Vol.2011-AL-137 No.3
2011/11/18

An approximate algorithm for computing Fisher's market equilibrium under a simple case of piecewise-linear, concave utilities

Nguyen Cam Ly[†1] and Hiroshi Imai[†1]

We give the first weakly polynomial time algorithm for computing an ϵ-approximate equilibrium for a simple case of piecewise-linear utilities case of Fisher's market model. Assume that set B of buyers and a set G of goods are given. Each buyer has an initial integral e_i of money. The integral utility and budget for buyer i of good j are U_{ij} and t_{ij}, respectively. Each buyer i is not allowed to spend more than budget t_{ij} for good j. For finding ϵ-approximate equilibrium, the algorithm runs in $O(((m+n)\log n/\epsilon)(m+n\log n))$, where $n=|B|+|G|$ and m is the number of pairs (i,j) for which $U_{ij}>0$. The algorithm is based on the previous best running time of $O((n\log n/\epsilon)(m+n\log n))$ for linear utilities case without constraining condition for budgets, due to Orlin[5].

1. Introduction

Fisher's market model (see[1]), one of major market models, has been studied for over a century. It is a simple model of an economic market in which buyers with specific amount of money want to buy their favourite goods among a collection of diverse goods. Each buyer has different utility for each good. In general case, utility of each buyer for each good is described by a concave function. For given prices, each buyer finds an optimal bundle of goods to maximize her utility. The problem is to find equilibrium prices so that the market clears, that is after each buyer is assigned her optimal bundle, there is no surplus or deficiency of the goods.

Last century, there were a few isolated results and some of them were excellent, e.g. Eisenberg and Gale[2], Scarf[3]. The first polynomial time algorithm for the model was developed by Devanur, Papadimitriou, Saberi and Vazirani[4]. 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. After that, Orlin[5] provided a weakly polynomial time algorithm and the first strongly polynomial time algorithm which improved upon the Devanur's one. The weakly polynomial time algorithm and the strongly polinomial time algorithm run in $O(n^4\log U_{max}+n^3\log e_{max})$ time and $O(n^4\log n)$ time, respectively. Moreover, the algorithm can be used to provide and ϵ-approximate solution in $O((n\log n/\epsilon)(m+n\log n))$, where ϵ is a positive number close to 0, $n=|B|+|G|$ and m is the number of pairs (i,j) for which $U_{ij}>0$.

Although those algorithms are good, they only solved the simplest case of Fisher's market model, that is utility of each buyer for each good is described by a linear function. Since the simple case is very far apart from the real market, the algorithms can hardly be applied to find equilibrium prices in the real market. To get closer to the real market, researchers are challenging with more complicated case of Fisher's market model, e.g. utility of each buyer for each good is described by a piecewise-linear, concave function. However, although the problem has been researched for many years and some good results were obtained, e.g. Vazirani and Yannakakis[6], there is still no algorithm for this open problem.

Here we provide a weakly polynomial time for computing Fisher's market equilibrium under a simple case of piecewise-linear, concave utilities. That is, for each good a buyer is given a budget. If the buyer spends less than or equal to given budget for one good, her utility is described by a linear function. Otherwise, she cannot increase her utility. The algorithm provides an ϵ-approximate solution in $O(((m+n)\log n/\epsilon)(m+n\log n))$ time.

This is the first algorithm for computing an equilibrium for the non-linear utilities case of Fisher's market model. Moreover, it allows us to get closer to solving the general piecewise-linear, concave utilities case.

1.1 Description of the model under linear utilities

Before describing our model, we first recall the description of the model under linear utilities, which is the most simple case of Fisher's market model. This model has been well-studied, and the best algorithm for computing its equilibrium prices was proposed by Orlin[5].

†1 Department of Computer Science, The University of Tokyo

© 2011 Information Processing Society of Japan

The market consists of a set B of *buyers* and a set G of divisible *goods*. For each buyer i, the integral amount e_i of money and for each good j the amount of this good are given. Since every good is divisible, for each good j, its amount can be assumed w.l.o.g. as unit. Moreover, the utility functions of the buyers are given. In the linear Fisher's market model, these functions are linear. Let U_{ij} where U_{ij} is integral, denote the utility derived by buyer i on obtaining a unit amount of good j. Therefore if the buyer i spends x_{ij} amount of money on good j whose price is p_j, then the utility she derives on good j is

$$u_{ij} = \frac{U_{ij}x_{ij}}{p_j}.$$

The total utility she derives is

$$\sum_{j\in G} u_{ij}.$$

$p = (p_1, p_2, ...)$ are said to be *equilibrium prices* if, after each buyer is assigned an optimal bundle of goods, i.e. bundle that maximize her utility, there is no surplus or deficiency of any good.

1.2 Description of the model under a simple piecewise-linear, concave utilities

In this paper, we propose an approximate algorithm for the following Fisher's market model with budgets. Similar to the model under linear utilities, we are given a set B of *buyers*, a set G of goods, the initial integral amount e_i of money, utility coefficient U_{ij}. Moreover, in this model, we are given integral budget amount t_{ij} of buyer i on good j. Let x is an allocation of money to goods, where x_{ij} is the amount of money that buyer i uses to buy good j. Our assumption is that, a buyer i cannot increase her utility by spending more than given budget t_{ij} on good j. In other words, if $x_{ij} \geq t_{ij}$, the utility she derives is exactly the same as the utility when she spends t_{ij} for good j. Let p is a vector of prices, in which p_j is the price of good j. Thus the utility buyer i derives is on good j is

$$u_{ij} = \begin{cases} \frac{U_{ij}x_{ij}}{p_j} & \text{if } x_{ij} < t_{ij} \\ \frac{U_{ij}t_{ij}}{p_j} & \text{if } x_{ij} \geq t_{ij}. \end{cases}$$

The total utility she derives is

$$\sum_{j\in G} u_{ij}.$$

We now define some technical terms:

- 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 $b_j(p,x) = -p_j + \sum_{i\in B} x_{ij}$
- The *bang-per-buck* of (i,j) is the ratio U_{ij}/p_j

We suppose that for each buyer i, there is a good j such that $U_{ij} > 0$, otherwise, we eliminate the buyer from the problem. Similarly, we suppose that for each good j there is a buyer i such that $U_{ij} > 0$. In addition, for each buyer i, we assume that $\sum_{j\in G, U_{ij}>0} t_{ij} \geq e_i$, otherwise buyer i will not spend all her money.

For buyer i, we sort all goods by decreasing *bang-per-buck*, and partition by equality into classes: $Q_1, Q_2, ...$ At prices p, goods in Q_l make i equally happy, and those in Q_l make i strictly happier than those in Q_{l+1}. Therefore, buyer i first buys goods in class Q_1. If she still has some money left after the amount of money spending for those goods reach her budgets, she buys goods in class Q_2, and so on.

Let l be the minimum value such that $\forall j \in Q_{l+1} \cup Q_{l+2} \cup ...$, j is i's undesirable good. A pair (i,j) is called *flexible edge* if j belong to class Q_l.

A pair (i,j) is called *forced edge* if good j belongs to one class in $Q_1, ..., Q_{l-1}$.

Let $D(p)$ and $F(p)$ denote the set of forced edges and the set of flexible edges with respect to p, respectively.

A pair (p,x) is an optimal solution if the following constraints are all satisfied:

(1) *Cash constraints*: For each $i \in B, c_i(x) = 0$.
(2) *Allocation of goods constraints*: For each $j \in G, b_j(p,x) = 0$.
(3) *Bang-per-buck constraints*: For each $i \in B$ and $j \in G$, if $x_{ij} > 0$, then $(i,j) \in D(p)$ or $(i,j) \in F(p)$.
(4) *Budget constraints*: For each $i \in B$ and $j \in G$, if $(i,j) \in D(p)$, then $x_{ij} = t_{ij}$. And $x_{ij} \leq t_{ij}$ for all $i \in B$ and $j \in G$.
(5) *Non-negativity constraints*: $x_{ij} \geq 0, p_j \geq 0$ for all $i \in B$ and $j \in G$.

2. The algorithm

2.1 Overview of the algorithm

The algorithm is based on the Δ-scaling algorithm proposed by Orlin[5] for computing equilibrium prices of Fisher's market model under linear utilities. Similar to Orlin's algorithm, our algorithm first decides an initial vector p^0 of

© 2011 Information Processing Society of Japan

prices. Then the algorithm modifies prices of goods and their allocation until an approximation solution is found. Different to Orlin's algorithm which only increases the prices of goods, our algorithm sometimes decreases those prices. Our algorithm also uses Δ as the *scaling parameter*. Here we modify the definition of Δ-*feasible* and Δ-*optimal* which were proposed by Orlin, as follows:

A solution (p, x) is said to be Δ-*feasible* if it satisfies the following conditions:

(1) $\forall j \in G$, $0 \le b_j(p, x) \le \Delta$;

(2) $\forall i \in B$ and $\forall j \in G$, if $x_{ij} > 0$, then $(i, j) \in D(p)$ or $(i, j) \in F(p)$, and x_{ij} is a multiple of Δ;

(3) $\forall i \in B$ and $\forall j \in G$, if $(i, j) \in D$, then $x_{ij} \ge t_{ij}$;

(4) $\forall i \in B$ and $\forall j \in G$, $0 \le x_{ij} \le t_{ij} + \Delta$ and $p_j \ge 0$.

A solution (p, x) is said to be Δ-*optimal* if it is Δ-feasible and satisfies the following condition:

(5) $\forall i \in B$, $0 \le c_i(x) < \Delta$.

During the algorithm, the Δ-feasibility conditions are preserved. Thus the following conditions are preserved. Buyers may spend more or less than their initial money. At any time, more than 100% of a good may be sold. Allocations are required to be multiples of Δ.

The algorithm starts with $\Delta = e_{max}$ and then runs a sequence of scaling phases. It changes prices and allocation so that a Δ-feasible solution changes to a Δ-optimal solution. Next, it transforms the Δ-optimal solution into a $\Delta/2$-feasible solution. Then Δ is replaced by $\Delta/2$, and the algorithm goes to the next scaling phases.

2.2 The initial solution

We set the initial solution as follows. Let $\Delta^0 = e_{max}$; $\forall i \in B$, let $U_{iG} = \sum_{j \in G} U_{ij}$; $\forall i \in B$ and $\forall j \in G$, let $\rho_{ij} = \frac{U_{ij} e_i}{n U_{iG}}$; $\forall j \in G$, let $p^0_j = max\{\rho_{ij} : i \in B\}$; $\forall i \in B$ and $\forall j \in G$. Let $F(p^0)$ be a set consists of edges (i, j) s.t. $\frac{U_{ij}}{p^0_j} = max\{\frac{U_{ij}}{p^0_j} : j \in G\}$.

Clearly, by setting these initial prices, every good has potential buyer(s). For each good j, we choose a buyer i such that $(i, j) \in F(p^0)$ and set $x^0_{ij} = \Delta$, and we change the type of arc (i, j) into backward arc or double directed arc acoordingly. $\forall k \in B, k \ne i$ let $x^0_{kj} = 0$.

We note that the initial solution (p^0, x^0) is Δ-feasible.

2.3 The residual network

Assume that (p, x) is a Δ-feasible solution. We define the residual network $N(p, x)$ as follows: the node set is $B \cup G$. We define four types of arcs as follows:

- *Forward arc* (i, j) satisfies $x_{ij} = 0$ and $(i, j) \in F(p)$.
- *Backward arc* (i, j) satisfies $x_{ij} \ge t_{ij}$ and $(i, j) \in F(p)$.
- *Double directed arc* (i, j) satisfies $0 < x_{ij} < t_{ij}$ and $(i, j) \in F(p)$.
- *Dash arc* (i, j) satisfies $x_{ij} \ge t_{ij}$ and $(i, j) \in D(p)$.

During the Δ-scaling phase, prices and allocation change, so the type of arcs might also be changed. A dash arc is not considered as a forward arc nor backward arc. A double directed arc is considered as both forward arc or backward arc.

2.4 Modify a Δ-feasible solution to Δ-optimal solution

We next describe how the algorithm transforms a Δ-feasible solution (p, x) into a Δ-optimal solution. First, we will show how to modify the prices. Then, we will demonstrate method to change allocation. Finally, we will describe algorithm of procedure **PriceAndAugment** whose input and output are a Δ-feasible solution and a Δ-optimal solution, respectively.

2.4.1 Price changes

Similar to Orlin's algorithm, our algorithm modifies the prices proportionally. However, here we consider two cases: the first is there exist node $r \in B$ with $c_r(x) \ge \Delta$, the second is there exist node $r \in B$ with $c_r(x) < 0$.

In the first case, let *ForwardActiveSet*(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. k is said to be *forward active* with respect to p, x, r if $k \in$ ForwardActiveSet(p, x, r).

In the second case, let *BackwardActiveSet* (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 k to r. k is said to be *backward active* with respect to p, x, r if $k \in$ BackwardActiveSet(p, x, r).

The algorithm replaces the price p_j of each forward active good j by $q \times p_j$ for some $q > 1$. Let $f(q) = PriceIncrease(p, x, r, q)$, where:

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

It replaces the price p_j of each backward active good j by $q \times p_j$ for some q

© 2011 Information Processing Society of Japan

$(0 < q < 1)$. Let $f(q) = PriceDown(p, x, r, q)$, where:

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

To update new prices, we also consider two cases: $c_r(x) \geq \Delta$ and $c_r(x) < 0$. **ForwardUpdatePrice**(p, x, r) is the vector $p\prime$ of prices obtained by setting $p\prime = Price(p, x, r, q\prime)$, where $q\prime$ is the maximum value of q such that $(p\prime, x)$ is Δ-feasible. At least one of three following conditions will be satisfied by $p\prime$:

(1) There is an edge $(i, j) \in F(p\prime) \setminus F(p)$, at which point node j becomes forward active.

(2) There is a dash arc (i, j) becomes a backward arc.

(3) There is a forward active node j with $b_j(p\prime, x) \leq 0$.

In the first two cases, **PriceAndAugment** will continue to update the prices. In the last case, it will carry out an **augmentation** from node r to node j.

BackwardUpdatePrice(p, x, r) is the vector $p\prime$ of prices obtained by setting $p\prime = Price(p, x, r, q\prime)$, where $q\prime$ is the minimum positive value of q such that $(p\prime, x)$ is Δ-feasible. At least one of three following conditions will be satisfied by $p\prime$:

(1) There is an edge $(i, j) \in F(p\prime) \setminus F(p)$, at which point node j becomes forward active.

(2) There is a dash arc (i, j) becomes a backward arc.

(3) There is a forward active node j with $b_j(p\prime, x) \geq \Delta$.

In the first two cases, **PriceAndAugment** will continue to update the prices. In the last case, it will carry out an **augmentation** from node j to node r.

Lemma 1. *If (p, x) is Δ-feasible and $p\prime$ is the vector of prices obtained by ForwardUpdatePrice(p, x, r) or BackwardUpdatePrice(p, x, r), then $(p\prime, x)$ is Δ-feasible.*

2.4.2 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 or from a node G to a node in B is called a *augmenting path*. Similar to Orlin's algorithm, a Δ-*augmentation* along the path P consists of replacing x by a vector $x\prime$, where:

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

Two following lemmas are easy to verify:

Lemma 2. *Suppose that (p, x) is a Δ-feasible, $c_r \geq \Delta$ and that P is an augmenting path from node r to a node $k \in G$ for which $b_j(p, x) \leq 0$. If $x\prime$ is obtained by a Δ-augmentation along path P, then $(p, x\prime)$ is Δ-feasible. In addition, $c_r(x\prime) = c_r - \Delta$, and $c_i(x\prime) = c_i(x)$ for $i \neq r$.*

Lemma 3. *Suppose that (p, x) is a Δ-feasible, $c_r < 0$ and that P is an augmenting path from a node $k \in G$ where $b_j(p, x) \geq \Delta$, to node r. If $x\prime$ is obtained by a Δ-augmentation along path P, then $(p, x\prime)$ is Δ-feasible. In addition, $c_r(x\prime) = c_r + \Delta$, and $c_i(x\prime) = c_i(x)$ for $i \neq r$.*

2.4.3 Algorithm1. Procedure PriceAndAugment

Input: A Δ-feasible solution (p, x).

Output: A Δ-optimal solution (p, x).

for all r such that $c_r(x) \geq \Delta$ **do**
  if there is no forward arc or double directed arc incident to r **then**
    find all goods $k \in G$ and $(r, k) \notin D(p) \cup F(p)$ such that $U_{rk}/p_k = \max\{U_{rj} : j \in G \text{ and } (r, j) \notin D(p) \cup F(p)\}$. Let (r, k) be in $F(p)$ and change all backward arcs incident to r into dash arcs.
  end if
  compute ForwardActiveSet(p, x, r);
  repeat
    replace p by **ForwardUpdatePrice**;
    recompute $N(p, x)$, and ForwardActiveSet(p, x, r) ;
  until there is an active node j with $b_j(p, x) \leq 0$
  let P be a path in $N(p, x)$ from r to j ;
  replace x by carrying out a Δ-augmentation along P ;
  recompute $c(x)$ and $b(p, x)$;
end for
for all r such that $c_r(x) \geq \Delta$ **do**
  compute BackwardActiveSet(p, x, r);

© 2011 Information Processing Society of Japan

repeat
replace p by **BackwardUpdatePrice** ;
recompute $N(p,x)$, and BackwardActiveSet(p,x,r) ;
until there is an active node j with $b_j(p,x) \geq \Delta$
let P be a path in $N(p,x)$ from j to r ;
replace x by carrying out a Δ-augmentation along P ;
recompute $c(x)$ and $b(p,x)$;
end for

2.5 Modify a Δ-optimal solution to a $\Delta/2$-feasible solution

We now describe how to modify a Δ-optimal solution (p,x) to a $\Delta/2$-feasible solution. In Orlin's algorithm, it is easy to modify a Δ-optimal solution (p,x) to a $\Delta/2$-feasible solution by decrease some x_{ij} by $\Delta/2$. In our algorithm, because of the condition of budgets and the existence of dash arcs and backward arcs x_{ij} cannot be changed freely. This problem is solved by a procedure **AllocationAndPrice**. The procedure first decrease some x_{ij} by $\Delta/2$ where $x_{ij} > t_{ij}+\Delta/2$. Thus, some $b_s(x)$ may become smaller than 0. It then modifies b_s which $b_s > \Delta/2$ or $b_s < 0$ by changing the allocation or price of good s.

In the case of $b_s > \Delta/2$, if there is a backward arc or a double directed arc x_{is} incident to s, the procedure substitutes x_{is} with $x_{is} - \Delta/2$. Otherwise, since b_s is positive, there is at least one dash arc incident to s. Then the procedure calls a procedure **SinglePriceIncrease**(p,x,s) to modify price of s. The procedure SinglePriceIncrease(p,x,s) substitutes $p_s(x)$ with $p\prime_s$, where $p\prime_s$ is the value at which point one of two following conditions will be satisfied by $(p\prime,x) = ((..,p_{s-1},p\prime_s,p_{s+1},..),x)$:
(1) There is a dash arc (i,s) becomes a backward arc.
(2) $b_s(p\prime,x) \leq \Delta/2$.

In the former case, the procedure decreases x_{is} by $\Delta/2$.

Similarly, in the case of $b_s < 0$, if there is a forward arc or a double directed arc x_{is} incident to s, AllocationAndPrice substitutes x_{is} with $x_{is}+\Delta/2$. Otherwise, it calls a procedure **SinglePriceDown**(p,x,s) to modify price of s. The procedure SinglePriceDown(p,x,s) substitutes $p_s(x)$ with $p\prime_s$, where $p\prime_s$ is the value at which point one of two following conditions will be satisfied by $(p\prime,x) = ((..,p_{s-1},p\prime_s,p_{s+1},..),x)$:
(1) There is an edge (i,s) becomes a forward arc.
(2) $b_s(p\prime,x) \geq 0$.

In the former case, the procedure increases x_{is} by $\Delta/2$.

The following lemma is straightforward.

Lemma 4. *Suppose that (p,x) is Δ-feasible and $p\prime$ is the vector of prices obtained by SinglePriceIncrease(p,x,s) or SinglePriceDown(p,x,s). Then $(p\prime,x)$ is Δ-feasible.*

We now present procedure AllocationAndPrice.

Algorithm2. AllocationAndPrice(p,x)
Input: A Δ-optimal solution (p,x)
Output: A $\Delta/2$-feasible solution (p,x)
$\Delta := \Delta/2$
for all x_{ij} such that $x_{ij} > t_{ij}+\Delta$ **do**
$x_{ij} := x_{ij} - \Delta$;
end for
recompute $c(x)$ and $b(p,x)$;
for all s such that $b_s(x) > \Delta$ **do**
if there is a node i with (i,s) is backward arc or double directed arc **then**
$x_{is} = x_{ij} - \Delta$;
else
replace (p,x) by SinglePriceIncrease(p,x,s);
if $b_s(p,x) > \Delta$ **then**
find a backward arc (i,s) incident to s, $x_{is} := x_{is} - \Delta$
end if
end if
recompute $c(x)$ and $b(p,x)$;
end for
for all s such that $b_s(x) < 0$ **do**
if there is a node i with (i,s) is forward arc or double directed arc **then**
$x_{is} = x_{ij} + \Delta$;
else

© 2011 Information Processing Society of Japan

replace (p, x) by SinglePriceDown(p, x, s);
if $b_s(p, x) < 0$ **then**
find a forward arc (i, s) incident to s, $x_{is} := x_{is} + \Delta$
end if
end if
recompute $c(x)$ and $b(p, x)$;
end for

2.6 Algorithm3. Scaling Algorithm

Input: ϵ, money e, utilities U, budgets t.
Output: A ϵ-approximate solution (p, x).
$\Delta := \Delta^0; p := p^0; x := 0$ (as per Section 2.2)
for $k = 1$ to PhaseNumber **do**
replace (p, x) by PriceAndAugment(p, x);
replace (p, x) by AllocationAndPrice(p, x);
end for

We now analyze the number of scaling phases *PhaseNumber*. We first give the definition of ϵ-approximate solution. A solution (p, x) is consider to be ϵ-approximate if it is Δ-optimal for $\Delta < \epsilon \Sigma_{j \in G} p_j$, where $\epsilon > 0$.

We have the following theorem. Since the proof is similar to proof of **Theorem** 3.2 in[5], we state the theorem without proof.

Theorem 5. *ScalingAlgorithm determines an ϵ-approximate solution after $O(\log(n/\epsilon))$ scaling phases.*

Therefore, PhaseNumber $= O(\log(n/\epsilon))$.

2.7 Running time and proof

Theorem 6. *During the Δ-scaling phase, ScalingAlgorithm transform a Δ-optimal solution into a $\Delta/2$-optimal solution with 1 call of PriceAndAugment and 1 call of AllocationAndPrice. Procedures PriceAndAugment and AllocationAndPrice can be implemented to run in $O((m + n)(m + \log n))$ time and $O(m + n)n$ time, respectively. For finding a ϵ-approximate solution, the number of scaling phases is $O(\log(n/\epsilon))$. Therefore, the algorithm runs in $O((m + n)(m + \log n) \log(n/\epsilon))$ times.*

The proof is divided into three parts. First, we will prove that the number that PriceAndAugment calls for each $c_r(x) \geq \Delta$ or $c_r(x) < 0$ is $O(m + n)$, and the number that AllocationAndPrice calls for each $b_s(p, x) > \Delta$ or $b_s(p, x) < 0$ is also $O(m + n)$. Then, we will demonstrate how to increase or decrease each b_s in $O(n)$ time. Finally, we will show how to increase or decrease each c_r by Δ in $(O(m + n \log n))$ time.

Proof of number of iterations in PriceAndAugment and AllocationAndPrice. The bound on the number of iterations in AllocationAndPrice relies on the number $\Theta(p, x, \Delta)$ where $\Theta(p, x, \Delta) = \Sigma_{j \in G, b_j(p,x) > \Delta}(\lceil b_j(p, x)/\Delta \rceil - 1) - \Sigma_{j \in G, b_j(p,x) < 0}(\lfloor b_j(p, x)/\Delta \rfloor)$. After one iteration, if $b_j(p, x) > \Delta$ then it will be decreased by Δ, if $b_j(p, x) < 0$ then it will be increased by Δ or until $0 \leq b_j(p, x) \leq \Delta$. Therefore, Θ will be decreased at least by one after each iteration. The procedure AllocationAndPrice will halt when Θ becomes 0. We now prove that at the beginning of AllocationAndPrice, $\Theta < (m+n)$. At the beginning of AllocationAndPrice, since (p, x) is 2Δ-optimal, $\forall j \in G, b_j(p, x) \leq 2\Delta$, therefore $\Sigma_{j \in G, b_j(p,x) > \Delta}(\lceil b_j(p, x)/\Delta \rceil - 1) \leq |G|$. Moreover since $\forall i \in B$ and $\forall j \in G, x_{ij} \leq 2\Delta$, the number of x_{ij} will be decreased by Δ is at most m. Thus, after decreasing some x_{ij}, $-\Sigma_{j \in G, b_j(p,x) < 0}(\lfloor b_j(p, x)/\Delta \rfloor) \leq m$. Therefore $\Theta(p, x, \Delta) \leq (m + |G|) < (m + n)$.

The bound on the number of iterations in PriceAndAugment relies on the number $\theta(x, \Delta)$ where $\theta(x, \Delta) = \Sigma_{i \in B} \mid \lfloor c_i(x)/\Delta \rfloor \mid$. After one iteration, if $c_i(x) \geq \Delta$ then it will be decreased by Δ, if $c_i(x) < 0$ then it will be increased by Δ. Therefore, θ will be decreased by one after each iteration. The procedure PriceAndAugment will halt when θ becomes 0. We now prove that at the beginning of PriceAndAugment, $\theta \leq (2m + n)$. To prove that, we take a look at the process transforming a 2Δ-optimal solution into a Δ-feasible solution. When (p, x) is 2Δ-optimal solution, $\forall i \in B, 0 \leq c_i(x) < 2\Delta$, thus at this point $\theta(x, \Delta) \leq |B|$. During AllocationAndPrice, after at most m numbers of x_{ij} is decreased by Δ, $\Sigma_{i \in B} c_i(x, p)$ will be increase at most $m\Delta$. Moreover during the procedure, since there are at most $m + |G|$ number of $b_j(p, x)$ is decreased or increased by Δ, $\sum_{i \in B} \mid \lfloor c_i(x)/\Delta \rfloor \mid$ is increased by at most $m + |G|$. In the result, at the beginning of procedure AllocationAndPrice, $\theta(x, \Delta)$ is at most $2m + n$. □

Before going to the last two proofs, we define the ratio $\alpha_i(p) = U_{ij}/p_j$ where

© 2011 Information Processing Society of Japan

$j \in G$ and $(i, j) \in F(p)$.

Proof of time bound in one iteration in AllocationAndPrice. We first take a look at procedure that decreases $b_s(p, x)$ by Δ, where $b_s(p, x) > \Delta$. If there is a backward arc or double directed arc incident to s, then the procedure finishes in $O(1)$ time. Otherwise, p_s will be set as $\min\{\Sigma_{i \in B} x_{ij} - \Delta, U_{is}/\alpha_i(p) : (i, s) \in D(p)\}$. Price $p_s = \Sigma_{i \in B} x_{ij} - \Delta$ is the price at which b_s become Δ. Price $p_s = \min\{U_{is}/\alpha_i(p) : (i, s) \in D(p)\}$ is the price at which a dash arc becomes backward arc. Minimum of these values can be calculated in $O(n)$ time.

Similarly, in the procedure that increases $b_s(p, x)$ where $b_s(p, x) < 0$, if there is a forward arc or double directed arc incident to s, then the procedure finishes in $O(1)$ time. Otherwise, p_s will be set as $\max\{\Sigma_{i \in B} x_{ij}, U_{is}/\alpha_i(p) : (i, s) \notin (D(p) \cup F(p))\}$. Price $p_s = \Sigma_{i \in B} x_{ij}$ is the price at which b_s become 0. Price $p_s = \max\{U_{is}/\alpha_i(p) : (i, x) \notin (D(p) \cup F(p))\}$ is the price at which a dash arc becomes backward arc. Maximum of these values can be calculated in $O(n)$ time. Therefore, time bound in one iteration in AllocationAndPrice is $O(n)$.

□

Proof of time bound in one iteration in PriceAndAugment. We now demonstrate that one iteration in PriceAndAugment can be implemented in $O(m + n \log n)$ 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 **ForwardUpdatePrice** and **BackwardUpdatePrice**.

Our implementation relies on the implementation mentioned in the proof of **Theorem** 3.1 in[5]. Here we also store price implicitly and use a Fibonacci heap to store some data. The main difference between two implementations is that in ours we have to consider when a dash arc becomes backward arc.

Let r be the root node of one iteration in **PriceAndAugment** where $c_r(p) \geq \Delta$, and choose a node v so that $(r, v) \in F(p)$. Here p is the vector of prices at the beginning of the iteration. Let $p\prime$ be the vector of prices at some point during the iteration. We store the vector p, the price $p\prime_v$, and the vector $\alpha(p)$. Moreover, for each forward active node k ($k \in B \cup G$), we store γ_k, which is the price of node v when node k become forward active. If node $j \in G$ is not active with respect to $p\prime$, then $p\prime_j = p_j$. Otherwise, $p\prime_j = p_j \times p\prime_v/\gamma_j$.

We divide our proof into two cases: $c_r(p) \geq \Delta$ and $c_r(p) < 0$.

We begin with the first case. An edge (i, j) will join $F(p)$ at the next price update only if i is forward active and j is not forward active. (i, j) will become forward active if the updated price vector $\hat{p}$ satisfies $U_{ij}/\hat{p}_j = \alpha_i(\hat{p}) = \alpha_i(p)\gamma_i/\hat{p}_v$. So, if node $i \in B$ is forward active and node $j \in G$ is not forward active then the price β_{ij} of node when (i, j) belongs to $F(\hat{p})$ is $\beta_{ij} = \gamma_i \alpha_i(p) p_j / U_{ij}$.

And dash arc (i, j) will become backward arc at the next price update only if j is forward active and i is not forward active. (i, j) will become backward arc if the updated price vector $\hat{p}$ satisfies $U_{ij}/\hat{p}_j = \alpha_i(\hat{p})$. So, if node $j \in G$ is forward active and node $i \in B$ is not forward active and (i, j) is dash arc then the price β_{ij} of node when (i, j) becomes backward arc is $\delta_{ij} = U_{ij}\gamma_j/\alpha_i(p)p_j$.

For all forward inactive nodes $j \in G$, let $\beta_j = \min\{\beta_{ij} : i$ is forward active $\}$, for all forward inactive nodes $i \in B$, let $\delta_i = \min\{\delta_{ij} : j$ is forward active $\}$, and we store the β's and δ's in a Fibonacci heap, a data structure developed by Fredmand and Tarjan[7]. The Fibonacci heap supports "FindMin", "Insert", "Delete", and "Decrease Key". The FindMin and Decrease Key operations can each be implemented to run in $O(1)$ time. The Delete and Insert operation each take $O(n)$ time when operating on n elements.

We delete a node from the Fibonacci heap whenever the node becomes forward active. We carry out a Decrease Key whenever β_j is decreased for some j or δ_i is decreased for some i. This event may occur when node i becomes forward active and edge (i, j) is scanned, and $\beta_(ij)$ is determined, or when node j becomes forward active and edge (i, j) is scanned, and $\delta_(ij)$ is determined. Therefore, the time needed to compute when edges join $F(p)$ or dash arcs join $D(p)$ in one iteration in PriceAndAugment is $O(m + n \log n)$ time.

Moreover, we have to observe the price of v when b_j will equal 0. Let $q\prime$ be the minimum value ≥ 1 such that $b_j(q\prime p, x) = 0$. So, $b_j = 0$ when the price of node v is $q\prime\gamma_j$. We store, $\beta_j = q\prime\gamma_j$ into the same Fibonacci Heap as the forward inactive node of G. Therefore the total time to determine all price update in one iteration is $O(m + n \log n)$.

In the second case, $c_r(p) < 0$. An edge (i, j) will join $F(p)$ at the next price update only if j is backward active and i is not backward active. (i, j) will become backward active if the updated price vector $\hat{p}$ satisfies $U_{ij}/\hat{p}_j = \alpha_i(p)$. So, if node

ⓒ 2011 Information Processing Society of Japan

$j \in G$ is backward active and node $i \in B$ is not backward active then the price β_{ij} of node when (i, j) belongs to $F(\hat{p})$ is $\beta_{ij} = U_{ij}\gamma_j/\alpha_i(p)p_j$.

And dash arc (i, j) will become backward arc at the next price update only if i is backward active and j is not backward active. (i, j) will become backward arc if the updated price vector $\hat{p}$ satisfies $U_{ij}/\hat{p}_j = \alpha_i(\hat{p}) = \alpha_i(p)\gamma_i/\hat{p}_v$. So, if node $j \in G$ is backward active and node $i \in B$ is not backward active and (i, j) is dash arc then the price δ_{ij} of node when (i, j) becomes backward arc is $\delta_{ij} = \alpha_i(p)p_j\gamma_j/U_{ij}$.

For all backward inactive nodes $i \in G$, let $\beta_i = \max\{\beta_{ij} : j$ is backward active $\}$, for all backward inactive nodes $j \in G$, let $\delta_j = \max\{\delta_{ij} : i$ is backward active $\}$, and we store the β's and δ's in a Fibonacci heap. We delete a node from the Fibonacci heap whenever the node becomes backward active. We carry out a Decrease Key whenever β_i is increased for some i or δ_j is increased for some j. Similar to the case of $c_r(p) \geq \Delta$, the time needed to compute when edges join $F(p)$ or dash arcs join $D(p)$ in one iteration in PriceAndAugment is $O(m+n \log n)$ time.

Moreover, we have to observe the price of v when b_j will equal Δ. Let $q\prime$ be the maximum value ≤ 1 such that $b_j(q\prime p, x) = \Delta$. So, $b_j = \Delta$ when the price of node v is $q\prime\gamma_j$. We store, $\beta_j = q\prime\gamma_j$ into the same Fibonacci Heap as the forward inactive node of G. Therefore the total time to determine all price update in one iteration is $O(m + n \log n)$.

□

3. Conclusion and Open Problems

We have proposed an approximate algorithm for computing the market equilibrium prices of Fisher's market model under a simple case of piecewise-linear, concave utilities. One immediate question is to find an optimal solution for this problem. This question can be solved if we can find the bound of ϵ where a ϵ-approximate solution becomes an optimal solution.

Another question is that can we extend the algorithm to solve a more general case of Fisher's piecewise-linear, concave utilities. That is, the utility functions is the following. For buyer i and good j, given some budgets $0 = t_{0ij} < t_{1ij} < t_{2ij} < t_{3ij} < ...$ and corresponding utility coefficients $U_{1ij} > U_{2ij} > U_{3ij} > ... \geq 0$. Then the utility buyer i derived on good j when she spends x_{ij} (where $t_{lij} \leq x_{ij} < t_{(l+1)ij}$, l is a non negative integer) amount of money for good j whose price is p_j is:

$$u_{ij} = \frac{U_{(l+1)ij}(x_{ij} - t_{lij}) + \sum_{1 \leq k \leq l} U_{kij}(t_{kij} - t_{(k-1)ij})}{p_j}$$

References

1) 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.
2) E. Eisenberg, and D. Gale. Consensus of subjective probabilities: the Pari-Mutuel method. *Annals of Mathematical Statistic* 30: pp. 165-168, 1959.
3) H. Scarf *The Computation of Economic Equilibria (with collaboration of T. Hansen)*. Cowles Foundation Monograph No. 24., New Haven: Yale University Press, 1973.
4) 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*, pp. 1-18, 2008.
5) James B. Orlin. Improved algorithms for computing Fisher's market clearing prices *Proceedings of STOC'2010*, pp. 291-300.
6) Vijay V. Vazirani, and Mihalis Yannakakis. Market Equilibrium under Separable, Piecewise-Linear, Concave Utilities, with M. Yannakakis, Journal of ACM, vol. 58(3) (2011).
7) M.L. Fredman, and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Jornal of ACM 34*, pp. 596-615, 1987.

© 2011 Information Processing Society of Japan