CHAPTER 5

Combinatorial Algorithms for Market Equilibria

Vijay V. Vazirani

Abstract

Combinatorial polynomial time algorithms are presented for finding equilibrium prices and allocations for the linear utilities case of the Fisher and Arrow–Debreu models using the primal-dual schema and an auction-based approach, respectively. An intersting feature of the first algorithm is that it finds an optimal solution to a nonlinear convex program, the Eisenberg-Gale program.

Resource allocation markets in Kelly's model are also discussed and a strongly polynomial combinatorial algorithm is presented for one of them.

5.1 Introduction

Thinkers and philosophers have pondered over the notions of markets and money through the ages. The credit for initiating formal mathematical modeling and study of these notions is generally attributed to nineteenth-century economist Leon Walras (1874). The fact that Western economies are capitalistic had a lot to do with the overwhelming importance given to this study within mathematical economics – essentially, our most critical decision-making is relegated to pricing mechanisms. They largely determine the relative prices of goods and services, ensure that the economy is efficient, in that goods and services are made available to entities that produce items that are most in demand, and ensure a stable operation of the economy.

A central tenet in pricing mechanisms is that prices be such that demand equals supply; that is, the economy should operate at equilibrium. It is not surprising therefore that perhaps the most celebrated theorem within general equilibrium theory, the Arrow–Debreu Theorem, establishes precisely the existence of such prices under a very general model of the economy. The First Welfare Theorem, which shows Pareto optimality of allocations obtained at equilibrium prices, provides important social justification for this theory.

Although general equilibrium theory enjoyed the status of crown jewel within mathematical economics, it suffers from a serious shortcoming – other than a few isolated results, some of which were real gems, e.g., Eisenberg and Gale (1959) and Scarf

(1973), it was essentially a nonalgorithmic theory. With the emergence of new markets on the Internet, which already form an important part of today's economy and are projected to grow considerably in the future, and the availability of massive computational power for running these markets in a distributed or centralized manner, the need for developing an algorithmic theory of markets and market equilibria is apparent. Such algorithms can also provide a valuable tool for understanding the repercussions of technological advances, new goods or changes to the tax structure on existing prices, production, and consumption.

A good beginning has been made over the last 5 years within algorithmic game theory, starting with the work of Deng et al. (2002). However, considering the fact that markets were an active area of study for over a century within mathematical economics, it is safe to say that we have only scratched the surface of what should be a rich theory.

Irving Fisher (see Brainard and Scarf, 2000) and Walras (1874) gave two fundamental market models that were studied extensively within mathematical economics. The latter model is also called the exchange model or the Arrow–Debreu model (Arrow and Debreu, 1954). In this chapter we will present combinatorial algorithms for both these models for the case of linear utility functions. A second approach that has emerged for computing equilibria for these models is the efficient solution of convex programs, since equilibrium allocions for both these models can be captured via convex programs; see Chapter 6 for this approach.

Two techniques have been primarily used for obtaining combinatorial algorithms for these models – the primal-dual schema (Devanur et al. 2002) and an auction-based approach (Garg and Kapoor, 2004). We will present algorithms for the Fisher and Arrow–Debreu models, using the first and second techniques, respectively.

An interesting aspect of the first algorithm was the extension of the primal-dual schema from its usual setting of combinatorially solving, either exactly or approximately, linear programs, to exactly solving a nonlinear convex program (see Section 5.5). The latter program, due to Eisenberg and Gale (1959), captures equilibrium allocations for the linear case of Fisher's model. Unlike complementary slackness conditions for linear programs, which involve either primal or dual variables, but not both, KKT conditions for a nonlinear convex program simultaneously involve both types of variables. The repercussions of this are apparent in the way the algorithm is structured.

In a different context, that of modeling and understanding TCP congestion control,[1] Kelly (1997) defined a class of resource allocation markets and gave a convex program that captures equilibrium allocations for his model. Interestingly enough, Kelly's program has the same structure as the Eisenberg–Gale program (see also Chapter 22).

[1] In particular, Kelly's object was to explain the unprecedented success of TCP, and its congestion avoidance protocol due to Jacobson (1988), which played a crucial role in the phenomenal growth of the Internet and the deployment of a myriad of diverse applications on it. Fairness is a key property desired of a congestion avoidance protocol and Jacobson's protocol does seem to ensure fairness. Recent results show that if Jacobson's protocol is run on the end-nodes and the Floyd–Jacobson protocol (Floyd and Jacobson, 1993) is run at buffer queues, in the limit, traffic flows converge to an optimal solution of Kelly's convex program, i.e., they are equilibrium allocations, see Low and Lapsley (1999). Furthermore, Kelly used his convex programming formulation to prove that equilibrium allocations in his model satisfy proportional fairness (see Section 5.13), thereby giving a formal ratification of Jacobson's protocol.

The *flow market* is of special significance within this framework. It consists of a network, with link capacities specified, and source – sink pairs of nodes, each with an initial endowment of money; allocations in this market are flows from each source to the corresponding sink. The problem is to find equilibrium flows and prices of edges (in the context of TCP, the latter can be viewed as drop rates at links).

Kelly's model attracted much theoretical study, partly with a view to designing next-generation protocols. Continuous time algorithms (though not having polynomial running time), for finding equilibrium flows in the flow market, were given by Kelly et al. (1998) (see also Wang et al., 2005, for more recent work along these lines). Soon after the appearance of Devanur et al. (2002), Kelly and Vazirani (2002) observed that Kelly's model esentially generalizes Fisher's linear case and stated, "Continuous time algorithms similar to TCP are known, but insights from discrete algorithms may be provocative."

With a view to answering this question, a systematic study of markets whose equilibria are captured by Eisenberg-Gale-type programs was undertaken by Jain and Vazirani (2006). In Section 5.14 we present, from this paper, a strongly polynomial algorithm for the special case of the flow market when there is one source and multiple sinks.

5.2 Fisher's Linear Case and the Eisenberg–Gale Convex Program

Fisher's linear case[2] is the following. Consider a market consisting of a set B of *buyers* and a set A of divisible *goods*. Assume $|A| = n$ and $|B| = n'$. We are given for each buyer i the amount e_i of money she possesses and for each good j the amount b_j of this good. In addition, we are given the utility functions of the buyers. Our critical assumption is that these functions are linear. Let u_{ij} denote the utility derived by i on obtaining a unit amount of good j. Thus if the buyer i is given x_{ij} units of good j, for $1 \le j \le n$, then the happiness she derives is

$$\sum_{j=1}^{n} u_{ij} x_{ij}.$$

Prices $p_1, \ldots, p_n$ of the goods are said to be *market clearing prices* if, after each buyer is assigned an optimal basket of goods relative to these prices, there is no surplus or deficiency of any of the goods. Our problem is to compute such prices in polynomial time.

First observe that w.l.o.g. we may assume that each b_j is unit – by scaling the u_{ij}'s appropriately. The u_{ij}'s and e_i's are in general rational; by scaling appropriately, they may be assumed to be integral. We will make the mild assumption that each good has a *potential buyer*; i.e., a buyer who derives nonzero utility from this good. Under this assumption, market clearing prices do exist.

It turns out that equilibrium allocations for Fisher's linear case are captured as optimal solutions to a remarkable convex program, the Eisenberg–Gale convex program.

[2] See Section 5.13 for a special case of this market and a simple polynomial time algorithm for it.

Before stating the program, it will be instructive to list considerations that would be useful in deriving such a program.

Clearly, a convex program whose optimal solution is an equilibrium allocation must have as constraints the packing constraints on the x_{ij}'s. Furthermore, its objective function, which attempts to maximize utilities derived, should satisfy the following:

- If the utilities of any buyer are scaled by a constant, the optimal allocation remains unchanged.
- If the money of a buyer b is split among two new buyers whose utility functions are the same as that of b then sum of the optimal allocations of the new buyers should be an optimal allocation for b.

The money weighted geometric mean of buyers' utilities satisfies both these conditions:

$$\max \left(\prod_{i \in A} u_i^{e_i} \right)^{1/\sum_i e_i}.$$

Clearly, the following objective function is equivalent:

$$\max \prod_{i \in A} u_i^{e_i}.$$

Its log is used in the Eisenberg–Gale convex program:

$$\begin{array}{lll} \text{maximize} & \displaystyle\sum_{i=1}^{n'} e_i \log u_i & \\ \text{subject to} & u_i = \displaystyle\sum_{j=1}^{n} u_{ij} x_{ij} & \forall i \in B \\ & \displaystyle\sum_{i=1}^{n'} x_{ij} \le 1 & \forall j \in A \\ & x_{ij} \ge 0 & \forall i \in B, \forall j \in A \end{array} \tag{5.1}$$

where x_{ij} is the amount of good j allocated to buyer i. Interpret Lagrangian variables, say p_j's, corresponding to the second set of conditions as prices of goods. By the Karush, Kuhn, Tucker (KKT) conditions, optimal solutions to x_{ij}'s and p_j's must satisfy the following:

(i) $\forall j \in A : \ p_j \ge 0.$
(ii) $\forall j \in A : \ p_j > 0 \Rightarrow \ \sum_{i \in A} x_{ij} = 1.$
(iii) $\forall i \in B, \forall j \in A : \frac{u_{ij}}{p_j} \le \frac{\sum_{j \in A} u_{ij} x_{ij}}{e_i}.$
(iv) $\forall i \in B, \forall j \in A : x_{ij} > 0 \ \Rightarrow \frac{u_{ij}}{p_j} = \frac{\sum_{j \in A} u_{ij} x_{ij}}{e_i}.$

From these conditions, one can derive that an optimal solution to convex program (5.1) must satisfy the market clearing conditions.

The Eisenberg and Gale program also helps prove, in a very simple manner, the following basic properties of equilibria for the linear case of Fisher's model.

Theorem 5.1 *For the linear case of Fisher's model:*

- *If each good has a potential buyer, equilibrium exists.*
- *The set of equilibrium allocations is convex.*
- *Equilibrium utilities and prices are unique.*
- *If all u_{ij}'s and e_i's are rational, then equilibrium allocations and prices are also rational. Moreover, they can be written using polynomially many bits in the length of the instance.*

PROOF Corresponding to good j there is a buyer i such that $u_{ij} > 0$. By the third KKT condition,

$$p_j \geq \frac{e_i u_{ij}}{\sum_j u_{ij} x_{ij}} > 0.$$

Now, by the second KKT condition, $\sum_{i \in A} x_{ij} = 1$. Hence, prices of all goods are positive and all goods are fully sold.

The third and fourth conditions imply that if buyer i gets good j then j must be among the goods that give buyer i maximum utility per unit money spent at current prices. Hence each buyer gets only a bundle consisting of her most desired goods, i.e., an optimal bundle.

The fourth condition is equivalent to

$$\forall i \in B, \forall j \in A: \quad \frac{e_i u_{ij} x_{ij}}{\sum_{j \in A} u_{ij} x_{ij}} = p_j x_{ij}.$$

Summing over all j gives

$$\forall i \in B: \quad \frac{e_i \sum_j u_{ij} x_{ij}}{\sum_{j \in A} u_{ij} x_{ij}} = \sum_j p_j x_{ij}.$$

This implies

$$\forall i \in B: \quad e_i = \sum_j p_j x_{ij}.$$

Hence the money of each buyer is fully spent. This completes the proof that market equilibrium exists.

Since each equilibrium allocation is an optimal solution to the Eisenberg-Gale convex program, the set of equilibrium allocations must form a convex set.

Since log is a strictly concave function, if there is more than one equilibrium, the utility derived by each buyer must be the same in all equilibria. This fact, together with the fourth condition, gives that the equilibrium prices are unique.

Finally, we prove the fourth claim by showing that equilibrium allocations and prices are solutions to a system of linear equations. Let $q_j = 1/p_j$ be a new variable corresponding to each good j and let k be the number of nonzero x_{ij}'s in an equilibrium allocation. The system will consist of $k + l$ equations over $k + l$ unknowns, the latter being the n q_j's and the k the nonzero x_{ij}'s. The equations are corresponding to each good j, the equality given by the second KKT condition, and corresponding to each nonzero x_{ij}, the equality given by the fourth KKT condition. □

5.3 Checking If Given Prices Are Equilibrium Prices

Let $\boldsymbol{p} = (p_1, \ldots, p_n)$ denote a vector of prices. Let us first devise an algorithm for answering the following question: Is $\boldsymbol{p}$ the equilibrium price vector, and if so, find equilibrium allocations for the buyers.

At prices $\boldsymbol{p}$, buyer i derives u_{ij}/p_j amount of utility per unit money spent on good j. Clearly, she will be happiest with goods that maximize this ratio. Define her *bang per buck* to be $\boldsymbol{\alpha}_i = \max_j\{u_{ij}/p_j\}$. For each $i \in B$, $j \in A$, $\boldsymbol{\alpha}_i \geq u_{ij}/p_j$, with equality holding only if j is i's bang per buck good. If there are several goods maximizing this ratio, she is equally happy with any combination of these goods. This motivates defining the following bipartite graph, G. Its bipartition is (A, B) and for $i \in B$, $j \in A$, (i, j) is an edge in G iff $\boldsymbol{\alpha}_i = u_{ij}/p_j$. We will call this graph the *equality subgraph* and its edges the *equality edges*.

5.3.1 The Network $N(\boldsymbol{p})$

Any goods sold along the edges of the equality subgraph will make buyers happiest, relative to prices $\boldsymbol{p}$. Computing the largest amount of goods that can be sold in this manner, without exceeding the budgets of buyers or the amount of goods available (assumed unit for each good), can be accomplished by computing max-flow in the following network (see Figure 5.1). Direct edges of G from A to B and assign a capacity of infinity to all these edges. Introduce source vertex s and a directed edge from s to each vertex $j \in A$ with a capacity of p_j. Introduce sink vertex t and a directed edge from each vertex $i \in B$ to t with a capacity of e_i. The network is clearly a function of the prices $\boldsymbol{p}$ and will be denoted by $N(\boldsymbol{p})$.

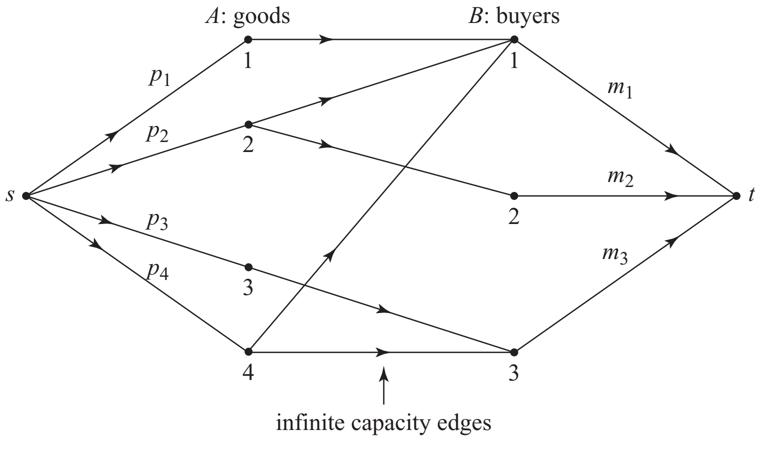

Figure 5.1. The network $N(p)$.

Corresponding to a feasible flow f in network $N(\boldsymbol{p})$, let us define the allocation of goods to the buyers to be the following. If edge (j, i) from good j to buyer i carries flow $f(j, i)$, then buyer i receives $f(j, i)/p_j$ units of good j.

The question posed above can be answered via one max-flow computation, as asserted in the following lemma. Its proof is straightforward and is omitted.

Lemma 5.2 *Prices* $\boldsymbol{p}$ *are equilibrium prices iff in the network* $N(\boldsymbol{p})$ *the two cuts* $(s, A \cup B \cup t)$ *and* $(s \cup A \cup B, t)$ *are min-cuts. If so, allocations corresponding to any max-flow in* N *are equilibrium allocations.*

5.4 Two Crucial Ingredients of the Algorithm

The algorithm starts with very low prices that are guaranteed to be below the equilibrium prices for each good. The algorithm always works on the network $N(p)$ w.r.t. the current prices $\boldsymbol{p}$. W.r.t. the starting prices, buyers have surplus money left. The algorithm raises prices iteratively and reduces the surplus. When the surplus vanishes, it terminates; these prices are equilibrium prices.

This algorithmic outline immediately raises two questions:

- How do we ensure that the equilibrium price of no good is exceeded?
- How do we ensure that the surplus money of buyers reduces fast enough that the algorithm terminates in polynomial time?

The answers to these two questions lead to two crucial ingredients of the algorithm: *tight sets* and *balanced flows*.

5.5 The Primal-Dual Schema in the Enhanced Setting

We will use the notation setup in the previous sections to describe at a high level the new difficulties presented by the enhanced setting of convex programs and the manner in which the primal-dual schema is modified to obtain a combinatorial algorithm for solving the Eisenberg–Gale convex program.

The fundamental difference between complementary slackness conditions for linear programs and KKT conditions for nonlinear convex programs is that whereas the former do not involve both primal and dual variables simultaneously in an equality constraint (obtained by assuming that one of the variables takes a nonzero value), the latter do.

As described in the previous section, the algorithm will start with very low prices and keep increasing them greedily, i.e., the dual growth process is greedy. Indeed, all known primal-dual algorithms use a greedy dual growth process – with one exception, namely Edmonds' algorithm for maximum weight matching in general graphs (Edmonds, 1965).

Now, the disadvantage of a greedy dual growth process is obvious – the fact that a raised dual is "bad," in the sense that it "obstructs" other duals that could have led to a larger overall dual solution, may become clear only later in the run of the algorithm. In view of this, the issue of using more sophisticated dual growth processes has received a lot of attention, especially in the context of approximation algorithms. The problem with such a process is that it will make primal objects go tight and loose and the number of such reversals will have to be upper bounded in the running time analysis. The impeccable combinatorial structure of matching supports such an accounting and in fact this leads to a strongly polynomial algorithm. However, thus far, all attempts at making such a scheme work out for other problems have failed.

In our case, even though the dual growth process is greedy, because of the more complex nature of KKT conditions, edges in the equality subgraph appear and disappear as the algorithm proceeds. Hence, we are forced to carry out the difficult accounting process alluded to above for bounding the running time.

We next point out which KKT conditions the algorithm enforces and which ones it relaxes, as well as the exact mechanism by which it satisfies the latter. Throughout the algorithm, we enforce the first two conditions listed in Section 5.2. As mentioned in Section 5.4, at any point in the algorithm, via a max-flow in the network $N(\boldsymbol{p})$, all goods can be sold; however, buyers may have surplus money left over. W.r.t. a balanced flow in network $N(\boldsymbol{p})$ (see Section 5.7 for a definition of such a flow), let m_i be the money spent by buyer i. Thus, buyer i's surplus money is $\boldsymbol{\gamma}_i = e_i - m_i$. We will relax the third and fourth KKT conditions to the following:

- $\forall i \in B, \forall j \in A : \dfrac{u_{ij}}{p_j} \leq \dfrac{\sum_{j \in A} u_{ij} x_{ij}}{m_i}$.
- $\forall i \in B, \forall j \in A : x_{ij} > 0 \Rightarrow \dfrac{u_{ij}}{p_j} = \dfrac{\sum_{j \in A} u_{ij} x_{ij}}{m_i}$.

Consider the following potential function:

$$\Phi = \boldsymbol{\gamma}_1^2 + \boldsymbol{\gamma}_2^2 + \cdots + \boldsymbol{\gamma}_{n'}^2.$$

We will give a process by which this potential function decreases by an inverse polynomial fraction in polynomial time (in each phase, as detailed in Lemma 5.21). When Φ drops all the way to zero, all KKT conditions are exactly satisfied.

Finally, there is a marked difference between the way this algorithm will satisfy KKT conditions and the way primal-dual algorithms for LP's do. The latter satisfy complementary conditions in *discrete steps*, i.e., in each iteration, the algorithm satisfies at least one new condition. So, if each iteration can be implemented in strongly polynomial time, the entire algorithm has a similar running time. On the other hand, the algorithm for Fisher's linear case satisfies KKT conditions *continuously* – as the algorithm proceeds, the KKT conditions corresponding to each buyer get satisfied to a greater extent.

Observe that at the start of the algorithm, the value of ϕ is a function not just of the number of buyers and goods but of the length of the input (since it depends on the money possessed by buyers). Therefore, even though a phase of the algorithm can be implemented in strongly polynomial time, the running time of the entire algorithm is polynomial and not strongly polynomial. Indeed, obtaining a strongly polynomial algorithm for this problem remains a tantalizing open problem (see Section 5.15).

5.6 Tight Sets and the Invariant

Let $\boldsymbol{p}$ denote the current prices within the run of the algorithm. For a set $S \subseteq A$ of goods, let $\boldsymbol{p}(S)$ denote the total value of goods in S; this is simply the sum of current prices of goods in S. For a set $T \subseteq B$ of buyers, let $m(T)$ denote the total money possessed by the buyers in T; i.e., $m(T) = \sum_{i \in T} e_i$. For $S \subseteq A$, define its *neighborhood in $N(\boldsymbol{p})$*,

$$\Gamma(S) = \{j \in B \mid \exists i \in S \text{ with } (i, j) \in N(\boldsymbol{p})\}.$$

Clearly, $\Gamma(S)$ is the set of buyers who are interested in goods in S at current prices.

We will say that S is a *tight set* if the current value of S exactly equals the money possessed by buyers who are interested in goods in S; i.e., $\boldsymbol{p}(S) = m(\Gamma(S))$. Under this circumstance, increasing prices of goods in S may lead to exceeding the equilibrium price of some good. Therefore, when a set of goods goes tight, the algorithm freezes the prices of all goods in S. As described in Section 5.7, when new edges enter the equality subgraph, the algorithm may unfreeze certain frozen goods and again start increasing their prices.

A systematic way of ensuring that the equilibrium price of no good is exceeded is to ensure the following Invariant.

Invariant: The prices $\boldsymbol{p}$ are such that the cut $(s, A \cup B \cup t)$ is a min-cut in $N(\boldsymbol{p})$.

Lemma 5.3 *For given prices $\boldsymbol{p}$, network $N(\boldsymbol{p})$ satisfies the Invariant iff*

$$\forall S \subseteq A : \boldsymbol{p}(S) \leq m(\Gamma(S)).$$

PROOF The forward direction is trivial, since under max-flow (of value $\boldsymbol{p}(A)$) every set $S \subseteq A$ must be sending $\boldsymbol{p}(S)$ amount of flow to its neighborhood.

Let us prove the reverse direction. Assume that $(s \cup A_1 \cup B_1, A_2 \cup B_2 \cup t)$ is a min-cut in $N(\boldsymbol{p})$, with $A_1, A_2 \subseteq A$ and $B_1, B_2 \subseteq B$ (see Figure 5.2). The capacity of this cut is $\boldsymbol{p}(A_2) + m(B_1)$. Now, $\Gamma(A_1) \subseteq B_1$, since otherwise the cut will have infinite capacity. Moving A_1 and $\Gamma(A_1)$ to the t side also results in a cut. By the condition stated in the Lemma, $\boldsymbol{p}(A_1) \leq m(\Gamma(A_1))$. Therefore, the capacity of this cut is no larger than the previous one and this is also a min-cut in $N(\boldsymbol{p})$. Hence the Invariant holds. □

The Invariant ensures that, at current prices, all goods can be sold. The only eventuality is that buyers may be left with surplus money. The algorithm raises prices systematically, thereby decreasing buyers' surplus money. When $(s \cup A \cup B, t)$ is also a min-cut in $N(\boldsymbol{p})$, by Lemma 5.2, equilibrium has been attained.

5.7 Balanced Flows

Denote the current network, $N(\boldsymbol{p})$, by simply N. We will assume that network N satisfies the Invariant; i.e., $(s, A \cup B \cup t)$ is a min-cut in N. Given a feasible flow f in N, let $R(f)$ denote the residual graph w.r.t. f. Define the *surplus* of buyer i, $\gamma_i(N, f)$, to be the residual capacity of the edge (i, t) with respect to flow f in network N,

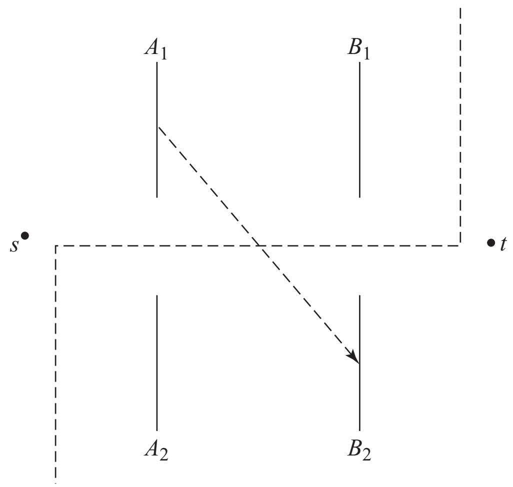

Figure 5.2. Min-cut in $N(p)$. There are no edges from A_1 to B_2.

i.e., e_i minus the flow sent through the edge (i, t). The *surplus vector* is defined to be $\gamma(N, f) := (\gamma_1(N, f), \gamma_2(N, f), \ldots, \gamma_n(N, f))$. Let $\|v\|$ denote the l_2 norm of vector v. A *balanced flow* in network N is a flow that minimizes $\|\gamma(N, f)\|$. A balanced flow must be a max-flow in N because augmenting a given flow can only lead to a decrease in the l_2 norm of the surplus vector.

Lemma 5.4 *All balanced flows in N have the same surplus vector.*

PROOF It is easy to see that if γ_1 and γ_2 are the surplus vectors w.r.t flows f_1 and f_2, then $(\gamma_1 + \gamma_2)/2$ is the surplus vector w.r.t the flow $(f_1 + f_2)/2$. Since the set of feasible flows in N is a convex region, so is the set of all feasible surplus vectors. Since a balanced flow minimizes a strictly concave function of the surplus vector, the optimal surplus vector must be unique. □

The following property of balanced flows will be used critically in the algorithm.[3]

Property 1: If $\gamma_j(N, f) < \gamma_i(N, f)$ then there is no path from node j to node i in $R(f) - \{s, t\}$.

Theorem 5.5 *A maximum-flow in N is balanced iff it satisfies Property 1.*

PROOF Let f be a balanced flow and let $\gamma_i(N, f) > \gamma_j(N, f)$ for some $i, j \in B$. Suppose, for the sake of contradiction, there is a path from j to i in $R(f) - \{s, t\}$.

In N, the only edge out of j is the edge (j, t). Since the path in $R(f)$ from j to i must start with a positive capacity edge which is different from edge (j, t), by flow conservation, the capacity of (t, j) must be positive in $R(f)$. Since $\gamma_i(N, f) > 0$, the edge (i, t) has a positive capacity in $R(f)$. Now, the edges (t, j) and (i, t)

[3] Unlike the previous sections, in Section 5.7, j will denote a buyer.

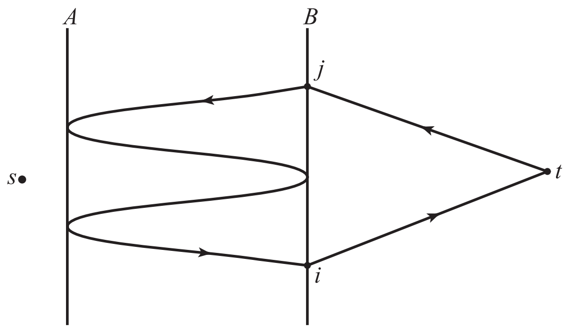

Figure 5.3. The circulation in $R(f)$ if Property 1 does not hold.

concatenated with the path from j to i gives us a cycle with positive residual capacity in $R(f)$ (see Figure 5.3). Sending a circulation of positive value along this cycle will result in another max-flow in which the residual capacity of j is slightly larger and that of i is slightly smaller; i.e., the flow is more balanced. This contradicts the fact that f is a balanced flow.

To prove the other direction, first observe that the l_2 norm of the surplus vector of a max-flow f satisfying Property 1 is locally optimum w.r.t. changes in pairs of components of the surplus vector. This is so because any circulation in $R(f)$ can only send flow from a high surplus buyer to a low surplus buyer resulting in a less balanced flow. Now, since l_2 norm is a strictly concave function, any locally optimal solution is also globally optimal. Hence, a max-flow f satisfying Property 1 must be a balanced flow. □

5.7.1 Finding a Balanced Flow

We will show that the following algorithm, which uses a divide and conquer strategy, finds a balanced flow in the given network N in polynomial time. As stated above, we will assume that this network satisfies the Invariant, i.e., $(s, A \cup B \cup t)$ is a min-cut in N.

Continuously reduce the capacities of all edges that go from B to t, other than those edges whose capacity becomes zero, until the capacity of the cut $(\{s\} \cup A \cup B, \{t\})$ becomes the same as the capacity of the cut $(\{s\}, A \cup B \cup \{t\})$. Let the resulting network be N' and let f' be a max-flow in N'. Find a maximal $s - t$ min-cut in N', say (S, T), with $s \in S$ and $t \in T$.

Case 1: If $T = \{t\}$ then find a max-flow in N' and output it – this will be a balanced flow in N.

Case 2: Otherwise, let N_1 and N_2 be the subnetworks of N induced by $S \cup \{t\}$ and $T \cup \{s\}$, respectively. (Observe that N_1 and N_2 inherit original capacities from N and not the reduced capacities from N'.) Let A_1 and B_1 be the subsets of A and B, respectively, induced by N_1. Similarly, let A_2 and B_2 be the subsets of A and B, respectively, induced by N_2. Recursively find balanced flows, f_1 and f_2, in N_1 and N_2, respectively. Output the flow $f = f_1 \cup f_2$ – this will be a balanced flow in N.

Lemma 5.6 *f is a max-flow in N.*

PROOF In the first case, i.e., $T = \{t\}$, the algorithm outputs a max-flow in N'. This flow must saturate the cut $(\{s\} \cup A \cup B, \{t\})$. However, since the capacity of this cut in N' is the same as the capacity of the cut $(\{s\}, A \cup B \cup \{t\})$, by the Invariant, this is also a max-flow in N.

Next let us consider the second case. Since N_1 and N_2 are edge-disjoint networks, $f = f_1 \cup f_2$ will be a feasible flow in N. We will show that f must saturate all edges from s to A and therefore by the Invariant, it is a max-flow.

Let g be a max-flow in N. Observe that N', and hence N, cannot have any edges from A_1 to B_2. Therefore, all flow of g going to A_1 must flow via B_1. Therefore, the restriction of g to N_1 saturates all edges from s to A_1 in N_1. Therefore, so must f_1 since it is a max-flow in N_1.

Let f' be a max-flow in N'. Since (S, T) is a min-cut in N', f' must saturate all edges from s to A_2. Furthermore, all flow of f' going to A_2 must flow via B_2, i.e., the restriction of f' to flow going through A_2 is a feasible flow in N_2. Since f_2 is a max-flow in N_2, it must also saturate all edges from s to A_2. Hence f saturates all edges from s to A in N, and is therefore a max-flow. □

Lemma 5.7 *f is a balanced flow in network N.*

PROOF We will show, by induction on the depth of recursion, that the max-flow output by the algorithm is a balanced flow in N. In the base case, the algorithm terminates in the first case; i.e., $T = \{t\}$, the surplus vector is precisely the amounts subtracted from capacities of edges from B to t in going from N to N'. Clearly, this surplus vector makes components as equal as possible, thus minimizing its l_2 norm.

Next assume that the algorithm terminates in the second case. By Lemma 5.6, f is a max-flow; we will show that it satisfies Property 1 and is therefore a balanced flow. By the induction hypothesis, f_1 and f_2 are balanced flows in N_1 and N_2, respectively, and therefore Property 1 cannot be violated in these two networks.

Let R be the residual graph of N w.r.t. flow f; we only need to show that paths in R that go from one part to the other do not violate Property 1. As already observed in the proof of Lemma 5.6, there are no edges from A_1 to B_2 in N, and therefore there are no residual paths from $j \in B_1$ to $i \in B_2$. There may however be paths going from $j \in B_2$ to $i \in B_1$ in R. We will show that for any two nodes $i \in B_1$ and $j \in B_2$, $\gamma_i(N, f) < \gamma_j(N, f)$, thereby establishing Property 1.

First observe that by the maximality of the min-cut found in N', all nodes in B_2 have surplus capacity > 0 w.r.t. flow f' in N' (all nodes having surplus zero must be in B_1). Therefore, the same amount, say X, was subtracted from the capac ity of each edge (i, t), $i \in B_2$, in going from network N to N'. We will show that $\gamma_i(N, f) > X$ for each $i \in B_2$. A similar proof shows that $\gamma_i(N, f) < X$ for each $i \in B_1$, thereby establishing Property 1.

Let L be the set of vertices in B_2 having minimum surplus w.r.t. f. Let K be the set of vertices in A_2 that are reachable via an edge from L in R. We claim that $\Gamma(K) = L$, because otherwise, there will be a residual path from $i \in L$ to $j \in B_2 - L$, thereby violating Property 1.

Let $c(K)$ denote the sum of capacities of all edges from s to vertices of K. Observe that all these edges are saturated in f' and this flow must leave via vertices of L. Let E_L denote the set of edges going from L to t. Let $c(L)$ and $c'(L)$ denote the sum of capacities of all edges in E_L in networks N and N', respectively. By the argument given above, $c'(L) > c(K)$.

Since X is subtracted from all edges in E_L in going from network N to N', $c(L) = c'(L) + |L|X$. The total surplus of the edges in E_L w.r.t. flow f is

$$c(L) - c(K) = c'(L) + |L|X - c(K) > |L|X.$$

Finally, since all edges in E_L have the same surplus, each has surplus $> X$. The lemma follows. □

Theorem 5.8 *The above-stated algorithm computes a balanced flow in network N using at most n max-flow computations.*

PROOF Clearly, the number of goods in the biggest piece drops by at least 1 in each iteration. Therefore, the depth of recursion is at most n. Next, observe that N_1 and N_2 are vertex disjoint, other than s and t, and therefore, the time needed to compute max-flows in them is bounded by the time needed to compute a max-flow in N. Hence, the total computational overhead is n max-flow computations. Finally, as shown in Lemma 5.7, the flow output by the algorithm is a balanced flow in N. □

5.8 The Main Algorithm

First we show how to initialize prices so the Invariant holds. The following two conditions guarantee this.

- The initial prices are low enough prices that each buyer can afford all the goods. Fixing prices at $1/n$ suffices, since the goods together cost one unit and all e_i's are integral.
- Each good j has an interested buyer, i.e., has an edge incident at it in the equality subgraph. Compute α_i for each buyer i at the prices fixed in the previous step and compute the equality subgraph. If good j has no edge incident, reduce its price to
$$p_j = \max_i \left\{ \frac{u_{ij}}{\alpha_i} \right\}.$$

If the Invariant holds, it is easy to see that there is a unique maximal tight set $S \subseteq A$. Clearly, the prices of goods in the tight set cannot be increased without violating the Invariant. On the other hand, the algorithm can raise prices of all goods in $A - S$. However, we do not know any way of bounding the running time of any algorithm based on such an approach. In fact, it seems that any such algorithm can be forced to take a large number of steps in which it makes only very small progress toward decreasing the surplus of the buyers, thereby taking super polynomial time.

Instead, we will show how to use the notion of balanced flow to give a polynomial time algorithm. The idea is to always raise prices of those goods which are desired by buyers having a lot of surplus money. Eventually, when a subset of these goods goes tight, the surplus of some of these buyers vanishes, thus leading to substantial progress. Property 1 of balanced flows provides us with a powerful condition to ensure that even as the network $N(\boldsymbol{p})$ changes because of changes in $\boldsymbol{p}$, the algorithm can still keep working with a set of buyers having a large surplus.

The iterative improvement steps follow the spirit of the primal-dual schema: The "primal" variables are the flows in the edges of $N(\boldsymbol{p})$ and the "dual" variables are the current prices. The current flow suggests how to improve the prices and vice versa.

A run of the algorithm is partitioned into *phases*, each phase ends with a new set going tight. Each phase is partitioned into iterations that are defined below.

A phase starts with computation of a balanced flow, say f, in the current network, $N(\boldsymbol{p})$. If the algorithm of Section 5.7 for finding a balanced flow terminates in Case 1, then by Lemma 5.2 the current prices and allocations are equilibrium prices and allocations and the algorithm halts. Otherwise, let δ be the maximum surplus of buyers w.r.t. f. Initialize I to be the set of buyers having surplus δ. Let J be the set of goods that have edges to I in $N(\boldsymbol{p})$. The network induced by $I \cup J$ is called the *active subgraph*.

At this point, we are ready to raise prices of goods in J. However, we would like to do this in such a way that for each buyer $i \in I$, the set of goods she likes best, which are all in J, remains unchanged as prices increase. This can be accomplished by raising prices of goods in J in such a way that the ratio of any two prices remains unchanged. The rest of the algorithm for a phase is as follows.

Step $\diamond$: Multiply the current prices of all goods in J by variable x, initialize x to 1 and raise x continuously until one of the following two events happens. Observe that as soon as $x > 1$, buyers in $B - I$ are no longer interested in goods in J and all such edges can be dropped from the equality subgraph and N.

- **Event 1:** If a subset $S \subseteq J$ goes tight, the current phase terminates and the algorithm starts with the next phase.
- **Event 2:** As prices of goods in J keep increasing, goods in $A - J$ become more and more desirable for buyers in I. If as a result an edge (i, j), with $i \in I$ and $j \in A - J$, enters the equality subgraph (see Figure 5.4). add directed edge (j, i) to network $N(\boldsymbol{p})$ and compute a balanced flow, say f, in the current network, $N(\boldsymbol{p})$. If the balanced flow algorithm terminates in Case 1, halt and output the current prices and allocations. Otherwise, let R be the residual graph corresponding to f. Determine the set of all buyers that have residual paths to buyers in the current set I (clearly, this set will contain all buyers in I). Update the new set I to be this set. Update J to be the set of goods that have edges to I in $N(\boldsymbol{p})$. Go to Step $\diamond$.

To complete the algorithm, we simply need to compute the smallest values of x at which Event 1 and Event 2 happen, and consider only the smaller of these. For Event 2, this is straightforward. We give an algorithm for Event 1 in the next section.

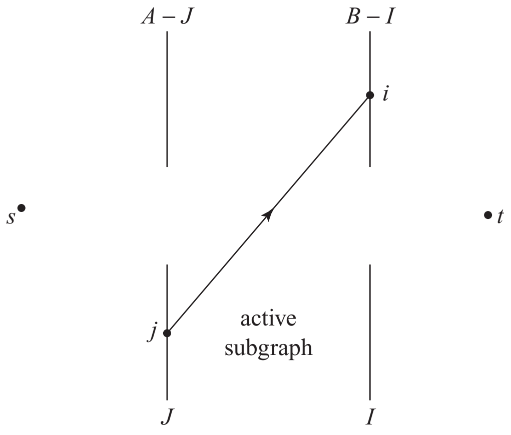

Figure 5.4. If Event 2 happens, edge (j, i) is added to $N(p)$.

5.9 Finding Tight Sets

Let $\boldsymbol{p}$ denote the current price vector (i.e., at $x = 1$). We first present a lemma that describes how the min-cut changes in $N(x \cdot \boldsymbol{p})$ as x increases. Throughout this section, we will use the function m to denote money w.r.t. prices $\boldsymbol{p}$. W.l.o.g. assume that w.r.t. prices $\boldsymbol{p}$ the tight set in G is empty (since we can always restrict attention to the active subgraph, for the purposes of finding the next tight set). Define

$$x^* = \min_{\emptyset \neq S \subseteq A} \frac{m(\Gamma(S))}{m(S)},$$

the value of x at which a nonempty set goes tight. Let S^* denote the tight set at prices $x^* \cdot \boldsymbol{p}$. If $(s \cup A_1 \cup B_1, A_2 \cup B_2 \cup t)$ is a cut in the network, we will assume that $A_1, A_2 \subseteq A$ and $B_1, B_2 \subseteq B$.

Lemma 5.9 *W.r.t. prices $x \cdot \boldsymbol{p}$:*

- *if $x \leq x^*$ then $(s, A \cup B \cup t)$ is a min-cut.*
- *if $x > x^*$ then $(s, A \cup B \cup t)$ is not a min-cut. Moreover, if $(s \cup A_1 \cup B_1, A_2 \cup B_2 \cup t)$ is a min-cut in $N(x \cdot \boldsymbol{p})$ then $S^* \subseteq A_1$.*

PROOF Suppose $x \leq x^*$. By definition of x^*,

$$\forall S \subseteq A : x \cdot m(S) \leq m(\Gamma(S)).$$

Therefore by Lemma 5.3, w.r.t. prices $x \cdot \boldsymbol{p}$, the Invariant holds. Hence $(s, A \cup B \cup t)$ is a min-cut.

Next suppose that $x > x^*$. Since $x \cdot m(S^*) > x^* \cdot m(S^*) = m(\Gamma(S^*))$, w.r.t. prices $x \cdot \boldsymbol{p}$, the cut $(s \cup S^* \cup \Gamma(S^*), t)$ has strictly smaller capacity than the cut $(s \cup A \cup B, t)$. Therefore the latter cannot be a min-cut.

Now consider the min-cut $(s \cup A_1 \cup B_1, A_2 \cup B_2 \cup t)$. Let $S^* \cap A_2 = S_2$ and $S^* - S_2 = S_1$. Suppose $S_2 \neq \emptyset$. Clearly $\Gamma(S_1) \subseteq B_1$ (otherwise the cut will have infinite capacity). If $m(\Gamma(S_2) \cap B_2) < x \cdot m(S_2)$, then by moving S_2 and $\Gamma(S_2)$ to the s side of this cut, we can get a smaller cut, contradicting the minimality of the cut picked. In particular, $m(\Gamma(S^*) \cap B_2) \le m(\Gamma(S^*)) = x^* \cdot m(S^*) < x \cdot m(S^*)$. Therefore $S_2 \ne S^*$, and hence, $S_1 \ne \emptyset$. Furthermore,

$$m(\Gamma(S_2) \cap B_2) \ge x \cdot m(S_2) > x^* m(S_2).$$

On the other hand,

$$m(\Gamma(S_2) \cap B_2) + m(\Gamma(S_1)) \le x^*(m(S_2) + m(S_1)).$$

The two imply that

$$\frac{m(\Gamma(S_1))}{m(S_1)} < x^*,$$

contradicting the definition of x^*. Hence $S_2 = \emptyset$ and $S^* \subseteq A_1$. □

Lemma 5.10 *Let $x = m(B)/m(A)$ and suppose that $x > x^*$. If $(s \cup A_1 \cup B_1, A_2 \cup B_2 \cup t)$ be a min-cut in $N(x \cdot \boldsymbol{p})$ then A_1 must be a proper subset of A.*

PROOF If $A_1 = A$, then $B_1 = B$ (otherwise this cut has ∞ capacity), and $(s \cup A \cup B, t)$ is a min-cut. But for the chosen value of x, this cut has the same capacity as $(s, A \cup B \cup t)$. Since $x > x^*$, the latter is not a min-cut by Lemma 5.9. Hence, A_1 is a proper subset of A. □

Lemma 5.11 *x^* and S^* can be found using n max-flow computations.*

PROOF Let $x = m(B)/m(A)$. Clearly, $x \ge x^*$. If $(s, A \cup B \cup t)$ is a min-cut in $N(x \cdot \boldsymbol{p})$, then by Lemma 5.9, $x^* = x$. If so, $S^* = A$.

Otherwise, let $(s \cup A_1 \cup B_1, A_2 \cup B_2 \cup t)$ be a min-cut in $N(x \cdot \boldsymbol{p})$. By Lemmas 5.9 and 5.10, $S^* \subseteq A_1 \subset A$. Therefore, it is sufficient to recurse on the smaller graph $(A_1, \Gamma(A_1))$. □

5.10 Running Time of the Algorithm

Let $U = \max_{i \in B, j \in A}\{u_{ij}\}$ and let $\Delta = nU^n$.

Lemma 5.12 *At the termination of a phase, the prices of goods in the newly tight set must be rational numbers with denominator $\le \Delta$.*

PROOF Let S be the newly tight set and consider the equality subgraph induced on the bipartition $(S, \Gamma(S))$. Assume w.l.o.g. that this graph is connected (otherwise we prove the lemma for each connected component of this graph). Let $j \in S$. Pick a subgraph in which j can reach all other vertices $j' \in S$. Clearly, at most $2|S| \le 2n$ edges suffice. If j reaches j' with a path of length $2l$, then $p_{j'} = ap_j/b$ where a and b are products of l utility parameters (u_{ik}'s) each. Since alternate edges of this path contribute to a and b, we can partition the u_{ik}'s in this subgraph into two sets such that a and b use u_{ik}'s from distinct sets. These considerations lead easily to showing that $m(S) = p_j c/d$ where $c \leq \Delta$. Now,

$$p_j = m(\Gamma(S))d/c,$$

hence proving the lemma. □

Lemma 5.13 *Consider two phases P and P', not necessarily consecutive, such that good j lies in the newly tight sets at the end of P as well as P'. Then the increase in the price of j, going from P to P', is $\geq 1/\Delta^2$.*

PROOF Let the prices of j at the end of P and P' be p/q and r/s, respectively. Clearly, $r/s > p/q$. By Lemma 5.12, $q \leq \Delta$ and $r \leq \Delta$. Therefore the increase in price of j,

$$\frac{r}{s} - \frac{p}{q} \geq \frac{1}{\Delta^2}.$$

□

Within a phase, we will call each occurrence of Events 1 and 2 an *iteration*.

Lemma 5.14 *The total number of iterations in a phase is bounded by n.*

PROOF After an iteration due to Event 2, at least one new good must move into the active subgraph. Since there is at least one good in the active subgraph at the start of a phase, the total number of iterations in a phase due to Event 2 is at most $n - 1$. Finally, the last iteration in each phase is due to Event 1. The lemma follows. □

Lemma 5.15 *If f and f^* are respectively a feasible and a balanced flow in $N(\boldsymbol{p})$ such that $\gamma_i(\boldsymbol{p}, f^*) = \gamma_i(\boldsymbol{p}, f) - \delta$, for some $i \in B$ and $\delta > 0$, then $\|\gamma(\boldsymbol{p}, f)^*\|^2 \leq \|\boldsymbol{\gamma}(\boldsymbol{p}, f)\|^2 - \delta^2$.*

PROOF Suppose we start with f and get a new flow f' by decreasing the surplus of i by δ, and increasing the surpluses of some other buyers in the process. We show that this already decreases the l_2 norm of the surplus vector by δ^2 and so the lemma follows.

Consider the flow $f^* - f$. Decompose this flow into flow paths and circulations. Among these, augment f with only those that go through the edge (i, t), to get f'. These are either paths that go from s to i to t, or circulations that go from i to t to some i_l and back to i. Then $\gamma_i(f') = \gamma_i(f^*) = \gamma_i(f) - \delta$ and for a set of vertices $i_1, i_2, \ldots, i_k$, we have $\gamma_{i_l}(f') = \gamma_{i_l}(f) + \delta_l$, s.t. $\delta_1, \delta_2, \ldots, \delta_k > 0$ and $\sum_{l=1}^{k} \delta_l \leq \delta$. Moreover, for all l, there is a path from i to i_l in $R(\boldsymbol{p}, f^*)$. Since f^* is balanced, and satisfies Property 1, $\gamma_i(f') = \gamma_i(f^*) \geq \gamma_{i_l}(f^*) \geq \gamma_{i_l}(f')$.

By Lemma 5.16, $\|\boldsymbol{\gamma}(\boldsymbol{p}, f')\|^2 \leq \|\boldsymbol{\gamma}(\boldsymbol{p}, f)\|^2 - \delta^2$ and since f^* is a balanced flow in $N(\boldsymbol{p})$, $\|\boldsymbol{\gamma}(\boldsymbol{p}, f^*)\|^2 \leq \|\boldsymbol{\gamma}(\boldsymbol{p}, f')\|^2$. □

Lemma 5.16 *If $a \geq b_i \geq 0, i = 1, 2, \ldots, n$ and $\delta \geq \sum_{j=1}^{n} \delta_j$ where $\delta, \delta_j \geq 0, j = 1, 2, \ldots, n$, then*

$$\|(a, b_1, b_2, \ldots, b_n)\|^2 \leq \|(a + \delta, b_1 - \delta_1, b_2 - \delta_2, \ldots, b_n - \delta_n)\|^2 - \delta^2.$$

PROOF

$$(a + \delta)^2 + \sum_{i=1}^{n} (b_i - \delta_i)^2 - a^2 - \sum_{i=1}^{n} b_i^2 \geq \delta^2 + 2a \left(\delta - \sum_{i=1}^{n} \delta_i \right) \geq \delta^2.$$

□

Let N_0 denote the network at the beginning of a phase. Assume that the phase consists of k iterations, and that N_t denotes the network at the end of iteration t. Let f_t be a balanced flow in N_t, $0 \leq t \leq k$.

Lemma 5.17 *f_t is a feasible flow in N_{t+1}, for $0 \leq t < k$.*

PROOF The lemma follows from the fact that each of the two actions, raising the prices of goods in J or adding an edge as required in Event 2, can only lead to a network that supports an augmented max-flow. □

Corollary 5.18 *$\|\boldsymbol{\gamma}(N_t)\|$ is monotonically decreasing with t.*

Let δ_t denote the minimum surplus of a buyer in the active subgraph in network N_t, for $0 \leq t < k$; clearly, $\delta_0 = \delta$.

Lemma 5.19 *If $\delta_{t-1} - \delta_t > 0$ then there exists an $i \in H$ such that $\gamma_i(\boldsymbol{p}_{t-1}) - \gamma_i(\boldsymbol{p}_t) \geq \delta_{t-1} - \delta_t$.*

PROOF Consider the residual network $R(\boldsymbol{p}_t, f)$ corresponding to the balanced flow computed at the end of iteration t. By definition of H_t, every vertex $v \in H_t \setminus H_{t-1}$ can reach a vertex $i \in H_{t-1}$ in $R(\boldsymbol{p}_t, f)$ and therefore, by Theorem 5.5, $\gamma_v(\boldsymbol{p}_t) \geq \gamma_i(\boldsymbol{p}_t)$. This means that minimum surplus δ_t is achieved by a vertex i in H_{t-1}. Hence, the surplus of vertex i is decreased by at least $\delta_{t-1} - \delta_t$ during iteration t. □

Lemma 5.20 *If $\delta_{t+1} < \delta_t$ then $\|\boldsymbol{\gamma}(N_t)\|^2 - \|\boldsymbol{\gamma}(N_{t+1})\|^2 \geq (\delta_t - \delta_{t+1})^2$, for $0 \leq t < k$.*

PROOF By Lemma 5.19, if $\delta_{t+1} < \delta_t$ then there is a buyer i whose surplus drops by $\delta_t - \delta_{t+1}$ in going from f_t to f_{t+1}. By Lemmas 5.15 and 5.17, we get the desired conclusion. □

Lemma 5.21 *In a phase, the square of the l_2 norm of the surplus vector drops by a factor of*

$$\left(1 - \frac{1}{n^2}\right).$$

PROOF We will first prove that

$$\|\boldsymbol{\gamma}(N_0)\|^2 - \|\boldsymbol{\gamma}(N_k)\|^2 \geq \frac{\delta^2}{n}.$$

Observe that the left-hand side can be written as a telescoping sum in which each term is of the form $\|\boldsymbol{\gamma}(N_t)\|^2 - \|\boldsymbol{\gamma}(N_{t+1})\|^2$. By Corollary 5.18, each of these terms is positive. Consider only those terms in which the difference $\delta_t - \delta_{t+1} > 0$. Their sum is minimized when all these differences are equal. Now using Lemma 5.20 and the fact that $\delta_0 = \delta$ and $\delta_k = 0$, we get that

$$\|\boldsymbol{\gamma}(N_0)\|^2 - \|\boldsymbol{\gamma}(N_k)\|^2 \geq \frac{\delta^2}{k}.$$

By Lemma 5.14, $k \leq n$, giving the desired inequality.

The above-stated inequality and the fact that $\|\boldsymbol{\gamma}(N_0)\|^2 \leq n\delta^2$ gives us

$$\|\boldsymbol{\gamma}(N_k)\|^2 \leq \|\boldsymbol{\gamma}(N_0)\|^2 \left(1 - \frac{1}{n^2}\right).$$

The lemma follows. □

Theorem 5.22 *The algorithm finds equilibrium prices and allocations for linear utility functions in Fisher's model using*

$$O(n^4(\log n + n \log U + \log M))$$

max-flow computations.

PROOF By Lemma 5.21, the square of the surplus vector drops by a factor of half after $O(n^2)$ phases. At the start of the algorithm, the square of the surplus vector is at most M^2. Once its value drops below $1/\Delta^4$, the algorithm achieves equilibrium prices. This follows from Lemmas 5.12 and 5.13 Therefore the number of phases is

$$O(n^2 \log(\Delta^4 M^2) \; = \; O(n^2(\log n + n \log U + \log M)).$$

By Lemma 5.14 each phase consists of n iterations and by Lemma 5.11 each iteration requires n max-flow computations. The theorem follows. □

5.11 The Linear Case of the Arrow–Debreu Model

The Arrow–Debreu model is also known as the Walrasian model or the exchange model, and it generalizes Fisher's model. Consider a market consisting of a set A of agents and a set G of goods; assume $|G| = n$ and $|A| = m$. Each agent i comes to the market with an initial endowment of goods, $e_i = (e_{i1}, e_{i2}, \ldots, e_{in})$. We may assume w.l.o.g. that the total amount of each good is unit, i.e., for $1 \leq j \leq n$, $\sum_{i=1}^{m} e_{ij} = 1$. Each agent has linear utilities for these goods. The utility of agent i on deriving x_{ij} amount of good j, for $1 \leq j \leq n$, is $\sum_{j=1}^{n} u_{ij} x_{ij}$.

The problem is to find prices $\boldsymbol{p} = (p_1, \ldots, p_m)$ for the goods so that if each agent sells her initial endowment at these prices and buys her optimal bundle, the market clears; i.e., there is no deficiency or surplus of any good. An agent may have more than one optimal bundle; we will assume that we are free to give each agent any optiaml bundle to meet the market clearing condition.

Observe that a Fisher market with linear utilities, n goods, and m buyers reduces to an Arrow–Debreu market with linear utilities, $n + 1$ goods and $m + 1$ agents as follows. In the Arrow–Debreu market, we will assume that money is the $n + 1$'st good, the first m agents correspond to the m buyers whose initial endowment is the money they come to the market with and the $m + 1$'st agent's initial endowment is all n goods. The first m agents have utilities for goods, as given by the Fisher market and no utility for money, whereas the $m + 1$'st agent has utility for money only.

We define the following terms for the algorithm below. For agent i, let $a_i = \sum_{j=1}^{m} e_{ij}$. Let $a_{\min}$ be the minimum among a_i, $1 \leq i \leq m$. Denote by $p_{\max}$ the maximum price assigned to a good by the algorithm. Denote by $u_{\min}$ and $u_{\max}$ the minimum and maximum values of u_{ij} over all agents i and goods j.

5.12 An Auction-Based Algorithm

We will present an auction-based algorithm for the linear case of the Arrow–Debreu model. It will find an approximate equilibrium in the following sense. For any fixed $\epsilon > 0$, it will find prices $\boldsymbol{p}$ for the goods such that the market clears and each agent gets a bundle of goods that provides her utility at least $(1 - \epsilon)^2$ times the utility of her optimal bundle.

The algorithm initializes the price of each good to be unit, computes the worth of the initial endowment of each agent, and gives this money to each agent. All goods are initially fully unsold.

We will denote by $\boldsymbol{p} = (p_1, p_2, \ldots, p_n)$ the vector of prices of goods at any point in the algorithm. As $\boldsymbol{p}$ changes, the algorithm recomputes the value of each agent's initial endowment and updates her money accordingly. Clearly, at the start of the algorithm, the total surplus (unspent) money of all agents is n.

At any point in the algorithm, a part of good j is sold at price p_j and part of it is sold at $(1 + \epsilon)p_j$. The run of the algorithm is partitioned into iterations. Each iteration terminates when the price of some good is raised by a factor of $(1 + \epsilon)$. Each iteration is further partitioned into rounds. In a round, the algorithm considers agents one by one in some arbitrary but fixed order, say $1, 2, \ldots, m$. If the agent being considered, i, has no surplus money, the algorithm moves to the next agent. Otherwise, it finds i's optimal good, in terms of bang per buck, at current prices; say, it is good j. It then proceeds to execute the operation of *outbid*. This entails buying back good j from agents who have it at price p_j and selling it to i at price $p_j(1 + \epsilon)$. This process can end in one of two ways:

- Agent i's surplus money is exhausted. If so, the algorithm moves on to the next agent.
- No agent has good j at price p_j anymore. If so, it raises the price of good j to $p_j(1+\epsilon)$ by setting p_j to $p_j(1+\epsilon)$. The current iteration terminates and agents' moneys are updated because of this price rise.

When the current round comes to an end, the algorithm checks if the total surplus money with the buyers is at most $\epsilon a_{\min}$. If so, the algorithm terminates. Otherwise, it goes to the next round.

At termination, the algorithm gives the unsold goods to an arbitrary agent to ensure that the market clears. It outputs the allocations received by all agents and the terminating prices $\boldsymbol{p}$. Observe, however, that some of good j may have been sold at price $(1+\epsilon)p_j$ even though the equilibrium price of good j is p_j. Because of this descrepancy, agents will only get approximately optimal bundles. Lemma 5.25 will establish a bound on the approximation factor.

Lemma 5.23 *The number of rounds executed in an iteration is bounded by*

$$O\left(\frac{1}{\epsilon}\log\frac{np_{\max}}{\epsilon a_{\min}}\right).$$

PROOF Observe that if outbid buys a good at price p_j, it sells it at price $(1+\epsilon)p_j$, thereby decreasing the overall surplus. Therefore, in each round that is fully completed (i.e., does not terminate mid-way because of a price increase), the total surplus of agents is reduced by a factor of $(1+\epsilon)$. The total surplus at the beginning of the iteration is at most the total money possessed by all agents, i.e., $np_{\max}$. The iteration terminates (and in fact the algorithm terminates) as soon as the total surplus is at most $\epsilon a_{\min}$. Therefore, a bound on the number of rounds in an iteration is

$$\log_{1+\epsilon}\frac{np_{\max}}{\epsilon a_{\min}}.$$

□

Lemma 5.24 *The total number of iterations is bounded by*

$$O\left(\frac{n}{\epsilon}\log p_{\max}\right).$$

PROOF Each iteration raises the price of a good by a factor of $(1+\epsilon)$. Therefore the number of iterations is bounded by

$$n\log_{1+\epsilon} p_{\max}.$$

□

Lemma 5.25 *Relative to terminating prices, each agent gets a bundle of goods that provides her utility at least* $(1-\epsilon)^2$ *times the utility of her optimal bundle.*

PROOF The algorithm always sells an agent her optimal goods relative to current prices $\boldsymbol{p}$ (recall, however, that at the time of the sale, an agent is charged a price of $(1+\epsilon)p_j$ for good j). There are two reasons why an agent i may end up with a suboptimal bundle in the end. First, at termination, part of her money may remain unspent. Let M denote the total worth of i's initial endowment at terminating prices. Assume that she spent M_1 of this. Since the total surplus money left at termination is at most $\epsilon a_{\min}$, $M_1 \geq (1-\epsilon)M$.

The second reason is that some part of good j may have been sold at price $(1+\epsilon)p_j$ to agent i, even though the equilibrium price announced is p_j. Equivalently, we may assume that i gets her optimal goods at prices $\boldsymbol{p}$ for a fraction of her money. The latter is at least

$$\frac{M_1}{1+\epsilon} \geq \frac{(1-\epsilon)M}{1+\epsilon} \geq (1-\epsilon)^2 M$$

money. The lemma follows. □

Theorem 5.26 *The algorithm given above finds an approximate equilibrium for the linear case of the Arrow–Debreu model in time*

$$O\left(\frac{mn}{\epsilon^2}\log\frac{nv_{\max}}{\epsilon a_{\min} v_{\min}}\log\frac{v_{\max}}{v_{\min}}\right).$$

PROOF Observe that each good whose price is raised beyond 1 is fully sold. Since the total money of agents is the total worth of all goods at prices $\boldsymbol{p}$, the condition that the total surplus money of agents is at most $\epsilon a_{\min}$ must be reached before the price of all goods increases beyond 1. Hence at termination, the price of at least one good is 1.

Clearly, at termination, the ratio of maximum to minimum price of a good is bounded by $v_{\max}/v_{\min}$. Therefore, $p_{\max}$ is bounded by $v_{\max}/v_{\min}$. Each round is executed in $O(m)$ time. Now the bound on the total running time follows from Lemmas 5.23 and 5.24. □

5.13 Resource Allocation Markets

Kelly considered the following general setup for modeling resource allocation. Let R be a set of resources and c: $R \rightarrow \mathbf{Z}^+$ be the function specifying the available capacity of each resource $r \in R$. Let $A = \{a_1, \ldots, a_n\}$ be a set of agents and $m_i \in \mathbf{Z}^+$ be the money available with agent a_i.

Each agent wants to build as many *objects* as possible using resources in R. An agent may be able to use several different subsets of R to make one object. Let $S_{i1}, S_{i2}, \ldots, S_{ik_i}$ be subsets of R usable by agent a_i, $k_i \in \mathbf{Z}^+$. Denote by x_{ij} the number of objects a_i makes using the subset S_{ij}, $1 \leq j \leq k_i$; x_{ij} is not rquired to be integral. Let $f_i = \sum_{j=1}^{k_i} x_{ij}$ be the total number of objects made by agent a_i. We will say that

$f_i, 1 \leq i \leq n$ is *feasible* if simultaneously each agent a_i can make f_i objects without violating capacity constraints on R.

Kelly gave the following convex program and showed that an optimal solution to it satisfies *proportional fairness*; i.e., if f_i^* is an optimal solution and f_i is any feasible solution, then

$$\sum_{i=1}^{n} \frac{f_i - f_i^*}{f_i^*} \leq 0.$$

Intuitively, the only way of making an agent happier by 5% is to make other agents unhappy by at least a total of 5%.

$$\begin{array}{lll}
\text{Maximize} & \displaystyle\sum_{a_i \in A} m_i \log f_i & \\
\text{Subject to} & f_i = \displaystyle\sum_{j=1}^{k_i} x_{ij} & \forall a_i \in A \\
 & \displaystyle\sum_{(ij):r \in S_{ij}} x_{ij} \leq c(r) & \forall r \in R \\
 & x_{ij} \geq 0 & \forall a_i \in A, 1 \leq j \leq k_i
\end{array} \qquad (5.2)$$

This general setup can be used to model many situations. The following are examples of situations of a combinatorial nature.

(i) **Market 1 (flow market):** Given a directed or undirected graph $G = (V, E)$, E is the set of resources, with capacities specified. Agents are source-sink pairs of nodes, $(s_1, t_1), \ldots, (s_k, t_k)$, with money $m_1, \ldots, m_k$, respectively. Each $s_i - t_i$ path is an object for agent (s_i, t_i).

(ii) **Market 2:** Given a directed graph $G = (V, E)$, E is the set of resources, with capacities specified. Agents are $A \subset V$, each with specified money. For $s \in A$ objects are branchings rooted at s and spanning all V.

(iii) **Market 3:** Same as above, except the graph is undirected and the objects are spanning trees.

Using KKT conditions, one can show that an optimal solution to this convex program is an equilibrium solution. Let $p_r, r \in R$ be Lagrangian variables corresponding to the second set of conditions; we will interpret these as prices of resources. By the KKT conditions optimal solutions to x_{ij}'s and p_r's must satisfy the following equilibrium conditions:

(i) Resource $r \in R$ has positive price only if it is used to capacity.

(ii) Each agent uses only the cheapest sets to make objects.

(iii) The money of each agent is fully used up.

Since the objective function of convex program (5.2) is strictly concave, one can see that at optimality, the vector $f_1, \ldots, f_n$ is unique. Clearly, this also holds for every equilibrium allocation.

5.14 Algorithm for Single-Source Multiple-Sink Markets

In this section, we consider the special case of a flow market, Market 1, with a single source and multiple sinks. We will assume that the underlying graph is directed. In case it is undirected, one can use the standard reduction from undirected graphs to directed graphs – replace each undirected edge (u, v) with the two edges (u, v) and (v, u) of the same capacity.

Formally, let $G = (V, E)$ be a directed graph with capacities on edges. Let $s \in V$ be the source node and $T = \{t_1, \ldots, t_r\}$ be the set of sink nodes, also called terminals. Let m_i be the money possessed by sink t_i. The problem is to determine equilibrium flow and edge prices. The following example may help appreciate better some of the intricacies of this problem.

Example 5.27 Consider graph $G = (V, E)$ with $V = \{s, a, b, c, d\}$ and sinks b and d with \$120 and \$10, respectively. The edges are $(s, a), (s, c)$ having capacity 2, (a, b) having capacity 1, and $(a, d), (c, d), (c, b)$ having capacity 10 (see Figure 5.5). The unique equilibrium prices are $p_{(s,a)} = \$10$, $p_{(a,b)} = \$30$, $p_{(s,c)} = \$40$, and the rest of the edges have zero price. At equilibrium, flow on path s, a, d is 1, on s, a, b is 1, and on s, c, b is 2. Simulating the algorithm below on this example will reveal the complex sequence of cuts it needs to find in order to compute the equilibrium. Computing equilibrium for other values of money is left as an intersting exercise.

We will present a strongly polynomial algorithm for this problem which is based on the primal-dual schema; i.e., it alternately adjusts flows and prices, attempting to satisfy all KKT conditions. Often, primal-dual algorithms can naturally be viewed as executing an auction. This viewpoint is leads to a particularly simple way of presenting the current algorithm. We will describe it as an ascending price auction in which the buyers are sinks and sellers are edges. The buyers have fixed budgets and are trying to maximize the flow they receive and the sellers are trying to extract as high a price as possible from the buyers. One important deviation from the usual auction situation is

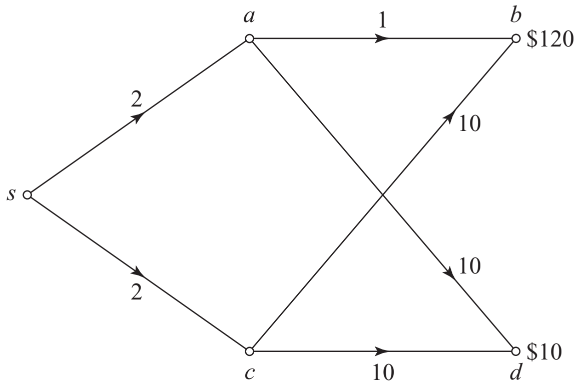

Figure 5.5. The network for Example 5.27.

that the sellers act in a highly coordinated manner – at any point in the algorithm, all edges in a particular cut, say $(S, \overline{S})$, raise their prices simultaneously while prices of the remaining edges remain unchanged. The prices of all edges are initialized to zero. The first cut considered by the algorithm is the (unique) maximal min-cut separating all sinks from s, say $(S_0, \overline{S_0})$.

Denote by rate(t_i) the cost of the cheapest $s - t_i$ path w.r.t. current prices. The flow demanded by sink t_i at this point is $m_i/\text{rate}(t_i)$. At the start of the algorithm, when all edge prices are zero, each sink is demanding infinite flow. Therefore, the algorithm will not be able to find a feasible flow that satisfies all demands. Indeed, this will be the case all the way until termination; at any intermediate point, some cuts will need to be oversaturated in order to meet all the demand.

The price of edges in cut $(S, \overline{S})$ is raised as long as the demand across it exceeds supply; i.e., the cut is oversaturated because of flow demanded by sinks in $\overline{S}$. At the moment that demand exactly equals supply, the edges in this cut stop raising prices and declare themselves sold at current prices. This makes sense from the viewpoint of the edges in the cut – if they raise prices any more, demand will be less than supply; i.e., the cut will be under-saturated, and then these edges will have to be priced at zero!

The crucial question is: when does the cut $(S, \overline{S})$ realize that it needs to sell itself? This point is reached as soon as there is a cut, say $(U, \overline{U})$, with $S \subset U$, such that the difference in the capacities of the two cuts is precisely equal to the flow demanded by sinks in $\overline{S} - \overline{U}$ (see Figure 5.6). Let $(U, \overline{U})$ be the maximal such cut (it is easy to see that it will be unique). If $U = V$, the algorithm halts. Otherwise, cut $(U, \overline{U})$ must be oversaturated – it assumes the role of $(S, \overline{S})$ and the algorithm goes to the next iteration.

Note that an edge may be present in more than one cut whose price is raised by the algorithm. If so, its price will be simply the sum of the prices assigned to these cuts.

Suppose that the algorithm executes k iterations. Let $(S_i, \overline{S_i})$ be the cut it finds in iteration i, $1 \le i \le k$, with $S_k = V$. Clearly, we have $S_0 \subset S_1 \subset \cdots \subset S_k = V$. Let T_i be the set of terminals in $S_i - S_{i-1}$, for $1 \le i \le k$. Let c_i be the set of edges of G in the cut $(S_i, \overline{S_i})$, for $0 \le i < k$ and p_i be the price assigned to edges in c_i. Clearly, for each terminal $t \in T_i$, $rate(t) = p_0 + \cdots + p_{i-1}$, for $1 \le i \le k$.

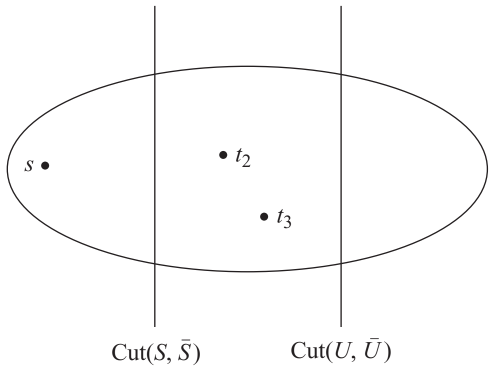

Figure 5.6. The total flow demanded by t_2 and t_3 equals the difference in capacities of cut $(S, \overline{S})$ and cut $(U, \overline{U})$.

Let G' denote the graph obtained by adding a new sink node t to G and edges (t_i, t) from each of the original sinks to t. Let the capacity of edge (t_i, t) be $m_i/\text{rate}(t_i)$. For convenience, even in G', we will denote $V - S$ by $\overline{S}$. It is easy to see that each of the cuts $(S_i, \overline{S_i} \cup \{t\})$ in G' has the same capacity, for $0 \le i \le k$, and each of these $k+1$ cuts is a mininimum $s - t$ cut in G'.

Let f' denote a maximum $s - t$ flow in G'. Obtain flow f from f' by ignoring flow on the edges into t. Then f is a feasible flow in G that sends $m_i/\text{rate}(t_i)$ flow to each sink t_i.

Lemma 5.28 *Flow f and the prices found by the algorithm constitute an equilibrium flow and prices.*

PROOF We will show that flow f and the prices found satisfy all KKT conditions.

- Since each of the cuts $(S_i, \overline{S_i} \cup \{t\})$, for $0 \le i < k$ is saturated in G' by flow f', each of the cuts $c_0, c_1, \ldots, c_{k-1}$ is saturated by f. Hence, all edges having nonzero prices must be saturated.
- The cost of the cheapest path to terminal $t' \in T$ is $\text{rate}(t')$. Clearly, every flow to t' uses a path of this cost.
- Since the flow sent to $t' \in T$ is $m_i/\text{rate}(t')$, the money of each terminal is fully spent. □

Below we give a strongly polynomial time subroutine for computing the next cut in each iteration.

5.14.1 Finding the Next Cut

Let $(S, \overline{S})$ be the cut in G, whose price is being raised in the current iteration and let c be the set of edges in this cut and f its capacity. Let T' denote the set of sinks in $\overline{S}$. Let p' denote the sum of the prices assigned to all cuts found so far in the algorithm (this is a constant for the purposes of this subroutine) and let p denote the price assigned to edges in c. The cut $(S, \overline{S})$ satisfies the following conditions:

- It is a maximal minimum cut separating T' from s.
- At $p = 0$, every cut $(U, \overline{U})$, with $S \subseteq U$, is oversaturated.

Let p^* be the smallest value of p at which there is a cut $(U, \overline{U})$, with $S \subset U$, in G such that the difference in the capacities of $(S, \overline{S})$ and $(U, \overline{U})$ is precisely equal to the flow demanded by sinks in $U - S$ at prices p^*; moreover, $(U, \overline{U})$ is the maximal such cut. Below we give a strongly polynomial algorithm for finding p^* and $(U, \overline{U})$.

Define graph G' by adding a new sink node t to G and edges (t_i, t) for each sink $t_i \in \overline{S}$. Define the capacity of edge (t_i, t) to be $m_i/(p' + p)$ where m_i is the money of sink t_i (see Figure 5.7). As in Section 5.14 we will denote $V - S$ by $\overline{S}$ even in G'. The proof of the following lemma is obvious.

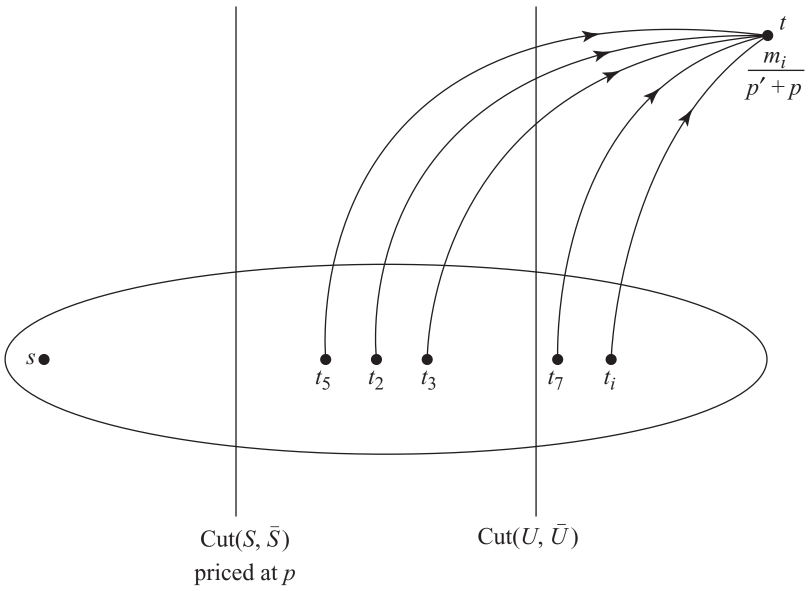

Figure 5.7. Graph G'.

Lemma 5.29 *At the start of the current iteration, $(S, \overline{S} \cup \{t\})$ is a maximal minimum $s - t$ cut in G'. p^* is the smallest value of p at which a new minimum $s - t$ cut appears in G'. $(U, \overline{U} \cup \{t\})$ is the maximal minimum $s - t$ cut in G' at price p^*.*

For any cut C in G', let $\text{cap}_p(C)$ denote its capacity, assuming that the prices of edges in c is p. For $p \geq 0$, define $\text{cut}(p)$ to be the maximal $s - t$ min-cut in G' assuming that the price assigned to edges in c is p. For cut $(A, \overline{A} \cup \{t\})$, $A \subseteq V$, let $\text{price}(A, \overline{A} \cup \{t\})$ denote the smallest price that needs to be assigned to edges in c to ensure that $\text{cap}_p(A, \overline{A} \cup \{t\}) = f$; i.e., $(A, \overline{A} \cup \{t\})$ is also a min $s - t$ cut in G'; if $(A, \overline{A} \cup \{t\})$ cannot be made a minimum $s - t$ cut for any price p then $\text{price}(A, \overline{A} \cup \{t\}) = \infty$. Clearly, $\text{price}(A, \overline{A} \cup \{t\}) \geq p^*$. Observe that determining $\text{price}(A, \overline{A} \cup \{t\})$ involves simply solving an equation in which p is unknown.

Lemma 5.30 *Suppose $p > p^*$. Let* $\text{cut}(p) = (A, \overline{A} \cup \{t\})$, *where $A \neq U$. Let* $\text{price}(A, \overline{A} \cup \{t\}) = q$ *and* $\text{cut}(q) = (B, \overline{B} \cup \{t\})$. *Then $B \subset A$.*

PROOF Since we have assumed that $A \neq U$, it must be the case that $\text{cap}_p(A, \overline{A} \cup \{t\}) > f$. Therefore, $q = \text{price}(A, \overline{A} \cup \{t\}) < p$. Let c_A and c_B denote the capacities of $(A, \overline{A} \cup \{t\})$ and $(B, \overline{B} \cup \{t\})$ at price $p = 0$. Let m_A and m_B denote the money possessed by sinks in $(A - S)$ and $(B - S)$, respectively.

Since $(A, \overline{A} \cup \{t\})$ is a maximal $s - t$ mincut at price p,

$$c_A + \frac{m_A}{p} < c_B + \frac{m_B}{p}.$$

```
Subroutine
Inputs: Cut $(S, \overline{S})$ in $G$ whose price is being raised in the current iteration.
Output: Price $p^*$ and next cut $(U, \overline{U})$.

  (i) $C \leftarrow (V, t)$
 (ii) $p \leftarrow$ price($C$)
(iii) While cut($p$) $\neq C$ do:
        (a) $C \leftarrow$ cut($p$)
        (b) $p \leftarrow$ price($C$)
 (iv) Output $(C, p)$
```

Figure 5.8. Subroutine for finding next cut.

Since $(B, \overline{B} \cup \{t\})$ is a maximal $s - t$ mincut at price q,

$$c_B + \frac{m_B}{q} < c_A + \frac{m_A}{q}.$$

The two together imply

$$\frac{m_B - m_A}{q} < c_A - c_B < \frac{m_B - m_A}{p}.$$

First suppose that $A \subset B$. Clearly $m_A \leq m_B$. But this contradicts the last inequality since $q < p$.

Next, suppose that A and B cross. By the last inequality above, there must be a price, r, such that $q < r < p$ at which $\text{cap}_r(A, \overline{A} \cup \{t\}) = \text{cap}_r(B, \overline{B} \cup \{t\}) = g$, say. By the submodularity of cuts, one of the following must hold:

(i) $\text{cap}_r((A \cap B), \overline{(A \cap B)} \cup \{t\}) \leq g$. Since the money possessed by sinks in $(A \cap B) - S$ is at most m_B, at price q, $\text{cap}_q((A \cap B), \overline{(A \cap B)}\{t\}) < \text{cap}_q(B, \overline{B} \cup \{t\})$. This contradicts the fact that $(B, \overline{B} \cup \{t\})$ is a min-cut at price q.

(ii) $\text{cap}_r((A \cup B), \overline{(A \cup B)} \cup \{t\}) \leq g$. Since the money possessed by sinks in $(A \cup B) - S$ is at least m_A, at price p, $\text{cap}_p((A \cup B), \overline{(A \cup B)} \cup \{t\}) < \text{cap}_p(A, \overline{A} \cup \{t\})$. This contradicts the fact that $(A, \overline{A} \cup \{t\})$ is a min-cut at price p.

Hence we get that $B \subset A$. □

Lemma 5.31 *Subroutine 5.8 terminates with the cut $(U, \overline{U} \cup \{t\})$ and price p^* in at most r max-flow computations, where r is the number of sinks.*

PROOF As long as $p > p^*$, by Lemma 5.30, the algorithm keeps finding smaller and smaller cuts, containing fewer sinks on the s side. Therefore, in at most r iterations, it must arrive at a cut such that $p = p^*$. Since cut$(p^*) = (U, \overline{U} \cup \{t\})$, the next cut it considers is $(U, \overline{U} \cup \{t\})$. Since price$(U, \overline{U} \cup \{t\}) = p^*$, at this point the algorithm terminates. □

Theorem 5.32 *The algorithm given in Section 5.14 finds equilibrium edge prices and flows using $O(r^2)$ max-flow computations, where r is the number of sinks.*

PROOF Clearly, the number of sinks trapped in the sets $S_0 \subset S_1 \subset \cdots \subset S_k$ keeps increasing and therefore, the number of iterations $k \leq r$. The running time for each iteration is dominated by the time taken by subroutine (5.8), which by Lemma 5.31 is r max-flow computations. Hence the total time taken by the algorithm is $O(r^2)$ max-flow computations. By Lemma 5.28 the flow and prices found by the algorithm are equilibrium flow and prices. □

5.15 Discussion and Open Problems

Linear utility functions provided us with perhaps the easiest algorithmic questions that helped us commence our algorithmic study of market equilibria. However, such functions are much too restrictive to be useful. Concave utility functions are considered especially useful in economics because they model the important condition of decreasing marginal utilities as a function of the amount of good obtained. Furthermore, if the utility functions are strictly concave, at any given prices, there is a unique optimal bundle of goods for each agent. This leads to the following remarkable communication complexity fact: In such a market, it suffices to simply announce equilibrium prices – then, all agents can individually compute and buy their optimal bundles and the market clears!

On the other hand, concave utility functions, even if they are additively separable over the goods, are not easy to deal with algorithmically. In fact, obtaining a polynomial time algorithm for such functions is a premier open problem today. For the case of linear functions, the approach used in Section 5.8 – of starting with very low prices and gradually raising them until the equilibrium is reached – is made possible by the property of weak gross substitutability. This property holds for a utility function if on raising the price of one good, the demand of another good cannot go down. As a consequence of this property, the need to decrease the price of the second good does not arise.

Concave utility functions do not satisfy weak gross substitutability. Exercises 5.5 and 5.6 outline an approach that attempts to finesse this difficulty for the case of piecewise-linear, concave functions. Does this approach lead to an efficient algorithm for computing, either exactly or approximately, equilibrium prices for such functions? If so, one can handle a concave function by approximating it with a piecewise-linear, concave function. Alternatively, can one show that finding an equilibrium for such utility functions is PPAD-hard?

Considering the properties of the linear case of Fisher's model established in Theorem 5.1, one wonders whether its equilibrium allocations can be captured via a linear program. Resolving this, positively or negatively, seems an exciting problem. Another question remaining open is whether there is a strongly polynomial algorithm for computing equilibrium prices for this case. Finally, we would like to point to the numerous questions remaining open for gaining a deeper algorithmic understanding of Eisenberg–Gale markets (Jain and Vazirani, 2006).

Acknowledgments

I wish to thank Deeparnab Chakrabarty, Nikhil Devanur, Sergei Izmalkov, Kamal Jain and Kasturi Vardarajan for valuable discussions and comments on the writeup.

Bibliography

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

W.C. Brainard and H.E. Scarf. How to compute equilibrium prices in 1891. *Cowles Foundation Discussion Paper*, (1270) 2000.

X. Deng, C. Papadimitriou, and S. Safra. On the complexity of equilibria. In *Proc. ACM Symp. on Theor. Comp.*, 2002.

N. Devanur, C.H. Papadimitriou, A. Saberi, and V.V. Vazirani. Market equilibrium via a primal-dual-type algorithm. In *Proc. IEEE Annual Symp. Fdns. of Comp. Sci.*, 2002. To appear in *J. ACM*. Journal version available at: `http://www-static.cc.gatech.edu/vazirani/market.ps`.

N. Devanur and V.V. Vazirani. The spending constraint model for market equilibrium: Algorithmic, existence and uniqueness results. In *Proc. 36th Symp. on Theory of Computing*, 2004.

J. Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *J. Res. Natl. Bur. Standards*, 69:125–130, 1965.

J. Edmonds. Optimum branchings. *J. Res. Natl. Bur. Standards, Section B*, 71:233–240, 1967.

E. Eisenberg and D. Gale. Consensus of subjective probabilities: The Pari-Mutuel method. *Annals Math. Stat.*, 30:165–168, 1959.

S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Networking*, 1(1):397–413, 1993.

R. Garg and S. Kapoor. Auction algorithms for market equilibrium. In *Proc. 36th Symp. on Theory of Computing*, 2004.

V. Jacobson. Congestion avoidance and control. In *ACM SIGCOMM*, pp. 314–329, 1988.

K. Jain and V.V. Vazirani. Eisenberg-gale markets: Algorithms and structural properties. In *Proc. 39th Symp. on Theory of Computing*, 2007.

F.P. Kelly. Charging and rate control for elastic traffic. *Euro. Trans. on Telecomm.*, 8:33–37, 1997.

F.P. Kelly, A.K. Maulloo, and D.K.H. Tan. Rate control in communication networks. *J. Oper. Res. Soc.*, 49:237–252, 1998.

F.P. Kelly and V.V. Vazirani. Rate control as a market equilibrium. Unpublished manuscript 2002. Available at: `http://www-static.cc.gatech.edu/vazirani/KV.pdf`.

S. Low and D. Lapsley. Optimization flow control, 1: basic algorithm and convergence. *IEEE/ACM Trans. Networking*, 7(6):861–874, 1999.

C.S.J.A. Nash-Williams. Edge-disjoint spanning trees of finite graphs. *J. London Math. Soc.*, 36:445–450, 1961.

H. Scarf. *The Computation of Economic Equilibria (with collaboration of T. Hansen)*. Cowles Foundation Monograph No. 24., New Haven: Yale University Press, 1973.

W.T. Tutte. On the problem of decomposing a graph into n connected factors. *J. London Math. Soc.*, 36:221–230, 1961.

V.V. Vazirani. Spending constraint utilities, with applications to the Adwords market. Submitted to *Math. of Operations Research*, 2006.

L. Walras. *Éléments d'économie politique pure ou théorie de la richesse sociale (Elements of Pure Economics, or the theory of social wealth)*. Lausanne, Paris, 1874. (1899, 4th ed.; 1926, rev ed., 1954, Engl. transl.).

J. Wang, L. Li, S.H. Low, and J.C. Doyle. Cross-layer optimization in TCP/IP networks. *IEEE/ACM Trans. Networking*, 13:582–268, 2005.

Exercises

5.1 Give a strongly polynomial algorithm for Fisher's linear case under the assumption that all u_{ij}'s are 0/1 (the algorithm given in Section 5.8 is not strongly polynomial).

5.2 Let us extend Fisher's linear model to assume that buyers have utility for money (Vazirani, 2006). Let u_{i0} denote the utility accrued by buyer i for one unit of money. Now, each buyer's optimal bundle can also include money—effectively this is part of their own money which they prefer not to spend at current prices. The notion of equilibrium also generalizes—all goods need to be sold and all money needs to be either spent or returned as part of optimal bundles. Extend the algorithm given in Section 5.8 to this situation, still maintaining its polynomial running time.

5.3 Let us define a new class of utility functions, *spending constraint utility functions* for Fisher's model (Vazirani, 2006). As before, let A and B be the set of goods and buyers, respectively. For $i \in B$ and $j \in A$, let $r^i_j : [0, e(i)] \to \mathbf{R}_+$ be the *rate function* of buyer i for good j; it specifies the rate at which i derives utility per unit of j received, as a function of the amount of her budget spent on j. If the price of j is fixed at p_j per unit amount of j, then the function r^i_j/p_j gives the rate at which i derives utility per dollar spent, as a function of the amount of her budget spent on j.

Relative to prices $\boldsymbol{p}$ for the goods, give efficient algorithms for

(a) computing buyer i's optimal bundle,
(b) determining if $\boldsymbol{p}$ are equilibrium prices, and
(c) computing equilibrium allocations if $\boldsymbol{p}$ are equilibrium prices.

5.4 Prove that equilibrium prices are unique for the model of Exercise 5.3.

5.5 It turns out that there is a polynomial time algorithm for computing equilibrium prices and allocations for the utility functions defined in Exercise 5.3 (Devanur and Vazirani, 2004; Vazirani, 2006). The following is an attempt to use this algorithm to derive an algorithm for computing equilibrium prices for the case of piecewise-linear, concave utility functions for Fisher's model.

Let f_{ij} be the piecewise-linear, concave utility function of buyer i for good j; f_{ij} is a function of x_{ij}, the allocation of good j to buyer i. Let $\boldsymbol{p}$ be any prices of goods that sum up to the total money possessed by buyers (as before, we will assume that there is a unit amount of each good in the market).

Let us obtain spending constraint utility functions from the f_{ij}'s as follows. Let g_{ij} be the derivative of f_{ij}; clearly, g_{ij} is a decreasing step function. Define

$$h_{ij}(y_{ij}) = g\left(\frac{y_{ij}}{p_{ij}}\right),$$

where y_{ij} denotes the amount of money spent by i on good j. Observe that function h_{ij} gives the rate at which i derives utility per unit of j received as a function of the amount of money spent on j. Hence h_{ij} is precisely a spending constraint utility function. Let us run the algorithm mentioned above on these functions h_{ij}'s to obtain equilibrium prices, say $\boldsymbol{p}'$.

Show that $\boldsymbol{p} = \boldsymbol{p}'$ iff prices $\boldsymbol{p}$ are equilibrium prices for the piecewise-linear, concave utility functions f_{ij}'s (equilibrium prices for piecewise-linear, concave utility functions need not be unique).

5.6 **Open problem** (Devanur and Vazirani, 2004): Consider the process given in Exercise 5.3, which, given starting prices $\boldsymbol{p}$, finds new prices $\boldsymbol{p}'$. By the assertion made in Exercise 5.3, the fixed points of this process are precisely equilibrium prices for the piecewise-linear, concave utility functions f_{ij}'s.

Does this procedure converge to a fixed point, and if so, how fast? If it does not converge fast enough, does it converge quickly to an approximate fixed point, which may be used to obtain approximate equilibrium prices?

5.7 Consider the single-source multiple-sink market for which a strongly polynomial algorithm is given in Section 5.14. Obtain simpler algorithms for the case that the underlying graph is a path or a tree.

5.8 Observe that the algorithm given in Section 5.14 for Market 1 defined in Section 5.13 uses the max-flow min-cut theorem critically (Jain and Vazirani, 2006). Obtain a strongly polynomial algorithm for Market 3 using the following max–min theorem.

For a partition $V_1, \ldots, V_k$, $k \geq 2$ of the vertices of an undirected graph G, let C be the capacity of edges whose end points are in different parts. Let us define the edge-tenacity of this partition to be $C/(k-1)$, and let us define the *edge-tenacity* of G to be the minimum edge-tenacity over all partitions. Nash-William (1961) and Tutte (1961) proved that the maximum fractional packing of spanning trees in G is exactly equal to its edge-tenacity.

5.9 Next consider Market 2 defined in Section 5.13. For the case $|A| = 1$, a polynomial time algorithm follows from the following max–min theorem due to Edmonds (1967).

Let $G = (V, E)$ be a directed graph with edge capacities specified and source $s \in V$. The maximum number of branchings rooted out of s that can be packed in G equals $\min_{v \in V} c(v)$, where $c(v)$ is the capacity of a minimum $s - v$ cut.

Next assume that there are two agents, $s_1, s_2 \in V$. Derive a strongly polynomial algorithm for this market using the following fact from Jain and Vazirani (2006). Let F_1 and F_2 be capacities of a minimum $s_1 - s_2$ and $s_2 - s_1$ cut, respectively. Let F be $\min_{v \in V - \{s_1, s_2\}} f'(v)$, where $f'(v)$ is the capacity of a minimum cut separating v from s_1 and s_2. Then:

(a) The maximum number of branchings, rooted at s_1 and s_2, that can be packed in G is exactly $\min\{F_1 + F_2, F\}$.

(b) Let f_1 and f_2 be two nonnegative real numbers such that $f_1 \leq F_1$, $f_2 \leq F_2$, and $f_1 + f_2 \leq F$. Then there exists a packing of branchings in G with f_1 of them rooted at s_1 and f_2 of them rooted at s_2.