We now bound W . Let $e_{min} = \min_i e_i$, $e = \sum_i e_i$, $v_{min} = \min_{i,j:v_{ij}>0} v_{ij}$, $v_{max} = \max_{i,j} v_{ij}$, $a_{max} = \max_j a_j$ and $a_{min} = \min_j a_j$. The maximum and the minimum price of any item is bounded, respectively, by $\frac{e}{a_{min}}$ and $\frac{(e_{min}v_{min})}{(a_{max}v_{max})}$. Therefore, the number of price raise for any item in the first iteration is bounded by $O(\log((ev_{max}a_{max})/(e_{min}v_{min}a_{min})))$. Using Theorem 2 and the fact that prices are rolled back by factor $(1 + \epsilon_k)^{2n}$ after iteration k and $\epsilon_{k+1} = \epsilon_k/2$, the number of price raises for any item in one call to algorithm `auction` at iterations subsequent to the first, can be bounded by $7n$. This gives a bound of $O(nm + m \log((ev_{max}a_{max})/(e_{min}v_{min}a_{min})) \log(1/\delta))$ for w . Therefore, we have the following time complexity of algorithm `roll-back`.

Theorem 4. *Algorithm `roll-back` terminates in $O(nm(n + m)(n + \log((ev_{max}a_{max})/(e_{min}v_{min}a_{min}))) \log(1/\delta))$ steps.*

A bound on $1/\delta$ can be provided when the input is rational with numbers bounded by M . By an analysis similar to that of Lemma 8 in [5], it can be shown that price of an item j , p_j is related to p_k , price of k , in the connected component of $G_o = (B, S, E)$ where E comprises edges (i, j) s.t. $x_{ij}^* > 0$ in the equilibrium allocation. In fact $p_j = \frac{a}{b} p_k$ where a and b are product of utility values of length l when the items j and k are connected by a path of length $2l$. Thus $1/\delta$ is bounded by nV^n where $V = \max_{i,j}\{v_{ij}\}$. We can thus obtain an exact algorithm for the market clearing problem.

5 A Faster Algorithm Using Path-Auctions

The market equilibrium problem (for Fisher's model with linear utilities) is strikingly similar to the maximum flow problem and the auction algorithm described in the previous section resembles the preflow push algorithms for the maximum flow problem. The procedure `outbid` may be viewed as a step where the surplus (excess flow) is transferred from one buyer (vertex) to another. However, there are two key differences between the maximum flow algorithms and the `auction` algorithm. Unlike the traditional flows, the surplus is not conserved; it decreases

by a factor $(1 + \epsilon)$ when transferred from one vertex to another. Secondly, the graph G on which the algorithm works changes dynamically, due to the changes in demand sets and assignments.

Consider the graph $G = (B, S, D \cup Z)$. With each edge is associated a capacity which represents the maximum amount of money that can be pushed along the edge. For an edge in D there is no bound whereas for an edge (u, v) in Z the capacity is bounded by $y_{vu} * p_u$. Consider a path $(u_1, v_1, u_2, v_2, \dots, u_k, v_k)$ in G , beginning at a buyer u_1 with positive surplus and ending at an item v_k that has no outgoing edges. Let the prices of these items be p_1, p_2, \dots, p_k respectively. Consider a bidding sequence where u_1 outbids u_2 on item v_1 , who in turn outbids u_3 on item v_2 , and so on, till u_{k-1} outbids u_k on item v_{k-1} . To acquire x units of item v_1 , u_1 needs $x p_1$ amount of money. This releases $p_1 x / (1 + \epsilon)$ surplus for u_2 , which if spent fully on acquiring v_2 , will generate a surplus of $p_1 x / (1 + \epsilon)^2$ at u_3 . If such a bidding is carried till the end of the path, the amount of surplus generated at u_k will be $p_1 x / (1 + \epsilon)^{k-1}$. Since v_k has no outgoing edge, all the buyers of v_k have v_k in their demand sets. If these buyers are outbid by u_k then they can reduce their allocations of item v_k by a factor $(1 + \epsilon)$ and switch from the lower price $(p_k / (1 + \epsilon))$ to the higher price (p_k) . Therefore the maximum amount u_k can bid on v_k is given by $\frac{\epsilon}{1 + \epsilon} \sum_{w \in B} y_{wk} p_k$. To model this, we add a special vertex t (called the sink) to G . For every vertex v which does not have an outgoing edge, we add an edge (v, t) of capacity $\frac{\epsilon}{1 + \epsilon} \sum_{w \in B} y_{wv} p_v$.

Path auctions are defined by such sequences of bidding along paths in G . An auction where bidding is done along paths in G such the surplus of all the nodes except the first node remains unchanged, is called a *path auction*.

We define the capacity of an auction path as the maximum amount of money that the first buyer can bid along the path, without changing the surplus of other buyers or the price of any item on the path. For the path $(u_1, v_1, \dots, u_k, v_k)$ the capacity is equal to $\min(\min_{1 \leq j \leq k} y_{(j+1)j} (1 + \epsilon)^{j-1} p_j, \epsilon (1 + \epsilon)^{k-2} \sum_{w \in B} y_{wk} p_k)$.

In order to compute the path capacities and carry out bidding on paths efficiently, we use *multiplier* flows which are used to model the fact that the surplus is not conserved as it traverses a path in the graph G . We then define operations on dynamic trees [18] with multiplier flows. We use these operations to efficiently implement the algorithm *path auctions*. We finally outline how the dynamic trees may be modified to support multiplier flows efficiently.

Multiplier flows

Given a directed graph $G = (V, E)$ with edge capacities $c : E \rightarrow R_+$ and multipliers $\eta : E \rightarrow R_+$. The multiplier of a path $P = (e_1, e_2, \dots, e_k)$ is defined as $\eta(P) = \prod_{i=1}^k \eta(e_i)$. A multiplier flow of value f through a directed path P carries a flow of value $\prod_{j=i}^k \eta(e_j) f$ through the edge e_i . Note that if the multiplier $\eta(e_j) = 1 + \epsilon$, for all j then a multiplier flow of value f on the path P translates into a flow of value $(1 + \epsilon)^{(k-j)} f$ through the edge e_j for all $j \leq k$. We define the *multiplier capacity* of the path P as $\min_{1 \leq i \leq k} c(e_i) / \eta(e_i, \dots, e_k)$. Note that the capacity of an auction path is equal to the product of the multiplier of the path and its multiplier capacity.

Dynamic trees, multiplier flows and efficient path-auctions

An efficient data structure to maintain dynamic trees was proposed in [18]. This data structure maintains a collection of vertex disjoint trees to efficiently (in $O(\log n)$ amortized time) carry out operations of combining/splitting the trees and updating capacity on paths from leaves to root of a tree. We adapt this data structure for multiplier flows by defining operations on trees.

We need the following definitions:

$\text{link}(u, v, c, \eta)$: Join the tree rooted at u to the tree node v by adding the edge (u, v) of capacity c and multiplier η .
 $\text{cut}(u, v)$: Split the tree containing the (u, v) edge into two trees by removing the edge.
 $\text{parent}(u)$: Returns the parent of u in the tree (nil if u is a root node).
 $\text{root}(u)$: Returns the root of tree containing the vertex u .
 $\text{children}(u)$: Returns the set of children of node u .
 $\text{capacity}(u, v)$: Returns the capacity of the edge (u, v) .
 $\text{multiplier}(u)$: Returns the multiplier of the path from u to $\text{root}(u)$.
 $\text{find-min}(u)$: Let V be the set of edges in the path from u to $\text{root}(u)$. This function returns $\arg \min_{v \in V} \text{capacity}(v, \text{parent}(v)) / \text{multiplier}(v)$.
 $\text{update}(u, x)$: Let V be the set of edges in the path from u to $\text{root}(u)$. This function updates the capacities for all the edges in the path as $\forall v \in V : \text{capacity}(v, \text{parent}(v)) += x \text{ multiplier}(v)$

The algorithm discovers a path from a vertex u with positive surplus ($r_u > 0$) to the sink t . It then finds out the maximum amount that may be bid along the path using its multiplier capacity. After the bidding there are three possibilities (as in algorithm `auction`) (a) price of last item needs to be raised; (b) y_{uv} goes to zero for some edge in the path, or (c) the surplus r_u of vertex u goes to zero. In case (c) the algorithm moves to another buyer with positive surplus. In case (b) the corresponding edge is removed from the graph G and another path from v to t is found. In case (a) the price of the item is raised and the required data-structures and variables updated suitably. We now present the algorithm in greater detail.

The algorithm first creates the graph G using the initial solution. The set D is maintained in the graph by keeping a heap of items for all $u \in B$, sorted by v_{uv}/p_v . The sets Z and Y are maintained by keeping two lists for each item $v \in S$. We add a special vertex t called as the sink in the graph G . All the vertices $v \in S$ which do not have an outgoing edge (in Z) are implicitly assumed to be connected to t . Initially all the trees are singleton vertices.

The algorithm picks a buyer vertex u with positive surplus and creates a path from u to the sink t by linking the vertices using the edges in G . For every buyer vertex in G , the demand set is well defined and hence there is always an outgoing edge from the vertex. For an item vertex v in G either there is an outgoing edge in Z or it is connected to t . Since the graph G is acyclic, a path to t can always be discovered.

For the edges in D that get linked in the process, the capacity is set to M and multiplier is set to 1. For the edges (v, u) in Z , the capacity is set to $\hat{y}_{uv}p_v$ and the multiplier is set to $(1 + \epsilon)$. For edges of the form (v, t) the capacity is set to $\frac{\epsilon}{1+\epsilon} \sum_{u \in B} \hat{y}_{uv}p_v$ and the multiplier is set to 1. Note that the capacity of auction path (u, \dots, t) is equal to $\text{multiplier}(u) \text{ find_min}(u)$.

For an item v such that $(v, t) \in G$, define $\gamma_v = (1 + \epsilon)\text{capacity}(v, t)/(\epsilon \sum_{u \in B} \hat{y}_{uv}p_v)$. For all other items v , define $\gamma_v = 0$. Now define the assignments h_{uv} and y_{uv} using the variables \hat{h}_{uv} and \hat{y}_{uv} and the capacities of edges in trees as follows: if $(v, u) \in Z$ is in a tree, then $y_{uv} = \text{capacity}(v, u)$ else $y_{uv} = \hat{y}_{uv}(1 - \gamma_v)$. If $(u, v) \in D$ is in a tree then $h_{uv} = (M - \text{capacity}(u, v))/p_v + \gamma_v \hat{y}_{uv}/(1 + \epsilon)$ else $h_{uv} = \gamma_v \hat{y}_{uv}/(1 + \epsilon)$.

With these definitions, it can be verified that the update step in the algorithm is indeed equivalent to a path auction from buyer u on the path to t . After the update step, the algorithm either raises the price of the last item in the path, or cuts an edge from the tree or moves to another buyer with positive surplus. In each of these cases, the variables and data structures are updated suitably.

Efficient implementation of dynamic trees

We next show how to modify the dynamic tree data structure of Sleator and Tarjan [18] to implement the requirements of pushing multiplier flows.

The collection of auction paths form a collection of vertex-disjoint trees and change over time. The dynamic tree data structure[18] applies in this context. Each path is represented by binary trees. Each node, v , of the binary tree data structure represents a sub-path $P(v)$, i.e. a sequence of edges corresponding to the edges stored at the leaves of the subtree rooted at v . At each node v we maintain a variable corresponding to the minimum capacity of the flow path, $Min(v)$, a variable representing a composite multiplier for the path P termed $Mult(v)$, and updates which are applicable to each edge of the path $P(v)$, $UP(v)$. The effective minimum capacity of the path corresponding to a node x in the tree is obtained as $EMin(x) = Min(x) - \sum_{y \in Y} UP(y) * Mult(y) / Mult(x)$, where Y is the set of nodes on the path from node (x) to the root. The second term denote the effective update variable at node x . The minimum value at a node v is effectively $Min(v) = \min(Min(u) - Up(v), (Min(w) - Up(w)) * Mult(u))$, where u and w are the left and right children of the node v , respectively. Further, $Mult(v) = Mult(u) * Mult(w)$.

If the tree data structure used is represented by splay trees [19], nodes along the path from the root to a particular node, say v , in the tree are affected. Let the path be $P(v)$. To implement the changes along the path, say at a node $y \in P(v)$, the effective update variable is computed at each child for node y , say w and z and $UP(w), UP(z)$ computed at these nodes. $UP(y)$ is set to zero. This makes the value of the update variable zero along the path and path changes can be now be made locally. Following [19], this results in an implementation of a sequence of k tree operation involving multiplier flows requiring $O(k \log n)$ operations where n are the maximum number of nodes in the splay trees.

Complexity of Path-Auctions

To bound the complexity of the algorithm we carry out amortized analysis of different operations. If W is the number of times prices are raised, then the number of dynamic tree operations are bounded by $O(nW)$. Each of the dynamic tree operation can be implemented in $O(\log n)$ amortized time. The time required to update the data structures is $O(n)$. This gives a overall time complexity of $O(nW \log n)$.

If we use the roll-back mechanism, W is bounded by $O(nm + m \log((ev_{max}a_{max})/(e_{min}v_{min}a_{min})) \log(1/\delta))$, where $1/\delta$ is bounded by nV^n and $V = \max_{ij} \{v_{ij}\}$ Thus we have the following result.

Theorem 5. *Using path-auctions the market equilibrium problem can be solved exactly in $O(n(nm + m \log((ev_{max}a_{max})/(e_{min}v_{min}a_{min})) \log(1/\delta)) \log n)$.*

6 Reduction from Max-flows

In order to show the inherent complexity of the market equilibrium problem, we reduce the problem of maximum flows with vertex capacities to the market equilibrium problem. Given a graph $G = (V, E)$ with vertex capacities $c : V \rightarrow R_+$, two special vertices s, t (called the source and the sink) and a flow value f the problem is to decide if it is possible to route f units of flow from s to t without sending a flow of value more than $c(v)$ from any vertex v . This problem can be reduced to the following market equilibrium.

For each vertex in $V - \{t\}$ create a buyer in B and for each vertex in $V - \{s\}$, create an item in S . For every vertex $j \in V - \{s, t\}$, let $e_j = a_j = c(j)$. Let $v_{ij} = 1$

if $(i, j) \in E$ and $v_{ij} = 0$ otherwise. Let $v_{ii} = 1$ for all $i \in V - \{s, t, \}$. Let $e_s = f$ and $a_t = f$. Note that the equilibrium prices are unique in the Fisher model. Therefore, it is easy to verify that a flow f can be routed from s to t in G iff all the equilibrium prices in the corresponding market equilibrium problem are unity.

References

1. K. Arrow and G. Debreu. Existence of an Equilibrium for a Competitive Economy. *Econometrica*, 22:265–290, 1954.
2. K.J. Arrow, H.D. Black, and L. Hurwicz. On the Stability of the Competitive Equilibrium, II. *Econometrica*, 27:82–109, 1959.
3. W. C. Brainard and H. E. Scarf. How to Compute Equilibrium Prices in 1891. Cowles Foundation Discussion Paper (1272), 2000.
4. Bruno Codenotti, Sriram Pemmaraju, and Kasturi Varadarajan. A note on the Computation of Equilibria in Exchange Markets with Gross Substitutibility. Preprint, 2004.
5. 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, November 2002.
6. 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.
7. N. R. Devanur and V.V. Vazirani. The Spending Constraint Model for Market Equilibrium: Algorithmic, Existence and Uniqueness Results. In *Proceedings of the 36th Annual ACM Symposium on the Theory of Computing*, 2004.
8. B. Eaves. A Finite Algorithm for the Linear Exchange Model. *Journal of Mathematical Economics*, 3:197–203, 1976.
9. E. Eisenberg. Aggregation of utility functions. *Management Sciences*, 7(4):337–350, 1961.
10. E. Eisenberg and D. Gale. Consensus of Subjective Probabilities: The Pari-Mutuel Method. *Annals of Mathematical Statistics*, 30:165–168, 1959.
11. Rahul Garg and Sanjiv Kapoor. Auction Algorithms for Market Equilibrium. In *Proceedings of the 36th Annual ACM Symposium on the Theory of Computing*, 2004.
12. Rahul Garg, Sanjiv Kapoor, and Vijay Vazirani. An Auction-Based Market Equilibrium Algorithm for the Separable Gross Substitutibility Case. In *Proceedings of the 7th. International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'04)*, 2004.
13. K. Jain, M. Mahdian, and A. Saberi. Approximating Market Equilibrium. In *Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX 2003)*, 2003.
14. Kamal Jain. A Polynomial Time Algorithm for Computing the Arrow-Debreu Market equilibrium for Linear Utilities. Preprint.
15. E.I. Nenakov and M.E. Primak. One algorithm for finding solutions of the Arrow-Debreu model. *Kibernetika*, 3:127–128, 1983.
16. D.J. Newman and M.E. Primak. Complexity of Circumscribed and Inscribed Ellipsoid Methods for Solving Equilibrium Economical Models. *Applied Mathematics and Computations*, 52:223–231, 1992.
17. M.E. Primak. A Converging Algorithm for a Linear Exchange Model. *Applied Mathematics and Computations*, 52:223–231, 1992.
18. Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.
19. Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985.
20. V. V. Vazirani. Market Equilibrium When Buyers Have Spending Constraints . Submitted, 2003.
21. Yinyu Ye. A Path to the Arrow-Debreu Competitive Market Equilibrium. Preprint, 2004.