A New Strongly Polynomial Algorithm for Computing Fisher Market Equilibria with Spending Constraint Utilities

by

Zi Wang

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
May 2016

Advisor: Richard Cole

Second Reader: Chee Yap

Abstract

This thesis develops and analyzes an algorithm to compute equilibrium prices for a Fisher market in which the buyer utilities are given by spending constraint functions, utility functions originally defined by Devanur and Vazirani [5].

Vazirani gave a weakly polynomial time algorithm to compute the equilibrium prices [10]. More recently Végh gave a strongly polynomial algorithm [11]. Here we provide another strongly polynomial algorithm, which arguably is conceptually simpler, although the running time is not always better.

Acknowledgments

I would like to thank Professor Richard Cole for his invaluable guidance in all academic respects. This thesis would not have been possible without his ingenious ideas. I also want to express my gratitude to the department of computer science for the generous support from the M.S. Thesis/Research Fellowship. Last but not least, I owe my deepest gratitude to my parents for their continued support.

In memory of Ahui (2001 – 2014)

My beloved dog and adolescent friend

纪念阿灰

Table of Contents

Introduction

Market equilibria are one of the fundamental topics in economics. Their study was initialized by Walras in 1874 [12]. The proof of their existence was given by Arrow, Block, and Hurwitz, and Nikaido and Uzawa [8, 2, 1]. Subsequently, there has been a large body of work concerned with the properties of the equilibria: their multiplicity, their stability, and their computability. Over the last 15 years or so, computer scientists have investigated their computational complexity. Early on it was shown that a variant of the problem for indivisible goods was NP-complete [4]. This led to the search for classes of markets and buyer utility functions for which computing equilibrium prices was in P. The first major result was by Devanur et al. [6] who showed that in Fisher markets, when the utilities were linear functions, there is a (weakly) polynomial time algorithm. Other works gave polynomial time algorithms when the utility functions were CES functions and Leontief functions, again for Fisher markets [7].

Devanur and Vazirani generalized the linear utilities to spending constraint utilities, arguing that they captured multiple natural settings, and they generalized their weakly polynomial algorithm for linear utility functions to this larger class [5].

Subsequently, Orlin [9] gave a strongly polynomial algorithm for the linear utility case. A variant of this algorithm was developed by Cole and Gkatzelis [3] for the spending restricted setting. It is this variant that provides the starting point for the algorithm in this paper. Végh [11] also gave a strongly polynomial algorithm for a variety of min-cost network flow problems, including a problem obtained as a reduction from the equilibrium computation for the spending constraint utilities.

In contrast to Végh's work, the present work tackles the algorithmic problem directly. We obtain a running time of $O(rm^2n + rm \log n[n \log n + m])$, whereas Végh achieves $O(rn^3 + r^2[r + n \log n] \log r)$. In general, $n = O(m) = O(r)$ and $m = O(n^2)$, thus which bound is better depends on the relative sizes of $r^2 \log r$ and $m^2 n$.

However, our hope is that the more direct approach taken in this work will facilitate extension of this methodology to other equilibrium problems for which strongly polynomial algorithms are not yet known, in particular for exchange markets, in which agents have linear utilities.

1
Description of the Problem

Definition 1.1. A *Fisher market*[1] consists of divisible goods G and agents B, called buyers. Let $n = |B| + |G|$. There is a fixed endogenous supply of each good (which WLOG is chosen to be 1 unit). A buyer i has a fixed endowment of $\mathbf{e}_i$ units of money. Each agent has a utility function, with the characteristic that the agent has no desire for its money, i.e. each agent seeks to spend all its money on goods. Suppose we assign a price $\mathbf{p}_j$ to each good j, then a (possibly non-unique) *demand* of agent i is a bundle of goods $(x_{i1}, x_{i2}, \ldots, x_{i|G|})$ that maximizes her utility subject to the budget constraint: $\sum_{j \in G} p_j x_{ij} \leq \mathbf{e}_i$. A *market demand* x for a good j is the total (possibly non-unique) demand for that good; $x_j = \sum_{i \in B} x_{ij}$. This is viewed as a function of the price vector $\mathbf{p} = (p_1, p_2, \ldots, p_{|G|})$. Prices $\mathbf{p}$ provide an *equilibrium* if the resulting markets can clear, that is there exists a market demand at these prices such that for all j, $x_j = 1$ if $p_j > 0$ and $x_j \leq 1$ if $p_j = 0$. For notational convenience, we define an *excess demand* for good j as $z_j = x_j - 1$. The equilibrium condition is that every excess demand for positive priced goods be zero. Prices $\mathbf{p}$ form an equilibrium if there exists market demands at these prices that clear the market.

The Fisher market is actually a special case of an Exchange economy. (To see this, view the money as another good, and the supply of the goods as being initially owned by another agent, who desires only money.)

In this thesis, we consider Fisher markets with spending constraint utilities. For each $i \in B$ and $j \in G$, f_{ij} is a utility function which depends on the spending that i has assigned to j. In our problem, f_{ij} is a decreasing step function. We call each step of f_{ij} a *segment*. Suppose f_{ij} has segments $s_{ij}^1, s_{ij}^2, ..., s_{ij}^k$ (strictly speaking, k is a function i and j) in left to right order, i.e. s_{ij}^h has range $(a_h, b_h]$ with $b_h = a_{h+1}$ for $h < k$, and $a_1 = 0$. Let s be one of the segments for f_{ij}. Suppose that s has range $(a, b], 0 \leq a < b \leq \mathbf{e}_i$ and $f_{ij}(x) = c$ for $x \in (a, b]$. We define capacity$(s) = b - a$ and utility$(s) = c$. Finally, m denotes the number of edges, that is of pairs (i, j) with utility$(s_{ij}^1) > 0$, where s_{ij}^1 denotes the first segment between i and j, and r denotes

[1] In much of the Computer Science literature the term market has been used to mean what is called an economy in the economics literature.

the number of segments. These edges of pairs (i, j) with utility$(s_{ij}^1) > 0$ form a bipartite graph on $B \times G$, which we call the *Interest Network*. d_i denotes the degree of vertex i in the Interest Network. It will be convenient to have n, m and r be powers of 2 and to assume $n < m \leq r$, which we do henceforth without loss of geneality.

For each segment s, $\sigma(s)$ denotes the spending on that segment, while $\mathbf{q}_{ij}$ denotes the spending of buyer i on good j. $\mathbf{q}$ denotes the vector $\mathbf{q}_{ij}$. The *surplus cash* of buyer i is defined as $c_i(\mathbf{q}) = \mathbf{e}_i - \sum_{j \in G} \mathbf{q}_{ij}$. The *oversold amount* of good j is $b_j(\mathbf{p}, \mathbf{q}) = -\mathbf{p}_j + \sum_{i \in B} \mathbf{q}_{ij}$. Naturally, buyer i will accrue utility from segments $s_{ij}^1, s_{ij}^2, \ldots$ in turn, as she increases her spending on good j. Given $\mathbf{p}$, $\mathbf{q}$, we partition the segments into three categories: *working* segments, *fulfilled* segments and *unused* segments. If $\mathbf{q}_{ij}$ lies in the range of segment s_{ij}^h, s_{ij}^h is called a *working* segment. The higher utility segments, $s_{ij}^g, g < h$, are called *fulfilled* segments. The lower utility segments, $s_{ij}^l, h < l$, are called *unused* segments. Let F be the set of *fulfilled* segments. A segment s is said to be *maximum bang per buck* (MBB) for i if $s \notin F$ and it maximizes utility$(s_{ij}^k)/\mathbf{p}_j$ for all $s_{ij}^k \notin F$. A segment is *sub-MBB* if its utility to price ratio is lower than that of the MBB segments. A segment is *filled* if its spending equals its capacity. Henceforth, we will drop the superscript k from the segment name when no ambiguity will result.

An equilibrium is reached if the following conditions are satisfied:

- For $j, k \in G$ and $i \in B$, if s_{ij} and s_{ik} are working segments, then utility$(s_j)/\mathbf{p}_j =$ utility$(s_i)/p_i$, in other words, all working segments are MBB.
- Every buyer has spent all her money: $\Sigma_{j \in G} \mathbf{q}_{ij} = \mathbf{e}_i$.
- Every good has been fully sold: $\Sigma_{i \in B} \mathbf{q}_{ij} = \mathbf{p}_j$.

The task is to find the equilibrium prices.

The *working network* is the graph with vertices $B \cup G$ and edge set consisting of the working segments. Let $\mathbf{e}_i^w$ be the money that i spends on her working segments. Let $\mathbf{q}_j^w = \mathbf{p}_j - \Sigma_{s \in F_j}$capacity$(s)$, where F_j is the set of filled segments incident to j; $\mathbf{q}_j^w$ is called the *working price* of j.

Assumption 1.1. *The capacity of each segment, the initial money of each buyer and the utilities of all segments are integral. This can always be achieved with suitable scaling.*

Definition 1.2. An equilibrium is *simple* if the partially, i.e., not fully filled working segments in the equilibrium solution form an acyclic graph. We also call the set of fulfilled and working segments in a simple equilibrium a simple equilibrium set.

Our algorithm will maintain the property that the set of partially filled working segments forms an acyclic graph.

2
A Weakly Polynomial Algorithm

In this chapter, we will describe a weakly polynomial algorithm. Our algorithm is motivated by the following lemma.

Lemma 2.1. *There is a polynomial time algorithm to test if a set of segments is a simple equilibrium segment set.*

Proof. Let W denote the working network. For each $i \in B$, if $s_{ij} \in W$ and $s_{ik} \in W$, utility$(s_{ij})/\mathbf{p}_j$ = utility$(s_{ik})/\mathbf{p}_i$. For each connected component C in W, $\Sigma_{i \in C \cap B} \mathbf{e}_i^w = \Sigma_{j \in C \cap G} \mathbf{p}_j^w$. The RHS can be expressed in terms of a single price and consequently the value of all the $\mathbf{p}_j^w$ can be deduced and so can the price vector. If a good j has no working segment incident to it, then $\mathbf{p}_j$ equals the sum of the spending on all the fulfilled segments incident to j.

We conclude that if we identified all the filled and working segments, we could derive the equilibrium prices. Further, given a candidate collection of filled and working segments, we could check whether the equilibrium has been reached, namely assuming the candidate collection yields an equilibrium, we compute the equilibrium prices as just described and check that the resulting prices yield spending that matches the capacity constraints of the candidate collection. □

To specify our algorithm, we need the following definition.

Definition 2.1. Let $\Delta_0, \mathbf{q}_0, \mathbf{p}_0$ represent the initial $\Delta, \mathbf{q}, \mathbf{p}$ respectively. A spending $\mathbf{q}$ is Δ-feasible if

1. $\forall$ i $\in$ B, $c_i(\mathbf{q}) \geq 0$;

2. $\forall$ j $\in$ G, if $\mathbf{p}_j > \mathbf{p}_j^0$, then $b_j(\mathbf{p}, \mathbf{q}) \geq d_j \Delta$;

3. All spending is on MBB and fulfilled segments, and for all i, j, $\mathbf{q}_{ij}$ is a multiple of Δ;

4. $\forall$ i $\in$ B and $\forall$ j $\in$ G, $\mathbf{q}_{ij} \geq 0$ and $\mathbf{p}_j \geq 0$;

A Δ-feasible spending is Δ-optimal if $\forall j \in G$, $d_j \Delta \leq b_j(\mathbf{p}, \mathbf{q}) \leq (d_j + 1)\Delta$, and $\forall i \in B$, $0 \leq c_i(\mathbf{q}) \leq \Delta$.

We will refer to our algorithm as the *Weakly Scaling algorithm.* The basic idea of the algorithm is to compute a Δ-optimal spending for an initial $\Delta = \Delta_0$, and then for $\Delta_0/2$, $\Delta_0/4$,... until some sufficiently small $\Delta = \Delta_{\text{fin}}$ is reached. Given a Δ-optimal spending, in the Δ scaling phase, it is converted to a $\Delta/2$-feasible spending by reducing the spending on some or all goods by $\Delta/2$. Then the *PriceIncrease* algorithm given below is used to find a $\Delta/2$-optimal spending. Again, the algorithm is initialized with a Δ_0-feasible spending and the *PriceIncrease* algorithm is used to find a Δ_0-optimal spending.

Price Increase Algorithm

The weakly scaling algorithm uses a subroutine, *PriceIncrease*, that takes $\mathbf{p}, \mathbf{q}, \Delta$ as inputs. It uses the following residual working network, $W(\mathbf{p}, \mathbf{q})$, which has vertex set $B \cup G$, forward arcs, from B to G, and backward arcs, from G to B. Let s_{ij} be an MBB segment with range $(a, b]$; then there is a corresponding forward arc with capacity $b - \mathbf{q}_{ij}$, and a backward arc with capacity $\mathbf{q}_{ij} - a$.

Next, we give an example of how to update the capacity in step 7 of $PriceIncrease$. Suppose the range of the new MBB segment is $(b, c]$; then the capacity of the forward arc is set to $c - b$ and the capacity of the backward arc is set to 0.

Rescaling

In the weakly scaling algorithm, to ensure that all spending is in units of Δ, we will round up the segment boundaries to be the multiples of Δ. We call these rounded segments the Δ-*shift* segments.

Given that the capacities of all the Δ-shift segments are multiples of Δ, one can easily verify that the $PriceIncrease$ algorithm preserves Δ-feasibility.

After finding the Δ-optimal spending, the scaling algorithm will use another subroutine, *Rescale*, to halve Δ and find a new Δ-feasible spending.

Changing the segments from Δ-shift to $\Delta/2$-shift segments could result in a segment with a lower utility to price ratio being assigned spending. This would violate our invariants regarding the segments. To restore the invariants, we will return each $\Delta/2$ spending that is now on a sub-MBB segment. In order to have this money available as needed, good j needs a reserve of $d_j\Delta/2$ money for this phase, and this is why they are oversold by $d_j\Delta$ at the start of the Δ-scaling

Algorithm 1: The $PriceIncrease(p, q, \Delta)$ algorithm.

Compute a Δ-optimal spending $\mathbf{q}$ at prices $\mathbf{p}$
while *the spending is not* Δ*-optimal* **do**
  for *each* $i \in B$ *with* $c_i(\mathbf{q}) > \Delta$ **do**
    let R be i's reachable set in $W(\mathbf{p}, \mathbf{q})$
    increase the prices of goods in R in proportion until:
    if *a new MBB segment* s_{ij} *appears* **then**
      update the capacity on the edge (i, j) in $W(\mathbf{p}, \mathbf{q})$
    if *there exists a good* j *such that* $b_j(\mathbf{p}, \mathbf{q}) = d_j\Delta$ **then**
      perform an augmentation by one Δ unit from i to j
  for *each* $g \in G$ *with* $b_g(\mathbf{p}, \mathbf{q}) > (d_g + 1)\Delta$ **do**
    let R be g's reachable set in $W(\mathbf{p}, \mathbf{q})$
    increase the prices of goods in R in proportion until:
    if *a new MBB segment* s_{ij} *appears* **then**
      update the capacity on the edge (i, j) in $W(\mathbf{p}, \mathbf{q})$
    if *there exists a good* j *such that* $b_j(\mathbf{q}) = d_j\Delta$ **then**
      perform an augmentation by one Δ unit along the path in $W(\mathbf{p}, \mathbf{q})$ from g to j
Return $\mathbf{p}, \mathbf{q}$

phase.

Initialization

Let $\Delta_0 = \max_{i \in B}\{e_i/2m\}$. Let f_{ij}^1 denote the first non-empty Δ_0-shift segment of f_{ij}. Let $U_{ij} = \text{utility}(f_{ij}^1)$ and $U_{iG} = \Sigma_{j \in G} U_{ij}$. If $\mathbf{e}_i < 2m\Delta_0$, we may have to keep buyer i from participating in the algorithm initially. She will not participate until $\mathbf{e}_i \geq 2m\Delta$. In addition, we set the initial prices low enough to ensure that every good can be fully oversold by $d_i\Delta_0$ (and for goods that no participating buyer desires, this means the price is set to zero initially). Setting $p_{ij}^0 = U_{ij}\mathbf{e}_i/2U_{iG}$ would make all the goods MBB for buyer i, and the prices would be low enough that buyer i could afford to buy them all and spend the remaining money to make sure they are oversold by $d_j\Delta_0$. Setting $\mathbf{p}_j^0 = \max_{i \in B}\{U_{ij}\mathbf{e}_i/2U_{iG}\}$ ensures that between them, the participating buyers can afford to buy all the goods in full, and each good j can be oversold by $d_j\Delta_0$. The goods are then allocated arbitrarily to buyers for which they are MBB. However, the spending between i and j could exceed the capacity of f_{ij}^1. Then we need to decrease j's price and retuen the spending to i in a manner analogous to *PriceIncrease* to ensure that all the non-filled segments are MBB.

When a buyer i joins the computation, the algorithm will compute the prices $\mathbf{p}^i$ induced by i and i's spending. Then the price is recomputed as follows: $\mathbf{p}_j \leftarrow \max\{\mathbf{p}_j, \mathbf{p}_j^i\}$ and the spending is adjusted accordingly. If the price of good j is defined by a newly participating buyer i, the spending on j by all the other buyers will be released and j will be oversold by $d_j\Delta$ to i.

Let $W^r(\mathbf{p}, \mathbf{q})$ be the network $W(\mathbf{p}, \mathbf{q})$ with the directions of all the segments reversed.

Algorithm 2: The Initialization algorithm.

```
compute the Δ-shift segments of all the f_ij
while there is a sub-MBB segment s_ij has spending do
    Let R be j's reachable set in W^r(p, q)
    decrease the prices of goods in R in proportion until:
    if there exists a good g such that b_g(p, q) = (d_g + 1)Δ then
        perform an augmentation by one Δ unit along the path from g to i in W^r(p, q)
Return p, q
```

The Scaling Algorithm

Algorithm 3: The Weakly Scaling algorithm.

Let $\Delta = \max_{i \in B}\{e_i/2m\}$; $\mathbf{p} = \max_{i \in B}\{U_{ij}\mathbf{e}_i/2U_{iG}\}$; for each good j over-allocate good j by $d_j\Delta$ to MBB buyers

$(\mathbf{p}, \mathbf{q}) \leftarrow Initialization(\Delta, \mathbf{p}, \mathbf{q})$

compute the Δ-shift segments of all the f_{ij}

$(\mathbf{p}, \mathbf{q}) \leftarrow PriceIncrease(\mathbf{p}, \mathbf{q}, \Delta)$

$E' = \{s_{ij} : \sigma(s_{ij}) > 8mn\Delta\}$

while E' *is not a simple equilibrium segment set* (See Lemma 2.1) **do**

  $(\mathbf{p}, \mathbf{q}) \leftarrow Rescale(\mathbf{p}, \mathbf{q}, \Delta)$

  $\Delta \leftarrow \Delta/2$

  introduce new buyers, update $\mathbf{p}$ and the spending

  $(\mathbf{p}, \mathbf{q}) \leftarrow PriceIncrease(\mathbf{p}, \mathbf{q}, \Delta)$

  $E' \leftarrow \{s_{ij} : \sigma(s_{ij}) > 8mn\Delta\}$

Return E'

3
Correctness and Complexity of the Weakly Scaling Algorithm

Lemma 3.1. *In the* Initialization *procedure, each new buyer causes at most* $4m$ *augmentations to reset the prices for the goods she buys.*

Proof. When a buyer i participates in the market, $2m\Delta \leq \mathbf{e}_i < 4m\Delta$. Thus, the spending returned to the buyer during *Initialization* is at most $4m\Delta$, and $4m$ augmentations suffice. □

We call the execution of the algorithm before Δ is halved in the Weakly Scaling algorithm a Δ-phase.

Lemma 3.2. *When no new buyers are introduced, each* Δ*-phase performs at most* m *augmentations and changes the spending on any segment by at most* $(m + 1/2)\Delta$.

Proof. Define $\Theta(\Delta) = \Sigma_{i \in B} \lfloor c_i((q))/\Delta \rfloor$ and $\Phi(\Delta) = \Sigma_{j \in G} \lfloor b_j(\mathbf{p}, \mathbf{q})/\Delta \rfloor$. At the end of *PriceIncrease* in the previous scaling phase, $\Theta(\Delta) + \Phi(\Delta) = \Sigma_{j \in G} d_j = m$. Note that this equality holds at the start of the current Δ-phase.

After Δ is halved, but before the call to the *PriceIncrease* algorithm, $\Theta(\Delta) + \Phi(\Delta) = 2m$. At the end of *PriceIncrease*, $\Theta(\Delta) + \Phi(\Delta) = m$. Note that each augmentation that starts from a buyer decreases $\Theta(\Delta)$ by 1. When an augmentation starting from a good is performed, $\Phi(\Delta)$ is decreased by at least 1. Thus, the total number of augmentation is at most m.

Since each augmentation changes the spending by Δ, During *PriceIncrease*, the spending changes by at most $m\Delta$. Also, the spending on each segment could change by $\Delta/2$ during rescaling. Therefore, each Δ-phase changes the spending on any segment by at most $(m+1/2)\Delta$. □

Lemma 3.3. *In line 9, each newly introduced buyer induces at most* $(4m - 1)$ *augmentations and changes the spending by at most* $(4m - 1)\Delta$.

Proof. When a buyer i participates in the market, $2m\Delta \leq \mathbf{e}_i < 4m\Delta$. As i will use up $\lfloor \mathbf{e}_i/\Delta \rfloor \Delta$ of her money, she will perform at most $(4m - 1)$ augmentations and change the spending by at most $(4m - 1)\Delta$. □

Definition 3.1. A segment s is Δ-abundant if $\sigma(s) \geq 8mn\Delta$.

Lemma 3.4. *If s is Δ-abundant, then it will continue to be Δ-abundant at the start of each subsequent Δ-phase in the Weakly Scaling algorithm.*

Proof. As s is abundant, $\sigma(s) \geq 8mn\Delta$. By Lemmas 3.2 and 3.3, $\mathbf{q}_{ij}$ is reduced by at most $[(4m-1)(n-1)+m+1/2]\Delta \leq (4m-1)n\Delta$ in each Δ-phase, as there are at most $n-1$ buyers. In each Δ-phase, a working segment loses at most $\Delta/2$ spending due to rounding on the boundary of the segment. Thus the value $\sigma'(s)$ of $\sigma(s)$ at the start of the next Δ-phase is lower bounded as follows: $\sigma(s)' \geq \sigma(s) - (4mn-n)\Delta - \Delta/2 \geq (8mn-4mn)\Delta \geq 8mn(\Delta/2)$. □

Let $U_{\max}$ be the maximum utility of all the segments and let $\delta = 1/n(U_{\max})^n$.

Lemma 3.5. *Suppose that the network of working segments that are not filled is acyclic. Then each segment with non-zero spending will have at least δ spending at the equilibrium.*

Proof. For filled segments, which by assumption have integral capacities, the spending is an integral multiple of δ. For the working segments, let C be a component of W, $i \in C \cap B$ and j, $k \in C \cap G$. If $s_{ij} \in C$ and $s_{ik} \in C$, utility$(s_{ij})/\mathbf{p}_j$ =utility$(s_{ik})/p_k$. By Assumption 1.1, the utilities and capacities of all the segments are integral; thus for each $i \in C \cap B$, $\mathbf{e}_i^w$ is integral. Also, as the sum of the prices for goods in C equals the sum of spending for buyers in G, each price can be written in units of $1/\Pi_{s_{ij} \in C}$ utility(s_{ij}), and for all $i \in C \cap B$, $\mathbf{e}_i^w$ can be written in units of $1/\Pi_{s_{ij} \in C}$utility(s_{ij}). As the working network is acyclic, it must have at least one leaf. Each leaf is either a buyer or a good. If the leaf is good j, the spending on the incident segment is $\mathbf{p}_j^w$ and if the leaf is buyer i, the spending is $\mathbf{e}_i^w$. Note that either of these values can be written in units of $1/\Pi_{s_{ij} \in C}$ utility(s_{ij}), and is therefore at least δ. We then remove the leaf and by processing the reduced graph recursively, we deduce that the claim also applies to the remaining working segments. □

Lemma 3.6. *For $\Delta_k \leq \delta/16mn$, the set of abundant segments form a simple equilibrium segment set and consequently induce an equilibrium pricing.*

Proof. Suppose that the loop in the Weakly Scaling algorithm were run indefinitely (i.e. without a stopping condition). By Lemma 3.2, once $8m\Delta_k \leq (\min \text{segment capacity} - \Delta_k)$, the set of

used segments (fulfilled and working segments) for each (buyer, good) pair will be essentially unchanged; only one segment can enter and leave the set of used segments, possibly repeatedly.

Since the changes to the spending on each segment tends to 0 as $k \to 0$, there is a limit spending on each used segment. Further, this limit spending forms an equilibrium. By Lemma 3.5, each used segment in this equilibrium has at least δ spending. Therefore, these segments are abundant once $\Delta_k \leq \delta/(16mn)$, and further no other segments will become abundant. Thus, for $\Delta_k \leq \delta/(16mn)$, the set of abundant segments E' is a simple equilibrium segment set (c.f. Definition 1.2 & Lemma 2.1). □

Lemma 3.7. *Each augmentation can be implemented in $O([n \log n + m])$ and each Δ-phase can be implemented in $O(m[n \log n + m])$ time if no new buyers are introduced.*

Proof. By Lemma 3.2, in each Δ-phase, there are at most m augmentations. In the remainder of this proof, we show how to implement a single augmentation in $O(n \log n + m)$ time, yielding the bound in the lemma.

We show how to find an augmentation for a buyer i who has Δ or more money to spend. The method is to raise the prices of the goods reachable from i in the residual working network, stopping when one of the goods, j say, in its reachable set has b_j reduce to 0. At this point we can perform an augmentation from i to j (which increases b_j to Δ and increases i's spending by Δ).

Maintaining the reachable set and identifying when $b_j = 0$ entails maintaining a set of critical values at which (i) the reachable set grows, or (ii) $b_j = 0$.

Note that the prices of the reachable goods are being increased by a common multiplier x, so we can express the time at which events (i) and (ii) occur in terms of values of x.

The reachable set grows when a new edge becomes MBB. For each component C outside the current reachable set R, there is one segment from R to C that will be the first to become MBB. The value of x at which this segment becomes critical is the critical value for C.

For each good j in R we need to keep the value of x at which $b_j = 0$.

The above critical values are stored in a Fibonacci heap F.

The next event of type (i) or (ii) is found by a delete min on F. If the event is the discovery of a new MBB edge, this causes R to grow, with two effects: the new outgoing edges from R need

to be considered as they may reduce the critical values for some component C still outside R. Over the course of the processing for this augmentation each edge induces at most one decrease key, for a total of at most m decrease keys.

For the goods newly added to R, their critical values need to be added to F, at a cost of $O(\log n)$ per good. This is a total cost of $O(n \log n)$.

There are at most n deletions, costing $O(n \log n)$ in total. Thus, the overall cost is $O(n \log n + m)$, as claimed.

Lastly, the augmentation itself is performed by traversing the path from i to j in R. As R is acyclic, it has size $O(n)$ and this traversal can be carried out in $O(n)$ time. Moreover, checking whether a new abundant segment emerges takes time $O(m)$.

□

Theorem 3.2. *The Weakly Scaling algorithm has at most* $O(\log(\mathbf{e}_{\max}) + \log n + n \log(U_{\max}))$ Δ*-phases and takes* $O(m(n \log n + m)[\log(\mathbf{e}_{\max}) + \log n + n \log(U_{\max}) + n])$ *time.*

Proof. The first claim follows from the fact that $\Delta_0 = \max_{i \in B}\{\mathbf{e}_i/2m\} \leq \mathbf{e}_{\max}/m$ and the final value of Δ, Δ_{fin} satisfies $\Delta_{\text{fin}} \geq 1/16mn^2U_{\max}^n$. The second claim follows from Lemmas 3.2 and 3.7. In addition, by Lemmas 3.3 and 3.1, each buyer will induce another $O(m)$ augmentations, which take time $O(m[n \log n + m])$ per buyer. □

4
A Strongly Polynomial Algorithm

Lemma 3.4 guarantees that once a segment becomes abundant, it will remain abundant. However, this depends on repeatedly halving Δ; our new algorithm needs not follow this sequence of halvings. Nonetheless, we will ensure that once a segment becomes abundant it will remain abundant. Thus, our goal will be to incrementally identify all the abundant segments (See also Lemmas 2.1 and 3.4). In this chapter, we will give a strongly polynomial algorithm to do this.

It will be convenient to have $\Delta_0 = 2^i$ for some integer i. This is achieved by rounding down $\max_{i \in B} e_i/2d_i$ to the nearest integer power of 2. As we will be doing this rounding repeatedly, we define $\mathrm{rd}(x) = 2^i$, where $2^i \leq x < 2^{i+1}$, i an integer. Our algorithm will maintain the following property.

Property 4.1. *For each segment s, there is a critical scaling factor Δ_s such that s is introduced at the start of the Δ_s-scaling phase. For each buyer b, we also define a critical scaling factor $\Delta_b = rd(\mathbf{e}_b/2m)$, which indicates the phase at which b is introduced.*

Consider a segment s with range $(a, b]$. Suppose that $(j-1)2^i \leq a < j \cdot 2^i \leq b < (j+1)2^i$, where j is an odd integer. Then $\Delta_s = 2^i$. The rounding of s will expand s at its left boundary and potentially shrink it at its right boundary, so that at the Δ-phase its boundaries take the nearest integer multiple of Δ.

To calculate Δ_s, we proceed as follows. We define $a' = a/\Delta_0$, $b' = b/\Delta_0$. Then find the binary representation of a' and b'. Let i be the first bit in which a' and b' disagree. If i is in the integral part ($i > 0$), then $k = 0$; otherwise ($i < 0$), $k = -i$. To find this i, compute $c = a' \oplus b'$ and then $i = \lfloor \log c \rfloor$.

We also need to modify the definition of abundance, Definition 3.1, for the strongly polynomial algorithm.

Definition 4.1. A segment is Δ-short if it has been introduced by the start of the Δ-scaling phase and its capacity is less than $\Delta/2$. A segment is Δ-long if it has been introduced by the start of Δ-scaling phase and its capacity is at least $\Delta/2$.

In the algorithm, we first identify all the critical scaling factors and sort them in descending

order. If Δ_k is a critical scaling factor and Δ_{k+1} is the next one in sorted order, when we are at the Δ_k-scaling phase and Δ_{k+1} is not far from Δ_k, specifically, $\Delta_k/\Delta_{k+1} \leq 512m^2n^2$, we will run the Weakly Scaling algorithm, reaching the Δ_{k+1}-scaling phase in $O(\log n)$ scaling phases. Otherwise, for each edge, as we shall see, the sequence of segments has the property that every Δ_k-short segment is adjacent to Δ_k-long segments. For each Δ_k-long segment, following $O(\log n)$ scaling phases after its introduction, it will either be abundant or never become fully filled. If one segment cannot be fully filled, then all the subsequent segments of the same edge will not be assigned any spending, and then there is no need to consider the subsequent segments further.

When Δ_{k+1} is far from Δ_k, between the Δ_k-scaling phase and the Δ_{k+1}-scaling phase, no new segments are introduced. We will then use the *FindNext* approach, as in Orlin's method, to either find a new abundant segment or to quickly reach Δ_{k+1}. In the former case, for each edge, we will need to consider at most 2 segments, one short and one long. One of them will become abundant within $O(\log n)$ phases.

Let $A(\Delta)$ denote the set of Δ-abundant working segments. We call the network defined by $B \cup G$ and $A(\Delta)$ the Δ-network. Let $N(\mathbf{p}, \mathbf{s})$ be the network with MBB segments as forward arcs and the abundant working segments as backward arcs, without rounding.

Let $C \subset B \cup G$. Then the surplus of C is defined as $s(C, \mathbf{p}) = \Sigma_{i \in B \cap C} \mathbf{e}_i^w - \Sigma_{j \in G \cap C} \mathbf{q}_j^w$. A component C of the Δ-network is Δ-*fertile* if either (1) C consists of a single buyer i from B and $\mathbf{e}_i \geq \Delta/4n$, or (2) $C \cap G \neq \varnothing$ and $s(C, \mathbf{p}) \leq -\Delta/4n$. A Δ-network is Δ-fertile if any of its components is Δ-fertile. Procedure *FindNext* will either make the Δ-network fertile, thereby ensuring that a new abundant segment will emerge in the next $O(\log n)$ scaling phases, or it will update Δ to Δ_u, which is $512m^2n^2\Delta_{k+1}$ (see line 14 of the *Strongly Polynomial* algorithm, and lines 11 and 12 of the *FindPrice* algorithm).

The strongly polynomial algorithm follows.

The FindNextDelta and FindNext Procedures

The goal of these procedures is to make the abundant network fertile. If any abundant component has a large negative surplus, then the abundant network is in fact already fertile. Otherwise, we will be increasing prices in a suitable abundant component C, which we call the *root* component, and in all the other abundant components reachable from C, so as to make one of these compo-

Algorithm 4: The strongly polynomial algorithm.

For each segment s and buyer b, compute the critical scaling factor and sort all factors in descending order $\Delta_1 \geq \Delta_2 \geq \ldots \geq \Delta_{r+|B|}$

Initialize $k = 0$

Let $\Delta = \max_{i \in B}\{e_i/2m\}$; $\mathbf{p} = \max_{i \in B}\{U_{ij}\mathbf{e}_i/2U_{iG}\}$; for each good j over-allocate good j by $d_j\Delta$ to MBB buyers

$(\mathbf{p}, \mathbf{q}) \leftarrow Initialization(\Delta, \mathbf{p}, \mathbf{q})$

$\Delta \leftarrow \Delta_0$

$(\mathbf{p}, \mathbf{q}) \leftarrow PriceIncrease(\mathbf{p}, \mathbf{q}, \Delta)$

$E' = \{s_{ij} : \sigma(s_{ij}) > 32mn\Delta\}$

while E' *is not a simple equilibrium segment set and* $k \leq (r + |B|)$ **do**

  if $\Delta/\Delta_{k+1} > 512m^2n^2$ *and the* Δ*-network is not fertile* **then**

    $\Delta_{\max} \leftarrow FindNextDelta(\mathbf{p}, \mathbf{q}, \Delta)$

    if $\Delta_{\max}/512m^2n^2 > \Delta_{k+1}$ **then**

      $\Delta_u \leftarrow \Delta_{\max}$

    else

      $\Delta_u \leftarrow 512m^2n^2\Delta_{k+1}$

    $(\mathbf{p}, \mathbf{q}, \Delta') \leftarrow FindNext(\mathbf{p}, \mathbf{q}, \Delta_u, \Delta)$

    $\Delta \leftarrow \Delta'$

    $E' \leftarrow \{s_{ij} : \sigma(s_{ij}) > 32m\Delta\}$

  else

    $(\mathbf{p}, \mathbf{q}) \leftarrow Rescale(\mathbf{p}, \mathbf{q}, \Delta)$

    $\Delta \leftarrow \Delta/2$

    while $\Delta \leq \Delta_{k+1}$ **do**

      $k \leftarrow k + 1$ (update k)

    introduce new buyers and update $\mathbf{p}$

    $(\mathbf{p}, \mathbf{q}) \leftarrow PriceIncrease(\mathbf{p}, \mathbf{q}, \Delta)$

    $E' \leftarrow \{s_{ij} : \sigma(s_{ij}) > 32m\Delta\}$

Return E'

nents fertile, component D say. We will stop the price increase at a point when $s(C, \mathbf{p})$ is still large enough to cover all the negative surplus of the components reachable from C, and yet we can guarantee that there is a reachable component D which is fertile.

Algorithm 5: The $FindNext(\mathbf{p}, \mathbf{q}, \Delta_u, \Delta)$ algorithm.

1 $(\mathbf{p}, \mathbf{q}, \Delta') \leftarrow FindPrice(\mathbf{p}, \mathbf{q}, \Delta_u, \Delta)$

2 $(\mathbf{p}, \mathbf{q}) \leftarrow PriceDecrease(\mathbf{p}, \mathbf{q}, \Delta')$

3 $(\mathbf{p}, \mathbf{q}) \leftarrow PriceIncrease(\mathbf{p}, \mathbf{q}, \Delta')$

4 Return $(\mathbf{p}, \mathbf{q}, \Delta')$

The strongly polynomial algorithm runs the two subroutines *FindNextDelta* and *FindNext* in turn. *FindNextDelta* finds a new smaller value for Δ at which some component D is fertile. *FindNext* begins by calling *FindPrice*, which finds prices that are close to the Δ-optimal prices $\mathbf{p}$; then it finds the Δ-optimal prices by applying *PriceDecrease* and *PriceIncrease* in turn.

We will be computing the spending on the segments in *FindNextDelta* and *FindPrice* as a linear function of the multiplier x by which the prices have been increased. The spending is computed as follows: we first assign spending equal to the negative surpluses via the segments connecting the root component C to each reachable component with negative surplus. This can be done by sending money from C along the MBB segments. If the MBB segments between the components form a cycle, we break the cycle by ordering the segments and disregard the segments with lower priority until the higher priority segments become fully filled.

We then assign spending within the abundant components. The idea is to spend as much money as possible on each good, and thus in each component $\tilde{C}$ to leave unspent money equal to the surplus of the components reachable from $\tilde{C}$. Each leaf good will receive spending from the parent buyer equal to its working price. Each leaf buyer will spend as much of her money on the parent goods as she can. Then these leaves are removed and the process is iterated.

Once the spending on segment s reaches its capacity, s will be filled and the capacity will be deducted from the working money of s's incident buyer and the working price of s's incident good. If as a result a root component splits into two or more components, we let C' denote the component that s points from, and keep increasing the prices of goods that are reachable from

Algorithm 6: The $FindNextDelta(\mathbf{p}, \mathbf{q}, \Delta)$ algorithm.

for *each abundant component* C *in the* Δ*-network* **do**
  Let $\mathbf{p}^l = \mathbf{p}$
  let v be any buyer in C and let R be v's reachable set in $N(\mathbf{p}^l, \mathbf{q})$
  increase local prices $\mathbf{p}^l$ of goods in R in proportion and compute the corresponding spending, until:
  if *a new segment becomes MBB* **then**
    update $N(\mathbf{p}^l, \mathbf{q})$ and R, and go to Step 4
  if *the spending on a segment equals its capacity* **then**
    remove the segment from $N(\mathbf{p}^l, \mathbf{q})$, update C, the working price and working money, and go to Step 3
  if $s(C, \mathbf{p}^l) = 0$ *or* $s(D, \mathbf{p}^l) \leq -s(C, \mathbf{p}^l)/(4n)$ *for some component* D *in* C*'s reachable set in the abundant network* **then**
    let $\Delta^C = s(C, \mathbf{p}^l)$ and stop increasing
$\Delta_{\max} \leftarrow \max_C \Delta^C$
Return $\Delta_{\max}$

Algorithm 7: The $FindPrice(\mathbf{p}, \mathbf{q}, \Delta_u, \Delta)$ algorithm.

for *each abundant component* C *in the* Δ*-network* **do**
  Let $\mathbf{p}^C = \mathbf{p}$
  while $s(C, \mathbf{p}^C) > \Delta_u$ **do**
    let v be a buyer in C and let R be v's reachable set in $N(\mathbf{p}, \mathbf{q})$
    increase prices $\mathbf{p}^C$ in R in proportion, until:
    if *a new segment becomes MBB* **then**
      update $N(\mathbf{p}, \mathbf{q})$ and R, and go to Step 4
    if *the spending on a segment equals its capacity* **then**
      remove the segment from the abundant network, update C, the working price and money, and go to Step 3
For each good $i, p_i \leftarrow \max_C p_i^C$
$\tilde{\Delta} \leftarrow \Delta_u / 32mn$
$\Delta' \leftarrow \mathrm{rd}(\tilde{\Delta})$
Return $(\mathbf{p}, \mathbf{q}, \Delta')$

C' (this is the updated C in line 8 of *FindNextDelta* and line 9 in *FindPrice*).

PriceDecrease

After spending as much money as possible on each good, we need to adjust the spending and possibly the prices to ensure, (1) no good is undersold, (2) once we revert to Δ-shift segments, the spending on each segment is a multiple of Δ. Moreover, we will need to restore the reserve of each good as in the Weakly Scaling algorithm. To accomplish these goals, we will use a subroutine *PriceDecrease*, similar to *PriceIncrease*.

Definition 4.2. A segment s is Δ-*unrounded* if the spending is not a multiple of Δ. A good j is Δ-*starved* if $b_j(\mathbf{p}, \mathbf{q}) < d_j\Delta$.

Algorithm 8: The $PriceDecrease(\mathbf{p}, \mathbf{q}, \Delta')$ algorithm.

1 Compute the Δ'-shift segments of all the working segments
2 **while** *there is a* Δ'*-unrounded segment* s_{ij} **do**
3 | return the overspent part from good j to buyer i
4 **while** *there is a* Δ'*-starved good* j **do**
5 | Let R be j's reachable set in $W^r(\mathbf{p}, \mathbf{q})$
6 | decrease the prices of goods in R in proportion until:
7 | **if** *a filled segment* s_{bg} *becomes MBB* **then**
8 | | update the capacity on edge (b, g)
9 | **if** *there exists a good* g *such that* $b_g(\mathbf{p}, \mathbf{q}) = (d_g + 1)\Delta'$ **then**
10 | | perform an augmentation by one Δ' unit along the path from g to j in $W^r(\mathbf{p}, \mathbf{q})$
11 Return $(\mathbf{p}, \mathbf{q})$

The updating capacity process on line 8 is similar to that in *PriceIncrease* but the new MBB segment is filled here whereas it is empty in *PriceIncrease*.

5
Correctness of the Strongly Polynomial Algorithm

To facilitate our proof, let $\mathbf{p}$ denote the prices when *FindNext* is called and Δ be the scaling factor at the start of the while loop iteration. We call each for loop iteration in *FindNextDelta* and *FindPrice* a *PushPrice* procedure for the corresponding component and let $\mathbf{p}^C$ be the prices induced by each component C's *PushPrice* procedure. Let $\mathbf{p}'$ be the prices output by *FindPrice*. $\mathbf{p}'_C$ is $\mathbf{p}'$ restricted to a component C. Let Δ' be the output scaling factor of *FindPrice*. Finally, let σ' denote the spending after *FindPrice*, and let σ denote the spending at the start of *FindPrice*. Note that when a new buyer b participates the market, $e_b \geq 2m\Delta$, and the network is already fertile.

In analyzing the period between the Δ and Δ_{k+1} scaling phases, there are four cases to consider:

Case 1: $\Delta/\Delta_{k+1} \leq 512m^2n^2$.

The next critical scaling factor Δ_{k+1} is reached after $O(\log n)$ scaling phases.

Case 2: $\Delta/\Delta_{k+1} > 512m^2n^2$ and the network is not fertile at the start of the while iteration, and $\Delta_{\max}$, the output of *FindNextDelta* satisfies $\Delta_{\max}/\Delta_{k+1} > 512m^2n^2$. Note that in this case, $\Delta_u = \Delta_{\max}$ (see line 12 of the *Strongly Polynomial* algorithm) and consequently $\Delta' = \mathrm{rd}(\Delta_{\max}/32mn)$ (see lines 11 and 12 of the *FindPrice* algorithm).

A new abundant segment will emerge in $O(\log n)$ scaling phases. Our proof of this is divided into four parts.

Let $\bar{\Delta} = \Delta'/4mn$.

1. (Lemma 5.6) Any abundant segment remains abundant.

2. (Corollary 5.1) By the end of the $\bar{\Delta}$-scaling phase, each Δ_k-short segment is adjacent to Δ_k-long segments.

3. (Lemma 5.9) Any Δ_k-long segment is either $\bar{\Delta}$-abundant or it never becomes filled subsequently.

4. (Lemma 5.13) A new abundant segment will emerge by the end of the $\bar{\Delta}$-scaling phase.

Case 3: $\Delta/\Delta_{k+1} > 512m^2n^2$ and the network is not fertile at the start of the while iteration, and $\Delta_{\max}$, the output of *FindNextDelta*, satisfies $\Delta_{\max}/\Delta_{k+1} \leq 512m^2n^2$.

Δ_u is set to $512m^2n^2\Delta_{k+1}$. Following this, after another $O(\log n)$ scaling phases, we will reach the next critical scaling phase, namely the Δ_{k+1}-scaling phase. We also have to show that our method for updating the spending maintains the abundance of the abundant segments, which we do in Lemma 5.14.

Case 4: $\Delta/\Delta_{k+1} > 512m^2n^2$ and the network is already fertile at the start of the while iteration.

Let $\hat{\Delta} = \Delta/64m^2n^2$. Then, as we show in Lemma 5.15, by the end of the $\hat{\Delta}$-scaling phase, a new abundant segment will emerge.

Case 2, Part I

It is convenient to extend the notation $\mathbf{e}_i$, $\mathbf{p}_j$, c_i and b_j to sets of buyers and goods respectively, writing $\mathbf{e}_B = \sum_{i\in B} \mathbf{e}_i$, $\mathbf{p}_G = \sum_{j\in G} \mathbf{p}_j$, $c_C = \sum_{i\in C\cap B} c_i$ and $b_C = \sum_{j\in C\cap G} b_j$.

Lemma 5.1. *When* FindNext *is called,* $\mathbf{e}_B - \mathbf{p}_G \leq 2m\Delta$.

Proof. Note that in this case no new buyers are introduced. A participating buyer i holds up to Δ unspent money at the end of the Δ-scaling phase, and these buyers contribute at most $\Sigma_{i\in B_1}\Delta$ to $\mathbf{e}_B - \mathbf{p}_G$. Each good j can be oversold by at most $(d_j+1)\Delta$, so all the goods together contribute at most $\Sigma_{j\in G}(d_j+1)\Delta$ to $\mathbf{e}_B - \mathbf{p}_G$. Thus, $\mathbf{e}_B - \mathbf{p}_G \leq (|B| + \Sigma_{j\in G}(d_j+1))\Delta \leq (|B| + m + |G|)\Delta \leq 2m\Delta$.

□

Lemma 5.2. $\Delta_{\max} \leq 4m\Delta$ *and* $\Delta' \leq \Delta/8n$.

Proof. By Lemma 5.1, before calling *FindNext*, the available money $\mathbf{e}_B - \mathbf{p}_G \leq 2m\Delta$. The available money at the end of *FindPrice* is at least $\Delta_{\max} - n\Delta_{\max}/4n \geq \Delta_{\max}/2$. Thus $\Delta_{\max}/2 \leq 2m\Delta$ and so $\Delta_{\max} \leq 4m\Delta$. Recall that $\Delta' = \mathrm{rd}(\Delta_{\max}/32mn)$; thus $\Delta' \leq 4m\Delta/32mn \leq \Delta/8n$. □

Lemma 5.3. *Each call to* PriceDecrease *performs at most* $2m$ *augmentations and increases the surplus of the whole network by at most* $3m\Delta'$.

Proof. At the end of *FindPrice*/start of *PriceDecrease*, each good is fully sold. Each good j is incident to at most d_j unrounded segments, which causes a return of at most $d_j\Delta'$ spending. Further, each good needs to be oversold by $d_j\Delta'$, so the total number of augmentations during *PriceDecrease* is at most $\Sigma_{j\in G}(d_j+d_j) = 2m$. Within each augmentation, the total price decrease for a good is at most Δ' and over the course of the *PriceDecrease* process each price may decrease by an additional amount strictly upper bounded by Δ'. Thus, the total price decrease during *PriceDecrease* is at most $(2m + |G|)\Delta' < 3m\Delta'$. □

Let α denote the available money $\mathbf{e}_B - \mathbf{p}'_G$ after *FindPrice*.

Lemma 5.4. $|\sigma'(s_{ij}) - \sigma(s_{ij})| \leq 5m\Delta - \alpha$ *for* $s_{ij} \in A(\Delta)$*, the set of* Δ*-abundant segments.*

Proof. Let's first consider a flow decomposition. Let $\hat{\sigma}$ be the unrounded spending and $\hat{\mathbf{q}}$ the corresponding spending on the edges which is obtained from $\mathbf{p}^w$ and which satisfies the following:

1. if s_{ij} is a working segment and $s_{ij} \notin A(\Delta)$, then $\hat{\sigma}(s_{ij}) = 0$;
2. if $s(C, \mathbf{p}) \geq 0$, then $c_C(\hat{\mathbf{q}}) = s(C, \mathbf{p})$ and $b_C(\mathbf{p}, \hat{\mathbf{q}}) = 0$;
3. if $s(C, \mathbf{p}) < 0$, then $b_C(\mathbf{p}, \hat{\mathbf{q}}) = s(C, \mathbf{p})$ and $c_C(\hat{\mathbf{q}}) = 0$.

Let $\gamma = \sigma - \hat{\sigma}$.

As the spending when *FindNext* is called is not Δ-fertile, $s(C, \mathbf{p}) \geq -\Delta/4n$ and so $b_C(\mathbf{p}, \hat{\mathbf{q}}) \geq -\Delta/4n$. We show the following claims:

1. $|\gamma(s_{ij})| \leq 3m\Delta$.

   $\hat{\sigma}$ differs from σ due to the following factors: (1) up to Δ unspent for each buyer i; (2) up to $(d_j + 1)\Delta$ overspending on each good j; (3) the negative surpluses of abundant components; (4) rounding on the working segments. Thus, $|\gamma(s_{ij})| \leq [\Sigma_{i\in B}1 + \Sigma_{j\in G}(d_j + 1) + n/4n + m]\Delta \leq (|B| + m + |G| + 1/4 + m)\Delta \leq 3m\Delta$. This difference was brought about by augmentations (including spending returns) which change the spending on any abundant segment by at most $3m\Delta$.

2. $|\sigma'(s_{ij}) - \hat{\sigma}(s_{ij})| \leq 2m\Delta - \alpha$.

   The calculation of σ' from the spending being computed in *FindNextDelta* and *FindPrice* is similar to how $\hat{\sigma}$ is obtained from $\mathbf{p}^w$ (except that there is spending on the non-fertile MBB segments between abundant components). By Lemma 5.1, before the call to *FindNext*, $\mathbf{e}_B - \mathbf{p}_G \leq 2m\Delta$, and as the available money after calling *FindNext*, $\mathbf{e}_B - \mathbf{p}'_G = \alpha$, $\Sigma_{j \in G}\mathbf{p}_j \leq \Sigma_{j \in G}\mathbf{p}'_j \leq \Sigma_{j \in G}\mathbf{p}_j + (2m\Delta - \alpha)$, and so $|\sigma'(s_{ij}) - \hat{\sigma}(s_{ij})| \leq 2m\Delta - \alpha$.

Therefore, if $s_{ij} \in A(\Delta)$, $|\sigma'(s_{ij}) - \sigma(s_{ij})| \leq |\sigma'(s_{ij}) - \hat{\sigma}(s_{ij})| + |\hat{\sigma}(s_{ij}) - \sigma(s_{ij})| \leq 5m\Delta - \alpha$. □

Lemma 5.5. *The execution of* FindNext *changes the spending on each abundant segment by at most* $6m\Delta$.

Proof. Recall that by Lemma 5.2, $\Delta' \leq \Delta/8n$. By Lemma 5.3, *PriceDecrease* performs at most $2m$ augmentations. Therefore, after *PriceDecrease*, the spending on each abundant segment decreases by at most $2m\Delta' < m\Delta/4$. Also, the available money before calling *PriceDecrease* is $\alpha = \mathbf{e}_B - \mathbf{p}'_G$, and by Lemma 5.3, *PriceDecrease* increases the surplus of the whole network by at most $3m\Delta' \leq 3m\Delta/8$. Hence, *PriceIncrease* can remove surplus of at most $3m\Delta/8 + \alpha$. Note that *PriceDecrease* only increases the surplus and *PriceIncrease* only decreases it. Thus, by Lemma 5.4, at the end of *PriceIncrease*, compared to the spending before calling *FindNext*, the spending on each abundant segment changes by at most $5m\Delta - \alpha + m\Delta/4 + 3m\Delta/8 + \alpha \leq 6m\Delta$. □

Lemma 5.6. *Each* $s_{ij} \in A(\Delta)$ *remains abundant after* FindNext.

Proof. Recall that by Definition 3.1, $\sigma(s_{ij}) \geq 8mn\Delta$ if $s_{ij} \in A(\Delta)$. Then by Lemma 5.5, the spending at the end of *PriceIncrease* is at least $8mn\Delta - 6m\Delta > 2m\Delta$. By Lemma 5.2, $\Delta \geq 8n\Delta'$, thus the spending at the end of *FindNext* is at least $2m\Delta \geq 16mn\Delta' > 16m\Delta'$. □

Case 2, Part II

Lemma 5.7. *If* $\Delta_k/\Delta_{k+1} > 2$*, then during every* Δ_α*-scaling phase with* $\Delta_k \geq \Delta_\alpha > \Delta_{k+1}$*, every* Δ_k*-short segment is adjacent to* Δ_k*-long segments.*

Proof. We will show that at the start of the Δ_k-scaling phase, no two Δ_k-short segments are adjacent to each other; and as no new segments appear until the Δ_{k+1}-scaling phase, every

Δ_k-short segment must be adjacent to Δ_k-long segments.

Suppose that at the start of the Δ_k-scaling phase, a Δ_k-short segment s_1 with range $(a_1, b_1]$ is adjacent to another Δ_k-short segment s_2 with range $(a_2, b_2]$, and $a_2 \geq b_1$. As s_1 and s_2 are introduced by the start of the Δ_k-scaling phase, for some integer j, $(j-1)\Delta_k < a_1 < j\Delta_k \leq b_1 \leq a_2 < (j+1)\Delta_k \leq b_2$. By Definition 4.1, $b_1 - a_1 < \Delta_k/2$ and $b_2 - a_2 < \Delta_k/2$. Therefore, there must exist a segment s_3 with range $(a_3, b_3]$ and $b_1 \leq a_3 < (j+1/2)\Delta_k \leq b_3 \leq a_2$. Consequently, s_3 will be introduced at the start of the $\Delta_k/2$-scaling phase. This contradicts the assumption that $\Delta_k > 2\Delta_{k+1}$. □

Corollary 5.1. *At the end of the $\bar{\Delta}$-scaling phase, every Δ_k-short segment is adjacent to Δ_k-long segments.*

Proof. Recall that $\bar{\Delta} = \Delta'/4mn = \mathrm{rd}(\Delta_{\max}/32mn)/4mn > \Delta_{\max}/512m^2n^2 \geq \Delta_{k+1}$. □

Case 2, Part III

Lemma 5.8. *When calling* FindNext*, if the unspent part of a segment s is at least $8mn\Delta$, then at the end of* FindNext*, the unspent part is at least $8mn\Delta'$.*

Proof. Recall that by Lemma 5.2, $\Delta' \leq \Delta/8n$. When calling the *FindNext* subroutine, the segment might be abundant or not. If s is abundant, then by Lemma 5.5, the spending changes at most $6m\Delta$, and by Lemma 5.2, $\Delta \geq 8n\Delta'$; thus the unspent part is at least $8mn\Delta - 6m\Delta = 2m\Delta > 2m \cdot 8n\Delta' > 8mn\Delta'$.

The spending on a segment s is unchanged from the start of the while loop iteration of the strongly polynomial algorithm up to the point *FindNext* is called. Thus, if the spending on s is not abundant when calling *FindNext*, then it connects abundant components. As in this case (Case 2) the abundant network is not fertile, each component C has $s(C, \mathbf{p}) > -\Delta/4n$. Thus, the spending computed during *FindPrice* satisfies $\sigma(s) \leq n\Delta/4n = \Delta/4$. By Lemma 5.3, *PriceDecrease* performs at most $2m$ augmentations. Therefore, after *PriceDecrease*, the spending on each abundant segment increases by at most $2m\Delta' \leq m\Delta/4$. By Lemma 5.1, the available money before calling *PriceDecrease* is at most $2m\Delta$, and by Lemma 5.3, *PriceDecrease* increases the surplus of the whole network by at most $3m\Delta' \leq 3m\Delta/8$. Hence *PriceIncrease* can remove surplus of at most $3m\Delta/8 + 2m\Delta$. Thus, by Lemma 5.4, at the end of *PriceIncrease*, the spending

on each abundant segment is at most $\Delta/4 + m\Delta/4 + 3/8m\Delta + 2m\Delta < 3m\Delta$. As when calling *FindNext*, the unspent part on s is at least $16m\Delta$, the capacity of s is at least $16m\Delta$. At the end of *FindNext*, the unspent part is at least $13m\Delta > 8mn\Delta'$. □

Lemma 5.9. *At the start of the $\bar{\Delta}$-scaling phase, any Δ_k-long segment is either $\bar{\Delta}$-abundant or it never becomes fully filled.*

Proof. $\Delta \leq \Delta_k$. Also, recall that by Lemma 5.2, $\Delta/\Delta' \geq 8n$, and $\bar{\Delta} = \Delta'/4mn$; thus $\Delta_k \geq \Delta > 32mn^2\bar{\Delta}$. By Definition 4.1, if a segment s is Δ_k-long, then its capacity is at least $\Delta_k/2 > 16mn^2\bar{\Delta}$. Also, as the Interest Network is a simple bipartite graph, $m \leq n^2/4$.

If the spending on s, $\sigma(s) \geq 8mn\bar{\Delta}$, then s is abundant. Otherwise, the unspent part of s is greater than $(16mn^2 - 8mn)\bar{\Delta} > 8mn\bar{\Delta}$. We will show that at the start of each subsequent Δ_λ-scaling phase, the unspent part of s is always at least $8mn\Delta_\lambda$.

As in the Weakly Scaling algorithm, at the end of a scaling phase, the scaling factor drops from Δ to $\Delta/2$. Thus we need to show that at the end of the Δ-scaling phase, the unspent part is at least $8mn\Delta/2 = 4mn\Delta$. By Lemmas 3.2 and 3.3, and taking of a possible Δ change due to rounding, the change in spending on each segment is at most $[(4m-1)(n-1)+m+1/2+1]\Delta \leq (4m-1)n\Delta$; thus, the unspent part is at least $[8mn - (4m-1)n]\Delta > 4mn\Delta$.

During the execution of *FindNext*, the scaling factor drops from Δ to Δ', then we need to show that at the end of *FindNext*, the unspent part is at least $8mn\Delta'$. This is shown by lemma 5.8. □

Case 2, Part IV

If a segment becomes filled during *FindNext*, it has to be removed. If the segment is within the abundant component, its removal splits its component into two parts, and if there is no other unfilled MBB segments available to reconnect the parts, we have to process the now divided component. We call the component that s points to the *leaf* component, and the component that s points from the *stem* component.

Lemma 5.10. *When a segment s becomes filled, the sum of the surpluses of the components that are reachable from s's leaf component is* 0 *(the sum includes the leaf component).*

Proof. Note that until s is filled the spending sent along s equals the negative surplus; this remains true at the moment s is filled. □

Lemma 5.11. *At the end of* FindPrice*, either there is an abundant component H consisting of a single buyer with $s(H, \mathbf{p}') = \Delta_{\max}$ or there exists an abundant component H with $s(H, \mathbf{p}') \leq -\Delta_{\max}/4n$.*

Proof. As specified in the stopping conditions in *FindNextDelta*, there exists some component D such that $s(D, \mathbf{p}') = \Delta_{\max}$. If D consists of a single buyer, we are done. Otherwise, there is a component C such that $s(C, \mathbf{p}^C) = \Delta_{\max}$. As $\Delta_{\max} > 0$, by the stopping conditions of *FindNextDelta*, C has a reachable component with $s(D, \mathbf{p}^C) \leq -\Delta_{\max}/4n$. Note that $\mathbf{p}'_D \geq \mathbf{p}^C$, and consequently $s(D, \mathbf{p}'_D) \leq -\Delta_{\max}/4n$. The only issue is that if during the execution of *FindPrice* some segment s inside D is filled and this results in D being split, then we need to argue that the surplus of one of the resulting components is still small enough.

To see this, note that the sum of surpluses of the components that s pointed to is 0. Further, each tree of components that D pointed to has non-positive surplus. Thus, s's leaf component D_l, has non-negative surplus, which implies that s's stem component D_s, has surplus $s(D_s, \mathbf{p}'_{D_s}) \leq -\Delta_{\max}/4n$. □

Lemma 5.12. *A the end of* FindPrice*, either there is an abundant component H consisting of a single buyer i with $\mathbf{e}_i^w \geq 32mn\Delta'$ or there exists an abundant component H with $s(H, \mathbf{p}') \leq -8m\Delta'$.*

Proof. Recall that $\Delta' = \mathrm{rd}(\Delta_{\max}/32mn)$. The lemma follows from Lemma 5.11. □

Lemma 5.13. *Before the end of the $\bar{\Delta}$-scaling phase, there will be a new abundant segment.*

Proof. Recall that $\bar{\Delta} = \Delta'/4mn$. By Lemma 5.12, at the end of *FindPrice*, there is an abundant component C that either consists of a single buyer i that has surplus $s(C) \geq 32mn\Delta' = 128m^2n^2\bar{\Delta}$, or C has surplus $s(C) \leq -8m\Delta'$.

Case 1: C consists a single buyer i with $s(C) \geq 128m^2n^2\bar{\Delta}$.

At the end of the $\bar{\Delta}$-scaling phase, as the spending is $\bar{\Delta}$-optimal, i's unspent money is at most $\bar{\Delta}$. Since i is incident on at most n edges, at least $(128m^2n^2 - 1)\bar{\Delta}/n > 64m^2n$ spending will go on one edge e comprising previously non-abundant segments at the end of the $\bar{\Delta}$-scaling phase.

Case 2: There is an abundant component C with surplus $s(C) \leq -8m\Delta'$.

By Lemma 5.3, *PriceDecrease* could increase C's surplus by at most $3m\Delta'$. Therefore, at the end of *PriceDecrease*, $s(C) \leq -8m\Delta' + 3m\Delta' = -5m\Delta' = -20m^2n\bar{\Delta} \leq -16m^2n\bar{\Delta}$. Since all the goods are at least fully sold at the end of the $\bar{\Delta}$-scaling phase, there will be an edge e with new spending at least $16mn\bar{\Delta}$ from $i \in B \setminus C$ to $j \in C \cap G$. This new spending goes to an edge, and more specifically, segments of this edge, that connect two abundant components. Consequently, these segments are not abundant at the start of this process.

We will show that this spending of at least $16mn\bar{\Delta}$ is allocated to at most two of e's segments, which were not previously abundant; hence one of these segments receives at least $8mn\bar{\Delta}$ spending and is therefore abundant now.

By Corollary 5.1, every Δ_k-short segment is adjacent to Δ_k-long segments. By Lemma 5.9, any Δ-long segment is either $\bar{\Delta}$-abundant or it never becomes fully filled. On the edge e, either the first segment receiving this spending is Δ-long, and never becomes filled; or the first segment is Δ-short, followed by a Δ-long segment that never becomes filled. Thus one of these two segments will receive at least $8mn\bar{\Delta}$ spending and becomes abundant. □

Case 3

Lemma 5.14. *When Δ_u is set to $512m^2n^2\Delta_{k+1}$, the previouly abundant segment remains abundant after calling* FindNext.

Proof. It is easy to verify that Lemmas 5.1, 5.3 and 5.4 still hold. As $\Delta_u = 512m^2n^2\Delta_{k+1}$ and $\Delta/\Delta_{k+1} > 512m^2n^2$, $\Delta_u < \Delta$ and thus $\Delta' = \Delta_u/32mn < \Delta/32mn$.

FindNext has three subroutines: *FindPrice*, *PriceDecrease* and *PriceIncrease*. By Lemma 5.4, the spending on each abundant segment changes by at most $5m\Delta - \alpha$ after *FindPrice*. By Lemma 5.3, *PriceDecrease* performs at most $2m$ augmentations. Therefore, after *PriceDecrease*, the spending changes by at most $2m\Delta' \leq 2m\Delta/32mn < \Delta/16n$. Also, the available money before calling *PriceDecrease* is α, and by Lemma 5.3, *PriceDecrease* increases the surplus of the whole network by at most $3m\Delta' \leq 3m\Delta/32mn < \Delta/8n$. Hence, *PriceIncrease* can remove surplus of at most $\Delta/8n + \alpha$, which is the upper bound of the change in spending on each segment. Note that *PriceDecrease* only increases the surplus and *PriceIncrease* only decreases it. In total, at the end of *PriceIncrease*, compared to the spending before calling *FindNext*, the spending on each abundant segment changes by at most $5m\Delta - \alpha + \Delta/16n + \Delta/8n + \alpha \leq 6m\Delta$.

By Definition 3.1, a segment s is abundant if $\sigma(s) \geq 8mn\Delta$. Then, at the end of *FindNext*, as $\Delta > 32mn\Delta'$, the spending on s is at least $8mn\Delta - 6m\Delta \geq 2m\Delta > 8mn\Delta'$. □

Case 4

Lemma 5.15. *A new abundant segment will emerge no later than the $\hat{\Delta}$-scaling phase.*

Proof. Recall that $\hat{\Delta} = \Delta/64m^2n^2$ and $\Delta/512m^2n^2 > \Delta_{k+1}$. Since the network is fertile, there is a component C such that either C is a single buyer and $s(C) \geq \Delta/4n \geq 16m^2n\hat{\Delta}$, or $s(C) \leq -\Delta/4n \leq -16m^2n\hat{\Delta}$. Then, there will be an edge e that receives at least $16mn\hat{\Delta}$ spending by the end of the $\hat{\Delta}$-scaling phase. At the end of the $\hat{\Delta}$-scaling phase, $\hat{\Delta} > 8\Delta_{k+1}$, so no new segments have been introduced by the end of the $\hat{\Delta}$-scaling phase. By Lemma 5.7, any Δ_k-short segment is adjacent to Δ_k-long segments.

A Δ_k-long segment s has capacity at least $\Delta_k/2 \geq \Delta/2 = 32m^2n^2\hat{\Delta}$. By Lemmas 3.2 and 3.3, during the Δ-phase, the change in spending on each segment is at most $[(4m-1)(n-1)+m+1]\Delta \leq (4m-1)n\Delta$. Thus in a sequence of successive rescalings, starting at the $\hat{\Delta}$-phases, the spending on each segment changes by at most $2(4m-1)n\hat{\Delta} < 8mn\hat{\Delta}$. We claim that by the end of the $\hat{\Delta}$-scaling phase, the Δ_k-long segment s is either $\hat{\Delta}$-abundant or will never become fully filled. If s is already abundant, we are done; otherwise, the unspent part is more than $32m^2n^2\hat{\Delta} - 8mn\hat{\Delta} \geq 24m^2n^2\hat{\Delta}$ and the spending on s can increase by at most $8mn\hat{\Delta}$. Then s will not be fully filled while the Δ-network remains fertile (which is part of the case 4 condition).

The rest of the proof of this lemma is similar to the proof of Lemma 5.13. □

6
Complexity of the Strongly Polynomial Algorithm

Lemma 6.1. FindNextDelta *and* FindPrice *can be implemented to run in* $O(m^2 n)$ *time.*

Proof. We first consider one for loop iteration. There are two types of critical events: (i) a new segment becomes MBB; (ii) an MBB segment becomes filled. Each segment becomes MBB or filled only once during each call to *FindNextDelta* and *FindPrice*. Note that by Lemma 5.10, when a Δ_k-long segment is about to be filled, one of the tree of components it points to has a negative surplus of at least $-\Delta/2n$. Therefore, on each edge e we only need to consider at most three segments, the already abundant segment, the immediately following Δ_k-short segment if any, and the immediately following Δ_k-long segment.

Let R denote the reachable set of the component that induces the for loop iteration. As the spending is acyclic, given the prices it can be computed in $O(n)$ time. The price at which a segment becomes MBB or filled can be expressed in terms of a multiplier x.

We deal with type (i) events with a list L_1 to store the critical values at which the currently non-MBB edges become MBB. We also keep a pointer to the minimum value in L_1. The size of L_1 is $O(m)$.

We deal with type (ii) events by storing the critical values at which an MBB segment becomes filled in a list L_2 and keep a pointer to the minimum value in L_2.

When a segment becomes MBB, we delete the edges that are MBB and insert the new candidate MBB edges, and then find the new minimum value in L_1, which takes time $O(m)$; we will also need to recalculate the spending, delete the current L_2, and build a new list L_2 of critical values, which takes time $O(n)$. In total, it takes $O(m)$ time to deal with a new MBB segment.

When an MBB segment becomes filled, we delete the edges that are not incident to the components reachable from the current search root, and find the minimum value in L_1, which takes time $O(m)$; we will also recalculate the spending, delete the current list and build a new list L_2 of critical values, which takes time $O(n)$. In total, it takes $O(m)$ time to deal with a filled

segment. Note that as a new MBB segment may lead to a recalculation of the spending, MBB segments may become filled when a new MBB segment emerges.

Therefore, each for loop iteration can be implemented to run in time $O(m^2)$. As there are at most n components, the for loop iteration is executed at most n times and thus *FindNextDelta* and *FindPrice* takes time $O(m^2 n)$. □

Theorem 6.1. *The Strongly Polynomial algorithm takes time* $O(rm^2 n + rm[n \log n + m] \log n)$.

Proof. We need to bound the cost of the $O(r + |B|) = O(r)$ events that a buyer or a segment is introduced, or a segment becomes abundant, which drives the strongly polynomial algorithm as specified in the correctness argument. We need to consider four cases:

Case 1: $\Delta/\Delta_{k+1} \leq 512m^2n^2$.

The next critical scaling factor Δ_{k+1} is reached after $O(\log n)$ scaling phases. By Lemma 3.7, the $O(\log n)$ scaling phases take time $O(m[n \log n + m] \log n)$.

Case 2: $\Delta/\Delta_{k+1} > 512m^2n^2$ and the network is not fertile at the start of the while iteration, and $\Delta_{\max}$, the output of *FindNextDelta* satisfies $\Delta_{\max}/\Delta_{k+1} > 512m^2n^2$.

FindNextDelta and *FindNext* are run, following which $O(\log n)$ scaling phases are needed to obtain a new abundant segment.

FindNextDelta takes time $O(m^2 n)$, as shown in Lemma 6.1.

FindNext consists of three subroutines: *FindPrice*, *PriceDecrease* and *PriceIncrease*. *FindPrice* takes time $O(m^2 n)$, as shown in Lemma 6.1. By Lemma 5.3, each *PriceDecrease* performs at most $2m$ augmentations. By Lemma 3.7, *PriceDecrease* can be implemented to run in $O(m[n \log n + m])$ time.

At the end of *FindPrice*, each component has a surplus of at most Δ_u, so the total network has a surplus of at most $n\Delta_u$. By Lemma 5.3, the surplus is further increased by at most $3m\Delta'$ by *PriceDecrease*. The total surplus of the network after *PriceDecrease* is at most $n\Delta_u + 3m\Delta'$. As $\Delta_u = O(mn\Delta')$, and each augmentation in *PriceIncrease* decreases the surplus of the network by Δ', the total number of augmentations performed by *PriceIncrease* is $O(mn^2)$. We now present an efficient implementaion of this *PriceIncrease* procedure.

Let $\mu = M/n$, where M denotes the total available money over all the participating buyers. The implementation starts with *PriceDecrease* using the scaling factor $\Delta = \mathrm{rd}(\mu)$. After this,

in turn we repeatedly halve Δ, run *PriceIncrease*, and *Rescale* until $\Delta = \Delta'$. As in the Weakly Scaling algorithm, each scaling phase needs $O(m[n \log n+m])$ time and the total number of scaling phases is $O(\log n)$. Thus, this implementation of the *PriceIncrease* takes time $O(m \log n[n \log n+ m])$.

Therefore, the *FindNext* subroutine takes time $O(m^2n + m \log n[n \log n + m])$. Lastly, the remaining $O(\log n)$ scaling phases take time $O(m \log n[n \log n + m])$. Overall, Case 2 takes time $O(m^2n + m \log n[n \log n + m])$.

Case 3: $\Delta/\Delta_{k+1} > 512m^2n^2$ and the network is not fertile at the start of the while iteration, and $\Delta_{\max}$, the output of *FindNextDelta* satisfies $\Delta_{\max}/\Delta_{k+1} \leq 512m^2n^2$.

FindNextDelta and *FindNext* are run, following which $O(\log n)$ scaling phases are needed to reach the next critical scaling phase. The running time is $O(m^2n + m \log n[n \log n + m])$, as in Case 2.

Case 4: $\Delta/\Delta_{k+1} > 512m^2n^2$ and the network is already fertile at the start of the while iteration.

After $O(\log n)$ scaling phases, a new abundant segment emerges. These phases take time $O(m[n \log n + m] \log n)$.

Between two critical events, we also need to check the relation between Δ and Δ_{k+1}, whether the network is abundant, and whether a new abundant segment emerges, which in total take time $O(m)$.

Therefore, the total time for the Strongly Polynomial algorithm is $O(rm^2n+rm \log n[n \log n+ m])$.

□

References

[1]Kenneth J. Arrow, H. D. Block, and Leonid Hurwicz. "On the Stability of the Competitive Equilibrium, II". In: *Econometrica* 27.1 (1959), pp. 82–109.

[2]Kenneth J. Arrow and Leonid Hurwicz. "Competitive Stability under Weak Gross Substitutability: The "Euclidean Distance" Approach". In: *International Economic Review* 1.1 (1960), pp. 38–49.

[3]Richard Cole and Vasilis Gkatzelis. "Approximating the Nash Social Welfare with Indivisible Items". In: *Proceedings of the 47th Annual ACM on Symposium on Theory of Computing (STOC). ACM*. 2015.

[4]Xiaotie Deng, Christos Papadimitriou, and Shmuel Safra. "On the Complexity of Price Equilibria". In: *J. Comput. Syst. Sci.* (Sept. 2003).

[5]Nikhil R. Devanur and Vijay V. Vazirani. "The spending constraint model for market equilibrium: Algorithmic, existence and uniqueness results". In: *In Proceedings of 36th Annual ACM Symposium on Theory of Computing (STOC). ACM*. 2004.

[6]Nikhil R. Devanur et al. "Market Equilibrium via a Primal–dual Algorithm for a Convex Program". In: *J. ACM* 55.5 (Nov. 2008).

[7]Lisa Fleischer et al. "A Fast and Simple Algorithm for Computing Market Equilibria." In: *WINE*. 2008.

[8]Hukukane Nikaidô and Hirofumi Uzawa. "Stability and Non-Negativity in a Walrasian Tâtonnement Process". In: *International Economic Review* 1.1 (1960), pp. 50–59.

[9]James B. Orlin. "Improved algorithms for computing fisher's market clearing prices: computing fisher's market clearing prices". In: *Proceedings of the 42nd Annual ACM on Symposium on Theory of Computing (STOC). ACM*. 2010.

[10]Vijay V. Vazirani. "Spending Constraint Utilities with Applications to the Adwords Market". In: (2010).

[11]László A. Végh. "Strongly Polynomial Algorithm for a Class of Minimum-cost Flow Problems with Separable Convex Objectives". In: *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing (STOC)*. 2012.

[12]Léon Walras. *Éléments d'économie politique pure, ou théorie de la richesse sociale (Elements of Pure Economics, or the theory of social wealth, transl. W. Jaffé)*. 1874.