Market Equilibrium with Transaction Costs

Sourav Chakraborty[1], Nikhil R Devanur[2], and Chinmay Karande[⋆3]

[1] Chennai Mathematical Institute, India. sourav@cmi.ac.in
[2] Microsoft Research, Redmond WA. nikdev@microsoft.com
[3] Google Inc., Mountain View CA. chinmayk@google.com

Abstract. Identical products being sold at different prices in different locations is a common phenomenon. To model such scenarios, we supplement the classical Fisher market model by introducing *transaction costs*. For every buyer i and good j, there is a transaction cost of c_{ij}; if the price of good j is p_j, then the cost to the buyer i *per unit* of j is $p_j + c_{ij}$. The same good can thus be sold at different (effective) prices to different buyers. We provide a combinatorial algorithm that computes ϵ-approximate equilibrium prices and allocations in $O\left(\frac{1}{\epsilon}(n + \log m)mn\log(B/\epsilon)\right)$ operations - where m is the number goods, n is the number of buyers and B is the sum of the budgets of all the buyers.

1 Introduction

Identical products being sold at different prices in different locations is a common phenomenon. Price differences might occur due to different reasons such as

- Shipping costs. Oranges produced in Florida are cheaper in Florida than they are in Alaska, for example.
- Trade restrictions. A seller with access to a wider market might sustain a higher price than one that does not.
- Price discrimination. A good might be priced differently for different people based on their respective ability to pay. For example, conference registration fees are typically lower for students than for professors.

To capture such scenarios, we supplement the classical Fisher model of a market by introducing *transaction costs*. For every buyer i and every good j, there is a transaction cost of c_{ij}; if the price of good j is p_j, then the cost to the buyer i *per unit* of j is $p_j + c_{ij}$. The same good can thus be sold at different effective prices to different buyers. Apart from non-negativity, the transaction costs are not restricted in any way and in particular, do not have to satisfy the triangle inequality.

Fisher's Market Model with Transaction Costs: In Fisher's model, a market $\mathcal{M}$ has n buyers and m divisible goods. Every buyer i has budget B_i. We consider *linear* utility functions, *i.e.*, the utility of a buyer i on obtaining a bundle of goods $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots)$ is $\sum_j u_{ij}x_{ij}$ where u_{ij} are given constants. Each

⋆ Part of the work done during the author's internship at Microsoft Research.

good has an available supply of one unit (which is without loss of generality). In addition to its price, a buyer also pays a transaction cost c_{ij} per unit of good j. The allocation bundle for buyer i is a vector $\boldsymbol{x}_i$ such that x_{ij} denotes the amount of good j allocated to buyer i. A price vector $\boldsymbol{p}$ is an equilibrium of $\mathcal{M}$ if there exists allocations $\boldsymbol{x}_i$ such that

- $\boldsymbol{x}_i$ maximizes the utility of i among all bundles that satisfy the budget constraint, i.e. $\boldsymbol{x}_i \in \arg\max_{\boldsymbol{y}_i} \{\sum_j u_{ij} y_{ij} : \sum_j (p_j + c_{ij}) y_{ij} \leq B_i\}$
- Every good is either fully allocated or is priced at zero, i.e. $\forall j$, either $\sum_i x_{ij} = 1$ or $p_j = 0$.

Characterization of Market Equilibrium: We now characterize the equilibrium prices and allocations in our model. The ratio $u_{ij}/(p_j + c_{ij})$ denotes the amount of utility gained by buyer i through one dollar spent on good j. At given prices, a bundle of goods that maximizes the total utility of a buyer contains only goods that maximize this ratio. Let $\alpha_i = \max_j u_{ij}/(p_j + c_{ij})$ be the bang-per-buck of buyer i at given prices. We will call the set $D_i = \{\, j \mid u_{ij} = \alpha_i (p_j + c_{ij}) \,\}$ the demand set of buyer i. Hence, $x_{ij} > 0 \;\Rightarrow\; j \in D_i$. The conditions characterizing these equilibrium prices and allocations appear in table A below.

An ϵ-approximate market equilibrium is characterized by relaxing the market clearing condition (Equation (3)) and optimal allocation condition (Equation (4)). Refer to equations (7) and (8) in table B.

A: Market Equilibrium		B: ϵ-Approximate Market Equilibrium	
$\forall i \;\; \sum_j (p_j + c_{ij}) x_{ij} = B_i$	(1)	$\sum_j (p_j + c_{ij}) x_{ij} = B_i$	(5)
$\forall j \;\; \sum_i x_{ij} \leq 1$	(2)	$\sum_i x_{ij} \leq 1$	(6)
$\forall j \;\; p_j > 0 \;\Rightarrow\; \sum_i x_{ij} = 1$	(3)	$p_j > \epsilon \;\Rightarrow\; \sum_i x_{ij} \geq 1/(1+\epsilon)$	(7)
$\forall i, j \;\; x_{ij} > 0 \;\Rightarrow\; \dfrac{u_{ij}}{\alpha_i} = p_j + c_{ij}$	(4)	$x_{ij} > 0 \;\Rightarrow\; \dfrac{u_{ij}}{\alpha_i} \geq \dfrac{p_j + c_{ij}}{1+\epsilon}$	(8)

The relaxation of exact equilibrium conditions can be achieved in other ways. For example, [7] use a definition of ϵ-approximate market equilibrium that relaxes the budget constraints. Our algorithm can be easily adapted to this definition by simple modifications to the termination conditions.

Our Result

Our main result is a combinatorial algorithm that computes ϵ-approximate equilibrium prices and allocations in $O\left(\frac{1}{\epsilon}(n + \log m) mn \log(B/\epsilon)\right)$ operations - where m is the number goods, n is the number of buyers and B is the sum of the budgets of all the buyers. This algorithm is a generalization of the auction algorithm of Garg and Kapoor [7] to our model with the transaction costs. This generalization is not straight forward; the presence of transaction costs introduces new challenges. The term 'auction algorithm' is used to describe ascending price algorithms (such as the one in [7]) which maintain a feasible allocation at all times. The algorithm makes progress by revoking a portion of goods currently assigned to a buyer and reallocating it to another buyer offering a higher price. Our method of reallocating goods is similar in spirit to the path auctions used by [9]. The auction algorithms in both [7] and [9] crucially use the properties of monotonic decrease in surplus and acyclicity of the demand graph. However, these properties cease to exist when transaction costs are introduced. The main technical contribution of this paper is in dealing with the absence of these properties (and yet getting almost the same results). A more detailed discussion of the same is presented in the full version of this paper [1].

Related Work

The computation of economic and game theoretic equilibria has been an active area of research over the past decade. Hardness results and algorithmic results [10] have been delineating the boundary between what is efficiently computable and what is not.

Convex programming has been one of the main tools in designing algorithms for market equilibrium. A simple modification of the convex program introduced by [6, 5] captures the equilibria of our problem as its optimal solution. (Refer to [1] for details) This proves *existence and uniqueness* of equilibria. It also implies that the ellipsoid algorithm can be used to get a polynomial time algorithm to compute the equilibrium[4]. The auction algorithm is combinatorial, runs faster and provides a simple alternative that can be implemented efficiently in practice. It is not clear if one can construct an interior point algorithm to solve the convex program. Ye [12] gave one such algorithm for the Eisenberg-Gale convex program.

A strongly polynomial time algorithm for the Fisher linear market was given by Orlin [11]; it does not seem like his ideas can be adapted directly to our setting. Chen *et al* [2] study a model similar to ours, in the context of profit-maximizing envy-free pricing (for a single commodity but at different locations).

Codenotti *et al* [3] studied transaction costs that are a fixed fraction of the price, and hence can be interpreted as taxes. The taxes could be uniform, that is, depend only on the good, or non-uniform, that is, depend on the good and the buyer. In the Fisher's model, our algorithm can also handle per-dollar taxes, with minimum modifications.

Extensions and Open Problems

All of our results can be easily extended to quasi-linear utilities, that is, the buyers have utility for money as well, which is normalized to 1. So the utility of the bundle $\boldsymbol{x}_i$ is $\sum_j (u_{ij} - p_j)x_{ij}$. Extending the results to other common utility functions is an open problem. In particular, Garg, Kapoor and Vazirani [8] extend the auction algorithm to separable weak gross substitute utilities. The potential

[4] Since the equilibrium could be irrational, the ellipsoid algorithm would compute an equilibrium with precision δ in time proportional to $\log(1/\delta)$.

function they use is the total surplus, and we don't know a combinatorial bound on the number of events in their algorithm. As mentioned earlier, this potential function cannot be used in the presence of transaction costs, and therein lies the difficulty in extending our results to this case.

The auction algorithm for the traditional models can be made to be distributed and even asynchronous, with a small increase in the running time. We show that a similar distributed/asynchronous version of the algorithm may not converge in the presence of transaction costs. An interesting open question is if there is some other asynchronous/distributed algorithm that also converges fast. In particular, is there a tattonnement process that converges fast (like in [4])?

Outline: We provide an overview, followed by the details of our algorithm in Section 2. Section 3 contains the analysis of the running time. The proofs of lemmas in Section 2 and 3 have been omitted due to space constraints and can be found in the full version of this paper [1].

2 Algorithm

Theorem 1. *We can find ϵ-approximate equilibrium prices and allocations in* $O\left(\frac{1}{\epsilon}(n+\log m)mn\log(B/\epsilon)\right)$ *operations where* $B = (1+\epsilon)\sum_i B_i$.

Overview Our algorithm maintains a set of prices and allocations and modifies them progressively. To initialize, we set all the prices $p_j = \epsilon$ and all the allocations are empty. The algorithm is organized in rounds. At the end of each round, we raise the price of one good by a multiplicative factor of $1+\epsilon$. Any allocations made before the price raise continue to be charged at the earlier, lower price. Therefore at any point in the algorithm, a good may be allocated to buyers at two different prices, p_j and $p_j/(1+\epsilon)$. During a round, we take a good away from a buyer at the lower price and allocate it to a buyer (possibly the same buyer) at the current, higher price. We find a sequence of such reallocations such that we eventually find a buyer with positive surplus and a good in her demand set such that all of that good is allocated at the current price. When we find such a buyer-good pair, we increase the price of that good and end the round. The algorithm terminates when the budgets of all the buyers are exhausted.

Following invariants are maintained throughout the algorithm:

I1: Buyers have non-negative surplus i.e. no buyer exceeds her budget.

I2: All prices are at least ϵ.

I3: Every good is either priced ϵ or is fully allocated.

I4: Any good j allocated to a buyer i must be approximately most desirable. (As in Equation (8))

I5: A good j is allocated at price either p_j or $p_j/(1+\epsilon)$ where p_j is the current price.

Invariant I3 is a tighter version of equation (7). We maintain I3 and I5 until the end of the algorithm whence we merge the two price tiers. This may lead to some goods being undersold, but we prove that equation (7) still holds. Also note that invariant I4 holds for any allocations, whether at the higher or lower price tier. Unless mentioned otherwise, the statements of all the lemmas that follow are constrained to maintain these invariants.

We now present the details of our algorithm. Each round consists of roughly two parts: 1) We construct a *demand graph* G on the set of buyers and 2) We perform multiple iterations of a reallocation procedure - which we call a *transfer walk*. At the end of each round, we increment the price of some good. The sequence of rounds ends when the surplus of all the buyers reduces to zero. At the end, we readjust the allocations to merge the two price tiers. In what follows, we explain our algorithm in three parts: a) Construction and properties of the demand graph, b) Transfer walks and c) Readjustment of allocations.

Notation: We denote the allocations of good j to buyer i at prices p_j and $p_j/(1+\epsilon)$ as h_{ij} and y_{ij} respectively. We denote by $z_j = 1 - \sum_i (h_{ij} + y_{ij})$ the amount of good j unassigned at any point in the algorithm. Given any prices and allocations, the surplus r_i of buyer i is the part of her budget unspent:

$$r_i \; = \; B_i \; - \; \sum_j (p_j + c_{ij}) h_{ij} \; - \; \sum_j \left(\frac{p_j}{1+\epsilon} + c_{ij} \right) y_{ij}$$

Notice that since the prices remain constant throughout a round except at the end, the demand sets of all the buyers are well defined. In each round we fix a function $\pi(i) = \min\{\, j \mid j \in D_i \,\}$. Intuitively, we will attempt to allocate the good $\pi(i)$ to i in this round, ignoring all the other goods in D_i for the moment. Any choice of a good from D_i suits as $\pi(i)$, but we fix a function for ease of exposition.

Construction and properties of the demand graph: We construct a directed graph G on the set of buyers. An edge exists from buyer i to k if and only if $y_{k\pi(i)} > 0$. A node i in this graph with (1) no out-edges (*i.e.* a sink), (2) $r_i > 0$ and (3) $z_{\pi(i)} = 0$ will be defined to be 'unsatisfiable'.

Lemma 1. *For an unsatisfiable node i, the price of the good $\pi(i)$ can be increased by a multiplicative factor of $1 + \epsilon$.*

But the graph G may not contain an unsatisfiable node to start with. Hence we perform a series of reallocations until we create and/or find such a node.

The reallocation involves the following step: For an edge $i \to k$ in G with $r_i > 0$, we take away the lower price allocation of good $\pi(i)$ for k and allocate it to i at the current price. In short, we perform the operations $y_{k\pi(i)} \leftarrow y_{k\pi(i)} - \delta$ and $h_{i\pi(i)} \leftarrow h_{i\pi(i)} + \delta$ for a suitably chosen value of δ. This process reduces r_i, $y_{k\pi(i)}$ and increases r_k. If $y_{k\pi(i)}$ reduces to zero, we drop the edge (i, k) from the graph. When we make such a reallocation, we say that we *transfer surplus* from i to k. Note that the surplus is not conserved. This is because the price paid by i for the same amount of the good, including the transaction costs, could even be lower than the price paid by k.

Lemma 2. *If the edge from i to k exists in G with $r_i > 0$, then we can transfer surplus from i to k such that either the surplus of i becomes zero or the edge (i, k) drops out of G.*

We can repeatedly apply Lemma 2 to transfer surplus along a path in G.

Corollary 1. *If there exists a path from i to k in G and $r_i > 0$, then we can transfer surplus from i to k such that either the surplus of all the nodes on the path except k becomes zero or an edge in the path drops out of G.*

Finally, G may contain cycles. Consider the edges (i_1, i_2) and (i_2, i_3) in G and let $j_1 = \pi(i_1)$ and $j_2 = \pi(i_2)$. If the transaction costs are all zero, then it can be argued that the last price raise for j_1 must have taken place before the last price raise for j_2. Repeating this argument, one can preclude the existence of cycles in G in absence of transaction costs. This acyclicity of G forms a pivotal argument in the algorithm of Garg and Kapoor [7]. In the full version of this paper [1], we provide a sketch of how a cycle can emerge in G when transaction costs are present. We also show that the algorithm of [7] can slow down indefinitely if G contains cycles. Therefore, we need to be able to transfer surplus around a cycle.

Lemma 3. *If there exists a cycle in G and exactly one node in the cycle has positive surplus, then we can transfer surpluses in such a way that either all the node in the cycle have zero surplus or an edge in the cycle drops out.*

In a round, we use the above lemmas to perform multiple iterations of the transfer walk.

Transfer Walk

Step 1: Find a node i_0 with a positive surplus. If there are no such nodes, then terminate the round and jump to readjustment of allocations.

Step 2: Follow a path going out of i_0 in G in a depth-first-search fashion. We look at the first edge in the adjacency list of the last visited node i on the path. Let (i, k) be this edge. If node k is yet unvisited, we follow that edge to extend the path. If k is already on the path, then we have found a cycle in G. Finally if i has no out-edges, then we have found a sink. Whichever the case, we now transfer surplus along the current path from i_0 to i as in Corollary 1. If an edge along the path drops out, we trigger event 2d. Otherwise, we trigger events 2a-2c depending upon case. The transfer walk must end in a finite number of operations in one the of following events:

Event 2a - The path reaches a sink i with $z_{\pi(i)} = 0$: Let $j = \pi(i)$. By Corollary 1, we must have transferred a positive surplus to i even if r_i was zero at the begining of the walk. Hence i is an unsatisfiable node. Raise $p_j \leftarrow (1+\epsilon)p_j$. Terminate the walk and the round.

Event 2b - The path reaches a sink i with $z_{\pi(i)} > 0$: Let $j = \pi(i)$. By invariant I3, $p_j = \epsilon$. We let $\delta = \min(\ r_i/\epsilon,\ z_j\ )$. We then assign $h_{ij} \leftarrow h_{ij} + \delta$. If $\delta = r_i/\epsilon$ then the surplus of i goes to zero otherwise z_j goes to zero. In either case we end this transfer walk.

Event 2c - The path finds a cycle: Let i be the last node visited on the path and an edge (i, k) in G reaches a node k already visited on the path. By Corollary 1, all the nodes in the cycle except i have zero surplus. Therefore, we apply Lemma 3 until the surplus of i becomes zero or an edge in the cycle drops out. We terminate the current walk.

Event 2d - An edge drops out during path transfer: In this case we terminate the current walk.

If a transfer walk ends in event 2a, we terminate the current round and start the next one. Otherwise if events 2b-2d are triggered, we start a new transfer walk. If the surplus of all buyers is found to be zero in Step 1, we move to the last phase, which is readjustment of allocations.

Readjustment of allocations: At the end of the transfer walks, all the required invariants are satisfied, but the same good may be allocated to the same or different buyers at different prices: p_j and $p_j/(1+\epsilon)$. Therefore in this phase, we merge the two tiers of allocation for every buyer-good pair to create the final allocations. For all i, j such that $y_{ij} > 0$, we assign $x_{ij} \leftarrow h_{ij} + \frac{\frac{p_j}{1+\epsilon}+c_{ij}}{p_j+c_{ij}} y_{ij}$. The final equilibrium prices are the prices at the termination of the algorithm.

Theorem 2. *The algorithm produces ϵ-approximate equilibrium prices and allocations.*

3 Analysis

Lemma 4. *If R is the number of rounds in the algorithm, then the number of transfer walks that end in an edge dropping out of G is at most nR.*

Proof of Theorem 1:

Initialization and readjustment: Both the initialization and final adjustment of allocations can be performed in mn operations.

The number of rounds: The price of exactly one good is raised by multiplicative factor of $1+\epsilon$ in each round except the last round. Starting at ϵ, the maximum value to which a price may be raised is $B = (1+\epsilon)\sum_i B_i$. Therefore, there can be at most $R = 1 + \frac{m}{\epsilon}\log(\frac{B}{\epsilon})$.

Constructing the graph: For each buyer, we maintain all the goods in a balanced tree data structure that sorts the goods first by the bang-per-buck $u_{ij}/(p_j + c_{ij})$ and then by the index j. In this manner, we can compute the function $\pi(i)$ in $O(\log m)$ time. Given $\pi(i)$, every node may have an edge to every other node. Therefore, the graph G can be constructed in $O(n^2 + n\log m)$ operations. After the price increase at the end of the round, the sorted trees can be maintained in time $O(n\log m)$ while the transfer of allocations from higher to lower price tier can be completed in $O(n)$ operations.

Number of transfer walks: All the remaining computation in the algorithm takes place within the transfer walks. Since we follow the first edge going out of each vertex, the depth-first-search requires only $O(n)$ operations. The surplus transfer along a path and a cycle can similarly be performed in $O(n)$ operations. When an edge drops out, updating G involves simply incrementing a pointer. Therefore, overall a transfer walk requires $O(n)$ operations.

We will now bound the number of transfer walks that happen throughout the algorithm, including all the rounds. We will classify them by the event that ends the walk. At most R transfer walks can terminate the round. At most m walks can end with z_j going zero. Lemma 4 bounds the number of walks that end with an edge dropping out of the graph. The only remaining case is that the walk ends when the surplus of the last visited node on the path vanishes. A transfer walk ending in this case leaves one less node in G with a positive surplus. To see this, observe that a transfer walk starts with a node on the same path with positive surplus and by the time it ends in this case, all the nodes on the path have zero surplus by Corollary 1 and Lemma 3.

Let r_+ denote the number of nodes in G with positive surplus. After initialization we have $r_+ = n$. The only event which may increase r_+ is event 2d. If an edge (i, k) drops out during surplus transfer along the path, node k may be left with some positive surplus that was absent at the start of the walk. Therefore r_+ increases by at most one in this event. Combined with Lemma 4, this implies a bound of $n + nR$ on the number of times r_+ reduces.

It is clear from the above analysis that the algorithm performs at most $O(nR)$ transfer walks. Combined with the other computation bounds, this yields an upper bound of $O\left(\frac{1}{\epsilon}(n + \log m)mn \log(B/\epsilon)\right)$ on the running time of the algorithm.

References

1. Sourav Chakraborty, Nikhil R. Devanur, and Chinmay Karande. Market equilibrium with transaction costs. *CoRR*, abs/1001.0393, 2010.
2. Ning Chen, Arpita Ghosh, and Sergei Vassilvitskii. Optimal envy-free pricing with metric substitutability. In *ACM Conference on Electronic Commerce*, pages 60–69, 2008.
3. Bruno Codenotti, Luis Rademacher, and Kasturi R. Varadarajan. Computing equilibrium prices in exchange economies with tax distortions. In *ICALP (1)*, pages 584–595, 2006.
4. Richard Cole and Lisa Fleischer. Fast-converging tatonnement algorithms for one-time and ongoing market problems. In *STOC*, pages 315–324, 2008.
5. Nikhil R Devanur. Fisher markets and convex programs. Unpublished Manuscript, 2008.
6. Jugal Garg. *Algorithms for Market Equilibrium.* PhD thesis, Dept. of Computer Science and Engineering, IIT Bombay, 2008.
7. R. Garg and S. Kapoor. Auction algorithms for market equilibrium. In *Proceedings of 36th STOC*, 2004.
8. R. Garg, S. Kapoor, and V. V. Vazirani. An auction-based market equilbrium algorithm for the separable gross substitutibility case. In *Proceedings, APPROX*, 2004.
9. Rahul Garg and Sanjiv Kapoor. Price roll-backs and path auctions: An approximation scheme for computing the market equilibrium. In *WINE*, pages 225–238, 2006.
10. Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay Vazirani, editors. *Algorithmic Game Theory*, chapter 2, 5 and 6. Cambridge University Press, 2007.
11. James B. Orlin. Improved algorithms for computing fisher's market clearing prices: computing fisher's market clearing prices. In *STOC*, pages 291–300, 2010.
12. Yinyu Ye. A path to the arrow-debreu competitive market equilibrium. *Mathematical Programming*, 111(1-2):315–348, 2008.