Improved Algorithms for Computing Fisher's Market Clearing Prices

James B. Orlin *
MIT Sloan School of Management
email: jorlin@mit.edu

April 27, 2010

Abstract

We give the first strongly polynomial time algorithm for computing an equilibrium for the linear utilities case of Fisher's market model. We consider a problem with a set B of buyers and a set G of divisible goods. Each buyer i starts with an initial integral allocation e_i of money. The integral utility for buyer i of good j is U_{ij}. We first develop a weakly polynomial time algorithm that runs in $O(n^4 \log U_{max} + n^3 e_{max})$ time, where $n = |B| + |G|$. We further modify the algorithm so that it runs in $O(n^4 \log n)$ time. These algorithms improve upon the previous best running time of $O(n^8 \log U_{max} + n^7 \log e_{max})$, due to Devanur et al. [5]

1 Introduction

In 1891, Irving Fisher (see [3]) proposed a simple model of an economic market in which a set B of buyers with specified amounts of money were to purchase a collection G of diverse goods. In this model, each buyer i starts with an initial allocation e_i of money, and has a utility U_{ij} for purchasing all of good j. The utility received is assumed to be proportional to the fraction of the good purchased.

In 1954, Arrow and Debreu [2] provided a proof that a wide range of economic markets (including Fisher's model) have unique market clearing prices, that is, prices at which the total supply is equal to the total demand. However, their proof relied on Kakutani's fixed point theorem, and did not efficiently construct the market clearing prices. Eisenberg and Gale [8] transformed the problem of computing the Fisher market equilibrium into a concave cost maximization problem. They deserve credit for the first polynomial time algorithm because their concave maximization problem is solvable in polynomial time using the ellipsoid algorithm.

Devanur et al [5] gave the first combinatorial polynomial time algorithm for computing the exact market clearing prices under the assumption that utilities for goods are linear. For a problem with a total of n buyers and goods, their algorithm runs in $O(n^8 \log U_{max} + n^7 \log e_{max})$ time, where U_{max} is the largest utility and e_{max} is the largest initial amount of money of a buyer, and where all data are assumed to be integral. More precisely, it determines the market clearing prices

*Research supported in part by ONR grant N00014-05-1-0165

Dagstuhl Seminar Proceedings 10171
Equilibrium Computation
http://drops.dagstuhl.de/opus/volltexte/2010/2672

as a sequence of $O(n^5 \log U_{max} + n^4 \log e_{max})$ maximum flow problems. (For details on solution techniques for maximum flows, see Ahuja et al. [1]).

Here we provide an algorithm that computes a market equilibrium in $O(n^4 \log U_{max} + n^3 \log e_{max})$ time. We also show how to modify the algorithm so that the running time is $O(n^4 \log n)$, resulting in the first strongly polynomial time algorithm for computing market clearing prices for the Fisher linear case.

The running time can be further improved to $O((n^2 \log n)(m + n \log n))$, where m is the number of pairs (i, j) for which $U_{ij} > 0$. Moreover, the algorithm can be used to provide an ϵ-approximate solution in $O((n \log 1/\epsilon)(m + n \log n))$ time, improving upon the running time of the algorithm by Garg and Kapoor [10] while using a stricter definition of ϵ-approximation.

1.1 A formal description of the model

Suppose that p is a vector of prices, where p_j is the price of good j. Suppose that x is an allocation of money to goods; that is, x_{ij} is the amount of money that buyer i spends on good j. The *surplus cash* of buyer i is $c_i(x) = e_i - \sum_{j \in G} x_{ij}$. The *backorder amount* of good j is $b_j(p, x) = -p_j + \sum_{i \in B} x_{ij}$.

We assume that for each buyer i, there is a good j such that $U_{ij} > 0$. Otherwise, we would eliminate the buyer from the problem. Similarly, for each good j, we assume that there is a buyer i with $U_{ij} > 0$. Otherwise, we would eliminate the good from the problem.

For a given price vector p, the ratio U_{ij}/p_j is called the *bang-per-buck* of (i, j) with respect to p. For all $i \in B$, we let $\alpha_i(p) = \max\{U_{ij}/p_j : j \in G\}$, and refer to this ratio as the *maximum bang-per-buck* for buyer i. A pair (i, j) is said to be an *equality edge* with respect to p if $U_{ij}/p_j = \alpha_i(p)$. We let $E(p)$ denote the set of equality edges.

A solution refers to any pair (p, x) of prices and allocations. We say that the solution is *optimal* if the following constraints are all satisfied:

i. *Cash constraints*: For each $i \in B$, $c_i(x) = 0$.

ii. *Allocation of goods constraints*: For each $j \in G$, $b_j(p, x) = 0$.

iii. *Bang-per-buck constraints*: For all $i \in B$ and $j \in G$, if $x_{ij} > 0$, then $(i, j) \in E(p)$.

iv. *Non-negativity constraints*: $x_{ij} \geq 0, p_j \geq 0$ for all $i \in B$ and $j \in G$.

The above optimality conditions are the market equilibrium conditions specified by Fisher, as well as the optimality conditions for the Eisenberg-Gale program [8].

1.2 Other algorithmic results

The computation of market clearing prices is an important aspect of general equilibrium theory. Nevertheless, as pointed out in [5], there have been few results concerning the computation of market equilibrium. We refer the reader to [5] for these references. Papers that are particular relevant to this paper include papers by Garg and Karpur [10], Ghiyasvand and Orlin [11], and Vazirani [14]. Garg and Kapoor presents a fully polynomial time approximation scheme (FPTAS) for computing approximate market clearing prices for the Fisher linear case. Ghiyasvand and

Orlin develop a faster FPTAS while using a stronger (less relaxed) definition of approximate market equilibrium. Vazirani develops a weakly polynomial time algorithm for a generalization of the Fisher linear case in which utilities are piecewise linear functions of the amounts spent, and in which there is a utility of saving money. This more general utility function permits the modeling of several types of constraints, including upper bounds on the amount spent by buyer i in the purchase of good j.

1.3 Contributions of this paper

The primary contributions of this paper are a strongly polynomial time algorithm for computing the market clearing prices, as well as a simple and efficient weakly polynomial time algorithm. Additional contributions of this paper include the following:

i. An algorithm for obtaining an ϵ-approximate solution that is faster than the algorithm of Garg and Kapoor [10], while simultaneously having a stricter definition of ϵ-approximation.

ii. Our weakly polynomial and strongly polynomial time algorithms terminate after an optimal set of edges is identified, which may be much sooner than the termination criterion given by Devanur et al. [5].

iii. Our running time analysis does not rely on the sum-of-squares potential function used in Devanur et al.

1.4 Overview of the weakly polynomial time algorithm

The algorithm first determines an initial vector p^0 of prices (as described in Section 3.1). During the algorithm, prices are modified in a piecewise linear manner (as per Devanur et al. [5]) so that prices are always non-decreasing. The algorithm relies on a parameter Δ, called the *scaling parameter*. We say that a solution (p, x) is Δ-*feasible* if it satisfies the following conditions:

i. $\forall i \in B, c_i(x) \geq 0$;

ii. $\forall j \in G$, if $p_j > p_j^0$, then $0 \leq b_j(p, x) \leq \Delta$;

iii. $\forall i \in B$ and $\forall j \in G$, if $x_{ij} > 0$, then $(i, j) \in E(p)$, and x_{ij} is a multiple of Δ;

iv. $\forall i \in B$ and $\forall j \in G, x_{ij} \geq 0$ and $p_j \geq 0$.

We say that a solution (p, x) is Δ-*optimal* if it is Δ-feasible and if in addition

v. $\forall i \in B$, $c_i(x) < \Delta$.

The Δ-feasibility conditions modify the optimality conditions as follows: buyers may spend less than their initial allocation of money; it is possible for none of good j to be allocated if $p_j = p_j^0$; more than 100% of a good may be sold; and allocations are required to be multiples of Δ.

Our algorithm is a "scaling algorithm" as per Edmonds and Karp [7]. It initially creates a Δ-feasible solution for $\Delta = e_{max}$. It then runs a sequence of scaling phases. The input for the Δ-scaling phase is a Δ-feasible solution (p, x). The Δ-scaling phase transforms (p, x) into a Δ-optimal

solution (p', x') via a procedure PriceAndAugment, which is called iteratively. The algorithm then transforms (p', x') into a $\Delta/2$-feasible solution. Then Δ is replaced by $\Delta/2$, and the algorithm goes to the next scaling phase. One can show that within $O(n \log U_{max} + e_{max})$ scaling phases, when the scaling parameter is less than $8n^{-2}(U_{max})^{-n}$, the algorithm determines an optimal solution.

1.5 Overview of the strongly polynomial time algorithm

If $x_{ij} \geq 3n\Delta$ at the beginning of the Δ-scaling phase, then $x'_{ij} \geq 3n(\Delta/2)$ for the solution x' at the beginning of the $\Delta/2$-scaling phase. We refer to edges (i, j) with $x_{ij} \geq 3n\Delta$ as "abundant." Once an edge becomes abundant, it remains abundant, and it is guaranteed to be positive in the optimal solution to which the scaling algorithm converges.

If a solution is "Δ-fertile" (as described in Section 4) at the end of the Δ-scaling phase, then there will be a new abundant edge within the next $O(\log n)$ scaling phases. If the solution (p, x) is not Δ-fertile, then the algorithm calls a special procedure that transforms (p, x) into a new solution (p', x') that is Δ'-fertile and Δ'-feasible for some $\Delta' \leq \Delta/n^2$. Since there are at most n abundant edges, the number of scaling phases is $O(n \log n)$ over all iterations.

2 Preliminaries

We next state how to perturb the data so as to guarantee a unique optimal allocation. We then establish some results concerning feasible and optimal solutions.

For a feasible price vector p, it is possible that the set $E(p)$ of equality edges has cycles. If p is the optimal vector of prices, this leads to the possibility of multiple optimal solutions.

Here we perturb the utilities as follows: for all pairs i, j such that $U_{ij} > 0$, we replace U_{ij} by $U_{ij} + \epsilon^{in} + \epsilon^{j}$, where ϵ is chosen arbitrarily close to 0. One can use lexicography to simulate perturbations without any increase in worst case running time. The advantage of perturbations is that it ensures that the set of equality edges is always cycle-free. Henceforth, we assume that the utilities have been perturbed, and the algorithm uses lexicography.

If E is the equality graph prior to perturbations, and if E' is the equality graph after perturbations, then E' is the maximum weight forest of E obtained by assigning edge (i, j) a weight of $in + j$. (This can be seen if one observes that (i, j) is an equality edge if $\log U_{ij} - \log p_j = \log \alpha_i$).

In the case that prices are changed and more than one edge is eligible to become an equality edge (prior to the perturbation), then the algorithm will select the edge with maximum weight for the perturbed problem. Henceforth, we assume that the utilities are perturbed, and we implement the algorithms using weights as a tie-breaking rule.

Suppose that $H \subseteq B \times G$. If H is cycle free, then *BasicSolution*(H) is the unique vector p satisfying the following:

i. If $(i, j) \in H$ and if $(i, k) \in H$, then $U_{ij}/p_j = U_{ik}/p_k$.

ii. For each connected component C of H, $\sum_{i \in C} e_i = \sum_{j \in C} p_j$.

iii. If $(i, j) \notin H$, then $x_{ij} = 0$.

iv. For all $i \in B$, $\sum_{j \in G} x_{ij} = e_i$.

v. For all $j \in G$, $\sum_{i \in B} x_{ij} = p_j$.

The first two sets of constraints uniquely determine the vector p. The latter three sets of constraints uniquely determine the allocation vector x. The following lemma implies that an optimal solution can be obtained from its optimal set of edges.

Lemma 2.1 *Suppose that (p^*, x^*) is the optimal solution for the Fisher Model. Let $H^* = \{(i,j) : x^*_{ij} > 0\}$. Then BasicSolution$(H^*) = (p^*, x^*)$. Moreover, if $x^*_{ij} > 0$, then $x^*_{ij} > e_{min}/D$, where $D = n(U_{max})^n$.*

Proof . We first observe that H^* is cycle free because the perturbation ensures that there is a unique optimal solution. Suppose that $(\hat{p}, \hat{x}) =$ BasicSolution(H^*). The optimal price vector satisfies the $|B|$ constraints given in the first two sets of constraints for BasicSolution(H^*). Since these constraints are linearly independent, $p^* = \hat{p}$.

We next consider the remaining three sets of constraints that determines $\hat{x}$. For each component C of H^*, there are $|C|$ variables and $|C| + 1$ constraints, one of which is redundant because the sum of the moneys equals the sum of the prices. These constraints are all satisfied by x^*. Since there are $|H^*|$ variables that are not required to be 0, and there are $|H^*|$ linearly independent constraints, it follows that $x^* = \hat{x}$.

Any solution satisfying the above systems of equations is a rational number with denominator less than D by Cramer's rule. Moreover, if $j \in G$ is in component C, then the numerator of the rational number representing p_j is a multiple of $\sum_{i \in B} e_i$, which is at least e_{min}. ■

In the following, the use of a set as a subscript indicates summation. For example, $e_S = \sum_{i \in S} e_i$. The following lemma is elementary and is stated without proof.

Lemma 2.2 *Suppose that (p, x) is any solution and that $B' \subseteq B$, and $G' \subseteq G'$. Then $e_B - p_G = c_B(x) + b_G(p, x)$.*

3 The Δ-scaling algorithm

3.1 The initial solution

We initialize as follows. Let $\Delta^0 = e_{max}/n$; $\forall i \in B$, let $U_{iG} = \sum_{j \in G} U_{ij}$; $\forall i \in B$ and $\forall j \in G$, let $\rho_{ij} = U_{ij}e_i/nU_{iG}$; $\forall j \in G$, let $p^0_j = \max\{\rho_{ij} : i \in B\}$; $\forall i \in B$ and $\forall j \in G$, $x^0_{ij} = 0$.

We note that the initial solution (p^0, x^0) is Δ^0-feasible. Moreover, it is easily transformed into a feasible solution by setting $x_{ij} = p_j$ for some i with $(i,j) \in E(p^0)$.

3.2 The residual network and price changes

Suppose that (p, x) is a Δ-feasible solution at some iteration during the Δ-scaling phase. We define the residual network $N(p, x)$ as follows: the node set is $B \cup G$. For each equality edge (i, j), there is an arc $(i, j) \in N(p, x)$. We refer to these arcs as the *forward arcs* of $N(p, x)$. For every (i, j) with $x_{ij} > 0$, there is an arc $(j, i) \in N(p, x)$. We refer to these arcs as the *backward arcs* of $N(p, x)$.

Let r be a *root node* of the residual network $N(p,x)$. The algorithm will select $r \in B$ with $c_r(x) \geq \Delta$. We let ActiveSet(p,x,r) be the set of nodes $k \in B \cup G$ such that there is a directed path in $N(p,x)$ from r to k; if $k \in$ ActiveSet(p,x,r), we say that k is *active* with respect to p,x and r.

As per price changes in [5], we will replace the price p_j of each active good j by $q \times p_j$ for some $q > 1$. This is accomplished by the following formula: $f(q) =$ Price(p,x,r,q), where

$$f_j(q) = \begin{cases} qp_j & \text{if } j \in \text{ActiveSet}(p,x,r) \\ p_j & \text{otherwise,} \end{cases}$$

and where q is below a certain threshold.

UpdatePrice(p,x,r) is the vector p' of prices obtained by setting $p' =$ Price(p,x,r,q'), where q' is the maximum value of q such that (p',x) is Δ-feasible. At least one of two conditions will be satisfied by p': (i) there is an edge $(i,j) \in E(p') \backslash E(p)$, at which point node j becomes active; or (ii) there is an active node j with $b_j(p',x) \leq 0$. In the former case, PriceAndAugment will continue to update prices. In the latter case, it will carry out an "augmentation" from node r to node j. The following lemma is straightforward and is stated without proof.

Lemma 3.1 *Suppose that (p, x) is Δ-feasible and p' is the vector of prices obtained by* UpdatePrice(p,x,r). *Then (p',x) is Δ-feasible.*

3.3 Augmenting paths and changes of allocation

Suppose that (p,x) is a Δ-feasible solution. Any path $P \subseteq N(p,x)$ from a node in B to a node in G is called an *augmenting path*. A Δ-*augmentation* along the path P consists of replacing x by a vector x' where

$$x'_{ij} = \begin{cases} x_{ij} + \Delta & \text{if } (i,j) \in P \text{ is a forward arc of } N(p,x); \\ x_{ij} - \Delta & \text{if } (i,j) \in P \text{ is a backward arc of } N(p,x); \\ x_{ij} & \text{otherwise.} \end{cases}$$

Lemma 3.2 *Suppose that (p,x) is Δ-feasible and that $c_r \geq \Delta$. Suppose further that P is an augmenting path from node r to a node $k \in G$ for which $b_j(p,x) \leq 0$. If x' is obtained by a Δ-augmentation along path P, then (p,x') is Δ-feasible. Moreover, $c_r(x') = c_r(x) - \Delta$, and $c_j(x') = c_j(x)$ for $j \neq r$.*

We present the procedures PriceAndAugment and ScalingAlgorithm next. We will establish their proof of correctness and running time in Subsection 3.4.

Algorithm 1. PriceAndAugment(p,x)
begin
  select a buyer $r \in B$ with $c_r(x) \geq \Delta$;
  compute ActiveSet(p,x,r);
  until there is an active node j with $b_j(p,x) \leq 0$ do
    replace p by UpdatePrice(p,x,r);
    recompute $N(p,x), b(p,x)$, and ActiveSet(p,x,r);

enduntil
let P be a path in $N(p,x)$ from r to an active node j with $b_j(p,x) \leq 0$;
replace x by carrying out a Δ-augmentation along P;
recompute $c(x)$ and $b(p,x)$;
end

Algorithm 2. ScalingAlgorithm(e,U)
begin
$\Delta := \Delta^0$; $p := p^0$; $x := 0$ (as per Section 3.1);
ScalingPhase:
while (p,x) is not Δ-optimal, replace (p,x) by PriceAndAugment(p,x);
$E' := \{(i,j) : x_{ij} \geq 4n\Delta\}$;
if BasicSolution(E') is optimal, then quit; else continue;
$\Delta := \Delta/2$;
for each $j \in G$ such that $b_j(p,x) > \Delta$, decrease x_{ij} by Δ for some $i \in B$;
return to ScalingPhase;
end

3.4 Running time analysis for each scaling phase and proof of correctness

Our bound on the number of calls of PriceAndAugment during a scaling phase will rely on a potential function argument. Let $\Phi(x,\Delta) = \sum_{i \in B} \lfloor c_i(x)/\Delta \rfloor$. We will show that $\Phi \leq n$ at the beginning of a scaling phase, $\Phi = 0$ at the end of a scaling phase, and each call of PriceAndAugment decreases Φ by exactly 1.

At the Δ-scaling phase, ScalingAlgorithm selects a buyer r with $c_r(x) \geq \Delta$. Ultimately it performs a Δ-augmentation. It continues the Δ-scaling phase until it obtains a Δ-optimal solution. Each Δ-augmentation reduces c_B by Δ. At the end of the scaling phase, ScalingAlgorithm does a check for optimality. If an optimal solution is not found, it divides Δ by 2, obtains a $\Delta/2$-optimal solution and goes to the next scaling phase.

Theorem 3.1 *During the Δ-scaling phase,* ScalingAlgorithm *transforms a Δ-feasible solution into a Δ-optimal solution with at most n calls of* PriceAndAugment*. Each call of* PriceAndAugment *can be implemented to run in $O(m + n \log n)$ time. The number of scaling phases is $O(e_{max} + n \log U_{max})$, and the algorithm runs in $O(n(m + n \log n)(e_{max} + n \log U_{max}))$ time.*

We will divide the proof of Theorem 3.1 into several parts. But first we deal with an important subtlety. We permit a good j in G to have no allocation if $p_j = p_j^0$. We need to establish that node j will receive some allocation eventually, and that $b_j(p,x) \geq 0$ at that time.

Lemma 3.3 *Suppose that $\Delta < p_j^0/n$, and that that (p,x) is a Δ-feasible solution at the end of the Δ-scaling phase. Then $b_j(p,x) \geq 0$.*

Proof. We prove the contrapositive. Suppose that $b_j(p,x) < 0$. By Property (ii) of the definition of Δ-feasibility, $p_j = p_j^0$. Select node $i \in B$ such that $(i,j) \in E(p^0)$. We will show that $c_i(x) \geq \Delta$, which is a contradiction.

By construction, $p_j^0 \leq e_i/n$, and so $e_i/n \geq n\Delta$. Note that for each $k \in G$,

$$\frac{U_{ik}}{p_k} \leq \frac{U_{ik}}{p_k^0} \leq \frac{U_{ij}}{p_j^0} = \frac{U_{ij}}{p_j}.$$

Therefore, $(i,j) \in E(p)$. Let $K = \{k \in G : x_{ik} > 0\}$. Because $(i,k) \in E(p)$, the above inequalities hold with equality. Therefore, for each $k \in K$, $p_k = p_k^0$. Moreover, $p_k^0 \leq e_i/n$ for each $k \in K$, and $c_i(x) = e_i - \sum_{k \in K} x_{ik} \geq e_i - \sum_{k \in K} (p_k + \Delta) \geq e_i - |G|e_i/n - |G|\Delta = |B|e_i/n - |G|\Delta > \Delta$. ■

We now return to the proof of Theorem 3.1.

Proof of correctness and number of calls of PriceAndAugment**.** By Lemma 3.1 and Lemma 3.2, if a solution is Δ-feasible at the beginning of a PriceAnd-Augment, then the sequence of solutions obtained during the procedure are all Δ-feasible.

We next bound the number of calls of PriceAndAugment during a scaling phase. Lemma 3.2 implies that each call of PriceAndAugment reduces Φ by exactly 1.

In the first scaling phase $c_i(x^0) \leq \Delta^0$ for all $i \in B$. Thus, $\Phi(x^0, \Delta^0) \leq |B| < n$.

Suppose next that Δ is a scaling parameter at any other scaling phase. Let (p,x) be the 2Δ-optimal solution at the end of the 2Δ-scaling phase. Thus $c_i(x) < 2\Delta$ for all $i \in B$. Therefore, $\Phi(x, \Delta) \leq |B|$. However, immediately after replacing 2Δ by Δ, the algorithm reduces by Δ the allocation to any good j with $b_j(p,x) > \Delta$. This transformation can increase Φ by as much as $|G|\Delta$. By Lemma 3.2, the number of calls of PriceAndAugment is at most $|B| + |G| = n$. ■

Proof of time bound per phase. We next show how to implement PriceAndAugment in $O(m + |B| \log |B|)$ time. The time for identifying an augmenting path, and modifying the allocation x is $O(n)$. So, we need only consider the total time for UpdatePrice.

We cannot achieve the time bound if we store prices explicitly since the number of prices changes during PriceAndAugment is $O(|B|^2)$. So, we store prices implicitly. Suppose that p is the vector of prices that is the input for PriceAndAugment, and let p' be the vector of prices at some point during the execution of the procedure and prior to obtaining an augmenting path. Let r be the root node of PriceAndAugment, and choose v so that $(r,v) \in E(p)$. Rather than store the vector p' explicitly, we store the vector p, the price p'_v, and the vector $\alpha(p)$. In addition, for each active node k, we store γ_k, which is the price of node v at the iteration at which node k became active. If node $j \in G$ is not active with respect to p', then $p'_j = p_j$. If j is active then Then $p'_j = p_j \times p'_v/\gamma_j$.

An edge (i,j) has the possibility of becoming an equality edge at the next price update only if i is active and j is not active. Suppose that $(i,k) \in E(p)$. In such a case, (i,j) will become active when the updated price vector $\hat{p}$ satisfies $U_{ij}/\hat{p}_j = \alpha_i(\hat{p}) = \alpha_i(p)\gamma_i/\hat{p}_v$. Accordingly, if node $i \in B$ is active, and if node $j \in G$ is not active, then the price β_{ij} of node v at the iteration at which (i,j) becomes an equality edge is $\beta_{ij} = \gamma_i \alpha_i(p) p_j / U_{ij}$.

For all inactive nodes $j \in G$, we let $\beta_j = \min\{\beta_{ij} : i \text{ is active}\}$, and we store the β's in a Fibonacci heap, a data structure developed by Fredman and Tarjan [9]. The Fibonacci heap is a priority queue that supports "FindMin", "Insert", "Delete", and "Decrease Key". The FindMin and Decrease Key operations can each be implemented to run in $O(1)$ time in an amortized sense. The Delete and Insert operations each take $O(\log |B|)$ time when operating on $|B|$ elements.

We delete a node from the Fibonacci heap whenever the node becomes active. We carry out a Decrease Key whenever β_j is decreased for some j. This event may occur when node i becomes active and edge (i, j) is scanned, and β_{ij} is determined. Thus the number of steps needed to compute when edges become equality edges is $O(m + |B| \log |B|)$ during PriceAndAugment.

In addition, one needs to keep track of the price of v at the iteration at which b_j will equal 0. Suppose that node $i \in B$ is active. Let q' be the minimum value ≥ 1 such that $b_j(q'p, x) = 0 \mod \Delta$. Thus $b_j = 0$ when the price of node v is $q'\gamma_j$. Accordingly, $\beta_j = q'\gamma_j$, and we store β_j in the same Fibonacci Heap as the inactive nodes of G. Thus the total time to determine all price updates is $O(m + |B| \log |B|)$. ■

3.5 Abundant edges and bounds on the number of scaling phases

Before proving the bound on the number of phases given in Theorem 3.1, we introduce another concept and prove a lemma.

An edge (i, j) is called Δ-*abundant* if $x_{ij} \geq 3n\Delta$, where x is the allocation at the beginning of the Δ-scaling phase. (We could use $2n$ rather than $3n$ here, but $2n$ will not be sufficient for the strongly polynomial time algorithm.)

The concept of abundance was introduced by Orlin [12] for a strongly polynomial time algorithm for the minimum cost flow problem. (See also, [13] and Chapter 10 of Ahuja et al., [1].) The following lemma shows that once an edge (i, j) becomes abundant, it stays abundant at every successive iteration. (The reader may note that the following lemma suggests that it would be better to define "abundance" with a right hand side of 2Δ rather than 3Δ. However, a larger value is needed for the proof of Property (iv) of Lemma 4.7.)

Lemma 3.4 *If edge (i, j) is abundant for the Δ-scaling phase, it is abundant at the $\Delta/2$-scaling phase, and at all subsequent scaling phases.*

Proof. Let x' and x'' be the initial and final allocations during the Δ-scaling phase. If (i, j) is Δ-abundant, then $x''_{ij} \geq 3n\Delta - n\Delta = 2n\Delta = 4n(\Delta/2)$. If it is abundant at the $\Delta/2$-scaling phase, by induction it will be abundant at all subsequent scaling phases. ■

Proof of Theorem 3.1: the number of phases. Suppose that (p^k, x^k) and Δ^k is the solution and scaling parameter at the beginning of the k-th scaling phase of the scaling algorithm. Because the change in any allocation and in any price is at most $n\Delta$ during the Δ-scaling phase, the sequences $\{p_k : k = 1, 2, \ldots\}$ and $\{x_k : k = 1, 2, \ldots\}$ both converge in the limit to the optimal solution (p^*, x^*).

Now suppose that $\Delta^k < D/8n$, where $D = (U_{max})^{-n}/n$. Let $E^k = \{(i, j) : x^k_{ij} > 4n\Delta^k\}$. We claim that BasicSolution(E^k) is optimal. Consider edge (i, j). If $(i, j) \in E^k$, then (i, j) is Δ-abundant, and by Lemma 3.4, $x^*_{ij} > 0$. If $(i, j) \notin E^k$, then $x^*_{ij} < x^k_{ij} + 4n\Delta^k < 1/D$. By Lemma 2.1, $x^*_{ij} = 0$. We have thus shown that BasicSolution(E^k) is optimal. The number of scaling phases needed to ensure that $\Delta^k < 1/8nD$ is $O(\log e_{max} + n \log U_{max})$. ■

3.6 Finding approximate solutions

Suppose that $\epsilon > 0$. A solution (p, x) is said to be *ϵ-approximate* if it is Δ-optimal for $\Delta < \epsilon \sum_{j \in G} p_j$.

Theorem 3.2 . ScalingAlgorithm *determines an ϵ-approximate solution after $O(\log(n/\epsilon))$ scaling phases. The running time is $O(n(m + n \log n) \log(n/\epsilon))$.*

Proof. Recall that $\Delta^0 = e_{max}$. Within $O(\log(n/\epsilon))$ scaling phases, $\Delta < \epsilon \cdot e_{max}/2n$. Suppose (p, x) is the solution at the end of that phase, and let $k = \text{argmax}\{e_i : i \in B\}$. Then $c_i(x) < \Delta$ and so $\sum_{j \in G} p_j > e_i - c_i(x) > e_{max}/2 > n\Delta/\epsilon$. Therefore, $\Delta < \epsilon \sum_{j \in G} p_j$. ■

The above approach is the fastest to determine an ϵ-approximate solution. Also, finding an approximate solution is quite practical. For example, if all initial endowments are in dollars and are bounded by $O(f(n))$ for some polynomial $f(\cdot)$, then ScalingAlgorithm determines a solution that is within 1 penny of optimal in $O(\log n)$ scaling phases.

3.7 A simple strongly polynomial time algorithm if $\log U_{max}$ is polynomially bounded.

Devanur et al. [5] showed how to find an optimal solution in the case that $U_{ij} = 0$ or 1 for each (i, j). Here we provide a simple algorithm for finding an optimal solution if U_{ij} is polynomially bounded in n.

In the following, we assume that buyers have been reordered so that $e_i \geq e_{i+1}$ for all $i = 1$ to $|B| - 1$. We let Problem(j) be the market equilibrium problem as restricted to buyers 1 to j, and including all goods of G. In the following theorem, $M = 4n^2(U_{max})^n$. We also relax (iii) of the definition of Δ-feasibility. We do not require allocations in abundant edges (see Subsection 3.5) to be multiples of Δ.

Theorem 3.3 *Suppose that $e_j > Me_{j+1}$, and that k is the next index after j for which $e_k > Me_{k+1}$. If one starts with an optimal solution for Problem(j), ScalingAlgorithm solves Problem(k) in $O(n(k - j) \log U_{max})$ scaling phases. By solving Problem(i) for each i such that $e_i > Me_{i+1}$, The total number of scaling phases to find an equilibrium is $O(n^2 \log U_{max})$.*

Proof. Suppose that (p', x') is the optimal solution for Problem(j). The algorithm solves Problem(k) starting with solution (p', x') and with scaling parameter $\Delta' = e_{j+1}$. We first show that this starting point is Δ'-feasible.

It is possible that allocations will not be a multiple of Δ'. However, if $x'_{il} > 0$ for some $i \leq j$, then by Lemma 2.1, $x'_{il} \geq e_j/D > 4ne_{j+1} = 4n\Delta'$. Thus (i, l) is abundant, and by Lemma 3.4, it remains abundant at all subsequent scaling phases.

We next bound the number of scaling phases for Problem(k). By Lemma 2.1, the optimum solution (p^*, x^*) for Problem(k) has the following property. If $x^*_{il} > 0$ for some $i \leq k$, then $x^*_{il} \geq e_k/D$. So, the algorithm will stop when the scaling parameter is less than $e_k/4nD$. Thus, the number of scaling phases is $O(\log 4nD(e_j/e_k)) = O(\log M^{k-j}) = O((k - j)(n) \log U_{max})$.

If we solve Problem(i) for each i such that $e_i > Me_{i+1}$, then the total number of scaling phases to find an equilibrium is $O(n^2 \log U_{max})$. ■

4 A strongly polynomial time algorithm

In this section, we develop a strongly polynomial time scaling algorithm, which we call StrongScaling. This algorithm is a variant of the scaling algorithm of Section 3. The number of scaling phases in StrongScaling is $O(n \log n)$, and the number of calls of PriceAndAugment is $O(n)$ at each phase. The total running time of the algorithm is $O((n^2 \log n)(m + n \log n))$.

4.1 Preliminaries

The algorithm StrongScaling relies on two new concepts: "surplus" of a subset of nodes and "fertile" price vectors. We also provide more details about abundant edges.

For the Δ-scaling phase, we let $A(\Delta)$ denote the set of Δ-abundant edges. We refer to each component in the graph defined by $A(\Delta)$ as a Δ-*component*. We let $\mathbf{C}(\Delta)$ denote the set of Δ-components.

Suppose that p is any price vector during the Δ-scaling phase. We let $N(p, \Delta)$ be the network in which the forward arcs are $E(p)$ and the backward arcs are the reversals of Δ-abundant edges. We refer to $N(p, \Delta)$ as the Δ-*residual network* with respect to p. We let *ActiveSet*(p, Δ, r) be the set of nodes reachable from r in the network $N(p, \Delta)$.

Suppose that $H \subseteq B \cup G$. We define the *surplus* of H with respect to a price vector p to be $s(p, H) = \sum_{i \in B \cap H} e_i - \sum_{j \in G \cap H} p_i$. Equivalently, $s(p, H) = e_H - p_H$.

Suppose that (p, x) is the solution at the beginning of the Δ-scaling phase. We say that a component H of $A(\Delta)$ is Δ-*fertile* if (i) H consists of a single node $i \in B$ and $e_i > \Delta/3n^2$, or if (ii) $s(p, H) \leq -\Delta/3n^2$. We say that the solution (p, x) is Δ-*fertile* if any of the components of $\mathbf{C}(\Delta)$ are Δ-fertile.

Lemma 4.1 *Suppose that (p, x) is the solution obtained at the end of the Δ-scaling phase. For each fertile Δ-component, there will be a new abundant edge incident to H within $5 \log n + 5$ scaling phases.*

Proof. Let Δ' be the scaling factor after at least $5 \log n + 4$ scaling phases. Then $\Delta' < \Delta/(16n^5)$. Let (p', x') be the solution at the beginning of the Δ'-scaling phase. If $H = \{i\}$, then $e_i > 3n^2\Delta'$, and so node i is incident to an edge (i, j) with $x'_{ij} > 3n\Delta'$. If H contains a node of G, then $s(p', H) \leq s(p, H) < -3n^2\Delta'$. Because $b_j(p', x') \geq 0$, it follows that there must be some edge (i, j) with $i \in B \setminus H$ and $j \in H \cap G$, such that $x'_{ij} \geq 3n\Delta'$, and so (i, j) is abundant at the Δ'-scaling phase. ■

Because the set of optimal edges is cycle free, there are at most $n - 1$ optimal edges, and so there are at most $n - 1$ abundant edges. If the solution at the beginning of each scaling phase were fertile, then ScalingAlgorithm would identify all abundant edges in $O(n \log n)$ scaling phases, after which the algorithm would identify the optimal solution. However, it is possible that a solution is infertile at a scaling phase, as illustrated in the following example, which shows that ScalingAlgorithm is not strongly polynomial.

Example. $B = \{1\}$; $G = \{2, 3\}$; $e_1 = 1$; $U_{12} = 2^K$; $U_{13} = 1$, and K is a large integer. The unique optimal solution is: $p_2 = x_{12} = 2^K/(2^K + 1)$, and $p_3 = x_{13} = 1/(2^K + 1)$. At the k-th

scaling phase for $k < K$, the Δ-optimal solution has $x_{12} = 1 - 2^k$ and $x_{13} = 2^k$. It takes more than K scaling phases before $(1,3)$ is larger than $2n\Delta$ and the optimal solution is determined.

4.2 The Algorithm StrongScaling

In this section, we present the algorithm StrongScaling. Its primary difference from ScalingAlgorithm occurs when a solution is Δ-infertile at the beginning of a scaling phase. In this case, StrongScaling calls procedure MakeFertile, which produces a smaller parameter Δ' (perhaps exponentially smaller) and a feasible vector of prices p' that is Δ-fertile. It then returns to the Δ'-scaling phase.

In addition, there are two more subtle differences in the algorithm StrongScaling. The procedure MakeFertile permits allocations on abundant edges to be amounts other than a multiple of Δ. In addition, the procedure produces a solution (p', x') such that $b_j(p', x')$ is permitted to be negative, so long as $b_j(p', x') \geq \Delta'/n^2$.

Algorithm 3. StrongScaling(e, U)
begin
  $\Delta := \Delta^0$; $p := p^0$; $x := 0$; Threshold $:= \Delta$;
  ScalingPhase:
    if BasicSolution$(A(\Delta))$ is optimal, then quit;
    while (p, x) is not Δ-optimal, replace (p, x) by PriceAndAugment(p, x);
    if no Δ-component is Δ-fertile, and if $\Delta \leq$ Threshold, then MakeFertile(p, x)
    else do
      $\Delta := \Delta/2$;
      for each $j \in G$ such that $b_j(p, x) > \Delta$, decrease x_{ij} by Δ for some $i \in B$;
      return to ScalingPhase;
end

Algorithm 4. MakeFertile(p, x, Δ)
begin
  $\Delta' =$ GetParameter(p, Δ);
  if $\Delta' > \Delta/n^2$ then Threshold $:= \Delta/n^5$ and quit;
  else continue
  $p' :=$ GetPrices(p, Δ');
  $x' :=$ GetAllocations(p, Δ');
end

In the remainder of this section, we let (p, x) be the solution at the end of the Δ-scaling phase, and we assume that (p, x) is infertile. We use $\mathbf{C}$ instead of $\mathbf{C}(\Delta)$.

The procedure MakeFertile(p, Δ) has three subroutines. The first subroutine is called GetParameter(p, Δ). It determines the next value Δ' of the scaling parameter. If $\Delta' > \Delta/n^2$, the procedure sets Threshold to Δ/n^5 and returns to the Δ-scaling phase. (The purpose of the parameter Threshold is to prevent MakeFertile from being called for at least $5 \log n$ scaling phases.) In this case, even though p is not Δ-fertile, there will be a new abundant edge within $O(\log n)$ scaling

phases. If $\Delta' \leq \Delta/n^2$, MakeFertile continues. The second subroutine is called GetPrice(p, Δ'), which determines a Δ'-feasible set of prices p'. The price vector p' is Δ'-fertile. In addition, $-\Delta'/2n \leq s(p', H) \leq \Delta'$ for all $H \in \mathbf{C}$. The third subroutine is called GetAllocations(p', Δ'). It obtains an allocation x' so that (p', x') is nearly Δ'-feasible, and where allocations are only made on abundant edges. Subsequently, StrongScaling runs the Δ'-scaling phase.

In the remainder of this section, we present the subroutines of MakeFertile. In addition, we will state lemmas concerning the properties of their outputs Δ', p', and x'. The algorithms GetParameter and GetPrices each rely on a procedure called SpecialPrice(p, H, t), where H is a Δ-component, and $t \geq 0$. The procedure produces a feasible price vector $\hat{p}$, which depends on H and t. The price vector p is increased until one of the following two events occurs: $s(\hat{p}, H) = t$, or there is a component $J \in \mathbf{C}$ with $s(\hat{p}, J) = -s(\hat{p}, H)/2n^2$.

The procedure GetParameter computes the next value of the scaling parameter. For each $H \in \mathbf{C}$ with at least one node in G, the procedure runs SpecialPrice$(p, H, 0)$ and computes the vector of prices as well as the scaling parameter, which we denote as Δ_H. If $H = \{i\}$ (and thus H has no node of G), then $\Delta_H := e_i$. The algorithm then sets $\Delta' = \max\{\Delta_H : H \in \mathbf{C}\}$.

The procedure GetPrices computes the next price vector. It computes SpecialPrice(p, H, Δ') for all $H \in \mathbf{C}$, and then lets p' be the component-wise maximum of the different price vectors.

These three procedures are presented next. (We will discuss the procedure GetAllocations later in this section.)

Procedure SpecialPrice(p, H, t)
begin
  $p' := p$;
  select a node $v \in H \cap G$;
  while $s(p', H) > t$ and $s(p', J) > -s(p', H)/2n^2$, for all $J \in \mathbf{C}$ do
    compute the nodes ActiveSet(p', Δ, v);
    $\forall$ inactive nodes $j \in G$, $\hat{p}_j := p'_j$;
    $\forall$ active nodes $j \in G$, $\hat{p}_j := qp'_j$, where $q > 1$ is the least value so that
      (i) a new edge becomes an equality edge or
      (ii) $s(\hat{p}, H) = t$ or
      (iii) $s(\hat{p}, J) = -s(\hat{p}, H)/2n^2$ for some $J \in \mathbf{C}$;
    $p' := \hat{p}$;
  endwhile
end

Procedure GetParameter(p, Δ)
begin
  $\forall$ components $H \in \mathbf{C}$ do
    if $H = \{i\}$ and $i \in B$, then $\Delta_H := e_i$;
    else $\hat{p} :=$ SpecialPrice$(p, H, 0)$, and $\Delta_H := s(\hat{p}, H)$;
  $\Delta' := \max\{\Delta_H : H \in \mathbf{C}\}$;
end

Procedure GetPrices(p, Δ')
begin
  $\forall$ components $H \in \mathbf{C}$ that contain a node of G do
    if $s(p, H) \leq \Delta'$, then $p^H := p$;
    else $p^H :=$ SpecialPrice(p, H, Δ');
  $\forall j \in G,\ p'_j :=$max $\{p_j^H : H \in \mathbf{C}\}$;
end

The following lemma concerns properties of Δ' and p' that are needed for the procedure GetAllocation. We let Δ_H and p^H denote parameters and prices calculated within GetParameter and GetPrices.

Lemma 4.2 *Suppose that Δ' and p' are the output from* GetParameter *and* GetPrices. *Then the following are true:*

i. For all $H \in \mathbf{C}$, $s(p', H) \geq -\Delta'/n^2$.

ii. If $H \in \mathbf{C}$ and if $H = \{i\}$ and $i \in B$, then $e_i \leq \Delta'$.

iii. For all $H \in \mathbf{C}$, $s(p^H, H) = min\{s(p, H), \Delta'\}$.

iv. There is a component $H \in \mathbf{C}$, such that $p_H^H = p'_H$ and $s(p', H) = \Delta'$.

v. Either there is a node $i \in B$ with $e_i = \Delta'$, or there exists $J \in \mathbf{C}$ with $s(p', J) = -\Delta'/n^2$.

vi. If $s(p', J) < 0$, then there is $H \in \mathbf{C}$ with $s(p', H) = \Delta'$ and $J \subseteq$ ActiveSet(p', H).

Proof. This lemma is highly technical, and the proof of Statements (iv), (v), and (vi) are involved. We first establish the correctness of Statements (i) to (iii). Prior to Proving Statements (iv), (v), and (vi), we will establish additional lemmas.

Proof of parts (i) to (iii). Consider Statement (i). There exists component K such that $s(p', H) = s(p^K, H) \geq -s(p^K, K)/n^2 \geq -\Delta'/n^2$, and so the statement is true. Now consider statement (ii). If $H = \{i\}$, then $\Delta_H = e_i \leq \Delta'$.

Consider statement (iii). If $s(p, H) < \Delta'$, then SpecialPrice(p, H, Δ') lets $p^H = p$. Otherwise, SpecialPrice increases the prices (and thus decreases $s(p^H, H)$) until $s(p^H, H) = \Delta'$. The other termination criteria do not play a role until $s(p^H, H) = \Delta_H \leq \Delta'$. ■

4.3 The auxiliary network and maximum utility paths

Before proving Statements (iv) to (vi) of Lemma 4.2, we develop some additional properties of prices and paths. We again assume that (p, x) is the input for MakeFertile. Let $A(\Delta)$ be the set of abundant edges. The *auxiliary network* is the directed graph N^* with node set $B \cup G$, and whose edge set is as follows. For every $i \in B$ and $j \in G$, there is an arc (i, j) with utility U_{ij}. For every $(i, j) \in A(\Delta)$, there is an arc (j, i) with utility $U_{ji} = 1/U_{ij}$. (There will also be an arc (i, j).) For every directed path $P \in N^*$, we let the utility of path P be $U(P) = \prod_{(i,j)\in P} U_{ij}$. For every pair of nodes j and k, we let $\mu(j, k) = \max\{U(P) : P$ is a directed path from j to k in $N^*\}$. That is, $\mu(j, k)$ is the maximum utility of a path from j to k.

Lemma 4.3 *The utility of each cycle in the auxiliary network N^* is at most 1.*

Proof. Let C be any cycle in the auxiliary network. For each edge (i, j) of the auxiliary network, let $U'_{ij} = U_{ij}/(\alpha_i(p) \cdot p_j)$. Since $U_{ij}/p_j \leq \alpha_i(p)$, it follows that $U'_{ij} \leq 1$ for all (i, j).

Let $U'(P) = \prod_{(i,j)\in P} U'_{ij}$. It follows that $U(P) = U'(P) \leq 1$, and the lemma is true. ■

We assume next that all components of $A(\Delta)$ have a single root node, and that the root node is in G whenever the component has a node of G. We extend μ to being defined on components as follows: $\mu(H, K) = \mu(v, w)$, where v is the root node of H, and w is the root node of K.

Lemma 4.4 *Suppose that $K \in \mathbf{C}$ and that $p^K = SpecialPrice(p, K, t)$. If component $H \subseteq ActiveSet(p^K, \Delta, K)$, then $\mu(K, H) = p^K_H/p^K_K$.*

Proof. For each (i, j) in the auxiliary network, let $U'_{ij} = U_{ij}/(\alpha_i(p^K) \cdot p^K_j)$. Then $U'_{ij} \leq 1$ for all (i, j). Suppose next that P is any maximum utility path from a node v to a node w with respect to U. Then P is also a maximum utility path from v to w with respect to U'. Moreover, if $v \in G$ and $w \in G$, then $U'(P) = U(P) \cdot p^K_v/p^K_w$.

We now consider the path P' from the root of K to the root of H in the residual network $N(p^k, \Delta)$. Such a path exists because H is active. Then each arc of P' is an equality arc, and $U'(P') = 1$. It follows that P' is a maximum utility path. Moreover, $U(P) = U'(P) \cdot p^K_H/p^K_K = p^K_H/p^K_K$. ■

Lemma 4.5 *Suppose that K and H are components of $\mathbf{C}$, and that p^K and p^H are price vectors determined in* GetPrices. *If $p^H_H < p^K_H$, then $p^H_H/p^K_K < \mu(H, K)$.*

Proof. $p^H_H/p^K_K = p^H_H/p^K_H \times p^K_H/p^K_K$. By assumption, $p^H_H/p^K_H < 1$. This implies that $p_H < p^K_H$, and so component $H \subseteq ActiveSet(p^K, \Delta, K)$. By Lemma 4.4, $p^K_H/p^K_K = \mu(H, K)$. Therefore $p^H_H/p^K_K < \mu(H, K)$. ■

4.4 The rest of the proof of Lemma 4.2

We are now ready to prove Statement (iv), (v), and (vi) of Lemma 4.2.

Proof of Statement (iv) of Lemma 4.2. We assume that there is no component H such that $s(p', H) = \Delta'$, and we will derive a contradiction. By statement (iii), if $s(p, H) \geq \Delta'$, then $s(p^H, H) = \Delta'$. By construction, if $s(p^H, H) = \Delta'$ and if $s(p', H) < \Delta'$, then there is some component K such that $s(p^K, K) = \Delta'$, and $p^H_H < p^K_H$. In such a case, we say that K dominates H.

Let H_1 be a component for which $s(p^H, H) = \Delta'$. By our assumption, there is a sequence of components $H_1, H_2, \ldots, H_k$ such that H_{i+1} dominates H_i for $i = 1$ to $k - 1$, and H_k dominates H_1. For each component $J \in \mathbf{C}$, let $g(J) = p^J_J$. Let $H_{k+1} = H_1$. By Lemma 4.5,

$$\prod_{i=1}^{k} \mu(H_{i+1}, H_i) > \prod_{i=1}^{k} g(H_i)/g(H_{i+1}) = 1.$$

For $i = 1$ to k, let P_i be the max multiplier path from H_{i+1} to H_i, and let $P = P_1, P_2, \ldots, P_k$ be the closed walk obtained by concatenating the k paths. Then $U(P) = \prod_{i=1}^{k} \mu(H_{i+1}, H_i)$ But by Lemma 4.3, $U(P) \leq 1$, which is a contradiction. We conclude that there is a component H such that $s(p', H) = \Delta'$. ■

Proof of Statement (v) of Lemma 4.2. Choose H such that $s(p', H) = \Delta'$. If $H = \{i\}, \Delta' = e_i$. The remaining case in when H contains a node of G. Then the termination criteria for SpecialPrice$(p, H, 0)$ specifies that that there is a component J such that $s(p^H, J) = -\Delta'/n^2$. We claim that $s(p', J) = s(p^H, J) = -\Delta'/n^2$. Otherwise, there is a component K such that $p^K_J > p^H_J$. But then $s(p^K, J) < s(p^H, J) = -s(p^H, H)/n^2 = -\Delta'/n^2 \leq -s(p^K, K)/n^2$, which is a contradiction of the termination rule for SpecialPrice(p, K, Δ'). ■

Proof of Statement (vi) of Lemma 4.2. We first prove the statement in the case that $s(p', J) < s(p, J)$, and thus $p'_J > p_J$. In this case, there exists a component H such that $p'_J = p^H_J > p_J$. Let Q be the max utility path from H to J in N^*. We claim that $p'_H = p^H_H$. Otherwise, there is a component K that dominates H; that is, $p^K_H > p^H_H$. By Lemma 4.4, $p^K_J > p^H_J$. Therefore, $s(p^K, J) < s(p^H, J) = -s(p^H, H)/n^2 = -\Delta'/n^2 \leq -s(p^K, K)/n^2$, a contradiction. Thus Statement (vi) is true in this case.

We will show that there is a component H with $s(p, H) \geq n\Delta'$ and $J \subseteq$ Activeset(p, Δ, H). In this case, $p'_J \geq p^H_J > p_J$, and thus Statement (vi) is true because it reduces to the case given above. We next determine an allocation $\hat{x}$ as restricted to the abundant edges and that is nearly Δ-feasible. We then use flow decomposition (see Ahuja et al. [1] pages 79-81) to determine the component H.

Consider the allocation $\hat{x}$ obtained from p that satisfies the following:

i. if $(i, j) \notin A(\Delta)$, then $\hat{x}_{ij} = 0$;

ii. $c(\hat{x}) \geq 0$; if $s(p, H) \geq 0$, then $c_H(\hat{x}) = s(p, H)$;

iii. $b(p, \hat{x}) \leq 0$; if $s(p, H) < 0$, then $b_H(p, \hat{x}) = s(p, H)$.

.

Consider the flow decomposition of $y = x - \hat{x}$. The flow decomposition is the sum of nonnegative flows in paths in $N(p, \Delta)$, where each path is from a deficit node (a node v with $\sum_{k \in G} \hat{y}_{vk} > 0$) to an excess node (a node w with $\sum_{k \in B} \hat{y}_{kw} < 0$).

Consider the flow in the decomposition out of component J. We claim that J is a deficit component, and the deficit is at least $(n-1)\Delta/n$. To see that this claim is true, note that $b_J(p, x) \geq 0$, and $b_J(p, \hat{x}) = s(p, J) < 0$. Moreover, since x_{ij} is a multiple of Δ on non-abundant edges, it follows that $b_J(p, x) \equiv s(p, J) \mod \Delta$, and thus $b_J(p, x) \geq s(p, J) + \Delta > -\Delta/n^2 + \Delta > (n-1)\Delta/n$. Since $b_J(p, \hat{x}) \leq 0$, it follows that there must be at least $(n-1)\Delta/n$ units of flow into component J in the flow decomposition. Accordingly, there must be at least one excess component H such that the flow from H to J in the flow decomposition is at least Δ/n. Since all flows in the decomposition are on equality edges, it follows that $J \subseteq$ ActiveSet(p, Δ, H). Moreover, $s(p, H) \geq \Delta/n \geq n\Delta'$, which is what we needed to prove. Therefore (vi) is true. ■

4.5 Determining the allocations

In procedure GetAllocations, for each Δ-component H with at least two nodes, we designate one node in $H \cap B$ as the *B-root* of H, and we designate one node in $H \cap G$ as the *G-root* of H. If H has a single node k, then k is the root.

The procedure will determine the allocation x' by first a basic feasible solution for a minimum cost flow problem. (See, e.g., [1].) In the procedure, we will let d_k denote the supply/demand of node k. If $i \in B$, then $d_i \geq 0$. If $j \in G$, then $d_j \leq 0$.

Procedure GetAllocations(p', Δ')
begin
  $\forall$ components $H \in \mathbf{C}$
    if i is the B-root of H, then $d_i = \max\{0, e_H - p'_H\}$;
    if j is the G-root of H, then $d_j = \min\{0, e_H - p'_H\}$;
    if k is not a B-root or G-root, then $d_k = 0$;
  select x' as the unique solution satisfying the supply/demand constraints
    and such that $x'_{ij} = 0$ for all non-abundant edges (i, j);
end

4.6 On the correctness of StrongScaling and its running time

In this section, we provide lemmas that lead up to the theorem stating the correctness and time bounds for StrongScaling.

Lemma 4.6 *Suppose that Δ' is the parameter obtained by running* GetParameter*, and suppose that $\Delta' > \Delta/n^2$. If the* ScalingAlgorithm *is run starting with solution (p, x) and with the Δ-scaling phase, then there will be a new abundant edge within $O(\log n)$ scaling phases.*

Proof. By Lemma 4.2, we can select a component H such that $s(p^H, H) = \Delta'$. Then there must also be a component J with $s(p^H, J) = -\Delta_H/n^2$. (The termination criterion would not end with $e_i = \Delta'$ because (p, x) was not fertile.)

Because $\Delta' > \Delta/n^2$, StrongScaling went to the $\Delta/2$-scaling phase, starting with the solution (p, x). Suppose that $\hat{\Delta} < \Delta'/n^2$, and let $(\hat{p}, \hat{x})$ be a feasible solution at the end of the $\hat{\Delta}$-scaling phase. We will show that $(\hat{p}, \hat{x})$ or else $(\hat{p}, \hat{x})$ is $\hat{\Delta}$-fertile, and there will be a new abundant edge within $O(\log n)$ scaling phases. We consider two cases. Assume first that $\hat{p}_H \leq p^H_H$. In this case, $s(\hat{p}, H) \geq \Delta' \geq n^2\hat{\Delta}$, and there must be an abundant edge directed out of H.

In the second case, $\hat{p}_H > p^H_H$. In this case, $s(\hat{p}, J) \leq s(p^H, J) = -\Delta'/n^2 < -\hat{\Delta}/n^4$. In this case, J is $\hat{\Delta}$-fertile, and there will be a new abundant edge in O(log n) scaling phases. ■

Lemma 4.7 *Suppose that Δ' and (p', x') are the parameter and solution obtained by running* MakeFertile*, and suppose that $\Delta' \leq \Delta/n^2$. Then the following are true:*

i. (p', x') *is Δ'-fertile.*

ii. $c_i(x') \leq \Delta'$ *for all $i \in B$.*

iii. $-\Delta'/n^2 \leq b_j(p', x') \leq 0$ *for all $j \in G$.*

iv. If $(i, j) \in A(\Delta)$, then $x'_{ij} > 2n\Delta'$.

v. If $\Delta'' < \Delta/3n^2$, then the solution at the Δ''-scaling phase is Δ''-feasible. Within an additional $2\log n + 4$ scaling phases, there will be a new abundant edge.

Proof. Statement (i) follows from Statements (iv) and (v) of Lemma 4.2. Statements (ii) and (iii) follow from Statements (i) and (iii) of Lemma 4.2, as well as the way that x' is constructed.

We next prove Statement (iv). Consider $\hat{x}$ described in the proof of Statement (vi) of Lemma 4.2. We first claim that $c_B(\hat{x}) < (n+1)\Delta$. The claim is true because $c_B(\hat{x})+b_G(p,\hat{x}) = e_B - p_G = c_B(x)+b_G(p,x) \leq n\Delta$. Also, $b_j(p,\hat{x}) \geq -\Delta/n^2$. Thus $c_B(\hat{x}) \leq n\Delta - b_G(p,\hat{x}) < n\Delta + (|G|/n^2)\Delta < (n+1)\Delta$.

Now consider the flow decomposition of $y = x - \hat{x}$ in the proof of Statement (vi) of Lemma 4.2. The sum of the excesses is at most $c_B(\hat{x}) < (n+1)\Delta$. It follows that the total flow in the decomposition emanating from excess nodes is less than $(n+1)\Delta$. Therefore, $|\hat{x}_{ij} - x_{ij}| < (n+1)\Delta$ for all $(i,j) \in A(\Delta)$.

Now consider x' as determined by Procedure GetAllocations. Note that $\sum_{j\in G} p_j \leq \sum_{j\in G} p'_j \leq \sum_{j\in G} p_j + n\Delta$. Therefore, $|\hat{x}_{ij} - x'_{ij}| < n\Delta$ for all $(i,j) \in A(\Delta)$. We conclude that $|x_{ij} - x'_{ij}| < (2n+1)\Delta$. Because $x_{ij} \geq 3n\Delta$, $x'_{ij} > 3n\Delta - (2n+1)\Delta = (n-1)\Delta > (n^3-n)\Delta'$, and thus (i,j) is abundant at the Δ'-scaling phase.

We next prove Statement (v). Let (p'', x'') be the solution at the Δ''-scaling phase, where $\Delta'' < \Delta'/4n^2$. Let J be a component with $s(p', J) < 0$. Statement (v) says that $b_J(p'', x'') \geq 0$.

So, suppose instead that $b_J(p'', x'') < 0$. We will derive a contradiction. We note that $p''_J = p'_J$. (Recall that a node j only increases its price when j is active and when there is no augmenting path. This cannot occur for a node j with $b_j < 0$.) We next claim that $p''_k = p'_k$ if $J \subset \text{ActiveSet}(p', \Delta, H)$. To see why, let Q' be a maximum multiplier path from k to J. Then by Lemma 4.4, $p'_J = p'_k \cdot \mu(k, J)$, and $p''_J \geq p''_k \cdot \mu(k, J)$. Since $p'_J = p''_J$, and $p''_k > p'_k$, it follows that $p''_k = p'_k$.

By Statement (vi) of Lemma 4.2, there is a component H with $s(p', H) = \Delta'$. In addition, $J \subseteq \text{ActiveSet}(p', \Delta, H)$. We are now ready to show to derive the contradiction, and thus show that $b_J(p'', x'') \geq 0$. Note that $s(p'', H) = \Delta'$, and $b(p'', H) \leq \Delta'' \leq \Delta/4n^2$. Now consider the flow decomposition $y' of x'' - x'$. The flow out of component H in y' is at least $(n-1)\Delta'/n$, and so there must be a flow of at least Δ'/n to some component K. Each arc in the flow decomposition is in $N(p', x')$. Then $p''_K = p''_H \cdot \mu(H, K)$, which implies that $p''_K = p'_K$. But then $b_K(p'', x'') = b_K(p', x'') > b_K(p', x') + \Delta'/n > (4n-1)\Delta''$, which is a contradiction. ■

Theorem 4.1 Optimality of Strong Scaling. *The algorithm* StrongScaling *finds an optimal solution* (p^*, x^*) *in* $O(n \log n)$ *scaling phases. It calls* MakeFertile *at most* n *times, and each call takes* $O(n(m + n \log n))$ *time. It calls* PriceAndAugment $O(n)$ *times per scaling phase. It runs in* $O((n^2 \log n)(m + n \log n)$ *time.*

Proof. Once all of the abundant edges have been identified, the algorithm finds the optimal solution. And a new abundant edge is discovered at least once every $O(\log n)$ scaling phases. Thus the algorithm terminates with an optimal solution.

We now consider the running time. The algorithm calls MakeFertile at most n times because each call leads to a new abundant edge. Each call of MakeFertile takes $O(n(m + n \log n)$ time. There are $O(\log n)$ scaling phases between each call of MakeFertile, and each scaling phase takes $O(n(m + n \log n)$ time. Thus the running time of the algorithm is $O((n^2 \log n)(m + n \log n))$ time. ■

5 Summary

We presented a new scaling algorithm for computing the market clearing prices for Fisher's linear market model. We anticipate that this technique can be modified to solve extensions of the linear model.

We have also presented a strongly polynomial time algorithm. The strongly polynomial time algorithmic development may possibly be modified to solve extensions of the linear model; however, such modifications are challenging because of the technical aspects of the proof of validity.

Acknowledgements
The author gratefully acknowledges all of the help and advice from Dr. Mehdi Ghiyasvand.

References

[1] R.K. Ahuja, T.L. Maganti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, N.J. 1993.

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

[3] W.C. Brainard, H.E. Scarf, D. Brown, F. Kubler, and K.R. Sreenivasan. How to compute equilibrium prices in 1891. Commentary. *The American Journal of Economics and Sociology* 64: 57-92, 2005.

[4] D. Chakrabarty, N. Devanur, and V. Vazirani. New results on rationality and strongly polynomial solvability in Eisenberg-Gale markets. In Proceedings of 2nd Workshop on Internet and Network Economics, 239-250, 2006.

[5] N.R. Devanur, C.H. Papadimitriou, A. Saberi, and V.V. Vazirani. Market equilibrium via a primal–dual algorithm for a convex program. *Journal of the ACM* 55: 1-18, 2008.

[6] N.R. Devanur and V.V. Vazirani. The spending constraint model for market equilibrium: algorithmic, existence and uniqueness results. Proceedings of 36th Symposium on the Theory of Computing, 519-528, 2004.

[7] J. Edmonds., and R.M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of ACM* 19: 248-264, 1972.

[8] E. Eisenberg and D. Gale. Consensus of subjective probabilities: the Pari-Mutuel method. *Annals of Mathematical Statistics* 30: 165-168, 1959.

[9] M.L. Fredman, and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of ACM* 34: 596-615, 1987.

[10] R. Garg and S. Karpur. Auction algorithms for market equilibrium. Proceedings of the 36th Symposium on the Theory of Computing, 511-518, 2004

[11] M. Ghiyasvand and J.B. Orlin An improved approximation algorithm for computing Arrow-Debreu market prices MIT Technical Paper. Submitted for publication. 2009.

[12] J.B. Orlin. Genuinely polynomial simplex and non-simplex algorithms for the minimum cost flow problem. Technical Report 1615-84, Sloan School of Management, MIT, Cambridge, MA. 1984.

[13] J.B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research* 41: 338-350, 1993.

[14] V.V. Vazirani. Spending constraint utilities, with applications to the Adwords market. Submitted to *Math of OR*, 2006.

[15] V.V. Vazirani. Combinatorial Algorithms for Market Equilibrium. in N. Nisan, T. Roughgarden, E'. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*, pages 103-134, Cambridge University Press, Cambridge, UK., 2007.