Design of a Combinatorial Algorithm for Computing Arrow-Debreau Market Equilibria

1 Motivation

In this project, we considered the problem of finding an algorithm for computing the market equillibrium in an Arrow- Debreu market.The market equillibrium problem consists of finding a set of prices and allocations of goods to economic agents such that each agent maximises his utility, subject to his budget constraints, and the market clears. Kamal Jain, in a paper in 2004, provided a first polynomial time exact algorithm for computing an Arrow-Debreu market equillibrium for the case of linear utilities. But his algorithm was based on solving a convex program using the ellipsoid algorithm and simultaneous diophantine equations. To date, no polynomial time combinatorial algorithm for the same is known. Thus, we sought to solve the open question of designing an efficient combinatorial algorithm for the A-D market equillibrium problem, and gain further structural insights about the theory in the process.

2 The Model

There are n people in the system. Each has an initial endowment from a set of m goods. Also, each person has a utility for these goods, and the total utility of person i for the goods he gets is $\sum_{j=1}^{m} u_{ij}x_{ij}$ where x_{ij} is the final allocation of goods of person i. We now need to find a set of prices p_j for the m goods such that when everyone sells their initial goods and with the money gotten buys his optimal set of goods, the market clears. Arrow-Debreau's theorem guarantees that such a set of prices exist. We attempt to construct it using a combinatorial algorithm.

Now, in the case of the utility functions being linear, as described above, we can assume that without loss of generality, the total amount of each good is 1. i.e. $\sum_{i=1}^{n} x_{ij} = 1$ for all goods j.

Further, we may assume that each person is endowed with exactly one good initially, and that his good is distinct from the others'. We can do this by "splitting" a person having multiple goods into multiple people with identical utility functions, and by treating 2 people who have the same good as if that was a different good for which everyone had the same utility for.

Finally, this gives us a simplified model where we have n people, each of them owning a single unit of a distinct good (i.e. the ith person owns the ith good), and we have to obtain prices for these n goods.

We also require the notion of a *non-zero liking graph*. This is a directed graph having n nodes, each node representing a person/good, where there is a directed edge from node i to node j if the ith person is interested in the jth good. i.e. edge ij exists iff $u_{ij} > 0$. With a little loss of generality, which is justified below, we assume that the non-zero liking graph consists of a single connected component. This assumption ensures that the prices of all the goods must be positive.

In order to justify the above assumption, note that if a price vector $\underline{p}$ is a solution, then any scalar multiple of it $\lambda\underline{p}$ will also be a solution. Hence, if we have solutions for each SCC of the non-zero liking graph, we could arbitrarily scale the solutions in each individual SCC, and they would still remain solutions of their corresponding SCCs. We do this scaling by ensuring that in the DAG decomposition of the SCCs, the prices of the lower component is scaled up by such a large margin that no one from the upper component would be interested in buying their goods. This would then give us a solution for the overall set of goods.

3 Non-convex Program

The following non-convex program as proposed by Kamal Jain, has all and only market equilibria as feasibile points

$$\forall j : \sum_{i=1}^{n} x_{ij} = 1$$

$$\forall i, j : x_{ij} \geq 0$$

$$\forall i, j : u_{ij}/p_j \leq (\sum_{k} u_{ik}x_{ik})/p_i$$

$$\forall i : p_i > 0$$

Note that in the above non-convex program, the third equation is meaningful only where $u_{ij} > 0$, which is precisely along the edges of the non-zero liking graph. Further, if we were to slightly modify the equation to get $\forall i, j$ where $u_{ij} > 0 : p_i/p_j \leq \sum_k u_{ik}x_{ik}/u_{ij}$. Which means that the product of the RHS values along any cycle should be ≥ 1 in order for the price vector to

even exist. Further, Farkas' lemma ensures that such a price vector would exist given this.

Hence, to our non-zero liking graph, we assign weights $w_{ij} = \sum_k u_{ik}x_{ik}/u_{ij}$. We now require that given the first two equations of the non-convex program, we also require that the product over all cycles of the non-zero liking graph of $w_{ij} \geq 1$. Or, if we were to assign weights of value $log(w_{ij})$, then we would require that this graph have no negative-weight cycles. We further work on trying to ensure that this graph has no negative weight cycles combinatorially.

4 Detection of negative cycles

Since we need to ensure that all cycles are non-negative, we try to iteratively increase the weight of the most negative cycle, while ensuring that no other cycle becomes more negative. If we can do this repeatedly, then eventually we would be able to get all cycles non-negative.

In order to find the most negative cycle, we try to find to the most negative cycle for each node of the graph. Note that in w_{ij}, we have control only over the values of x_{ik}, hence in trying to increase the weight of this particular cycle through a node i, we would be increasing the weights of all the edges outgoing from node i. Similarly, the decrease would be reflected throughout.

The value of the most negative cycle containing a node u can be found by performing a run of the Floyd-Warshall algorithm using $log(w_{ij})$ as weights of the edges. While Floyd-Warshall can be used to find the value of the most negative cycle, and also can be modified to find one such cycle that attains this value, it would not be possible to ensure that we find *all* the cycles containing a vertex having the most negative value.

Hence, now if the most negative cycle of the graph contains a node i, and there exists a node i' for which its most negative cycle has weight strictly larger than the most negative cycle (which contains node i), then we may transfer some amount of good j, where $x_{i'j} > 0$, and $u_{ij} > 0$ from user i to user i' thus increasing the weight of the cycle containing node i.

Does this system of nodes i, i' and good j always exist? Unfortunately, this is not the case. In fact, for a simple case of the non-zero liking graph containing just a single cycle, and if at some point of the algorithm users are allocated goods they would rather not have, then this cycle would be negative and clearly, there would not be any other less-negative cycle in the

graph. In such cases, we would need to transfer goods between users within the cycle such the overall effect is an increase in the weight of the cycle.

At this point, the analysis of the graph became too complex since there could be interleavings between nodes in a single most-negative cycle with other nearly-most-negative cycles, or even multiple most-negative-cycles which we would have to correct one by one.

5 Conclusion

Thus,we do have a combinatorial like characterisation for the A-D market equillibria i.e the no-negative cycle in the non-zero liking graph. This characterisation is similar to the no-negative cycle theorem in the min cost flow problem, and thus gives us promising hope for the existence of a combinatorial algorithm. But contrary to what we have in the mincost flow problem,we dont know how to fix a negative cycle if we find one. Finding an active combinatorial characterisation that tells us how to move from a state of non-eqillibrium to equillibrium in polynomial time will give us deep insights about the nature of the market equillibria in general.

6 References

1. K. Jain. A Polynomial Time Algorithm for Computing the Arrow-Debreu Market Equilibrium for Linear Utilities [2004].