Lecture 1

Complexity of Computing Equilibria

April 20, 2005
Lecturer: Kunal Talwar
Notes: Ethan Katz-Bassett

1.1 Overview

In earlier lectures, we have developed an understanding of the meaning of Nash equilibria. Today, we look at the complexity of finding Nash equilibria for certain classes of games. A pure equilibrium is a much clearer concept than mixed; it is much easier to justify that a pure equilbrium is what players will do in practice.

We look at games in which these pure equilibria are guaranteed to exist. The complexity in these cases lies in between P and NP. In a 2 player game, a pure Nash equilibrium is simple. The possible strategies define a matrix, and we can just check the cells in the matrix to identify pure equilibria– n^2 cells if each player has n strategies. So we look at multiplayer games with, say, n players. Even with only 2 strategies per player, there would be 2^n cells. So, we look at implicit representations.

1.2 Congestion Games

Informally, a congestion game involves players choosing from a set of resources, with the value of a resource descreasing as more players use it. For example, consider bees and flowers. A bee wants to choose a flower with a lot of nectar, but the flower is not worth as much to the bee if many other bees also choose it. As another example, consider choosing which bridge to take to the East Side. The utility of choosing 520 decreases as more drivers also choose it.

Now, let's consider a more formal example (see Figure 1.1). We have a directed graph and players $\{A, B, C\}$. All want to enter the graph s and leave at t.

Each edge has an increasing delay function, depending on the number of users (the triples on the edges). Under the paths indicated by the dotted lines (see Figure 1.1), A, C experience a delay of 7, and B experiences a delay of 8. This state is a Nash equilibrium, as no one has an incentive to deviate. However, it is not the best social welfare: if C switches to take the route along the bottom 2 edges, the delays will be $A = 4, B = 7, C = 9$.

We can formally define the family of congestion games:

Definition 1.1. A *congestion game* consists of:

- A set E of resources

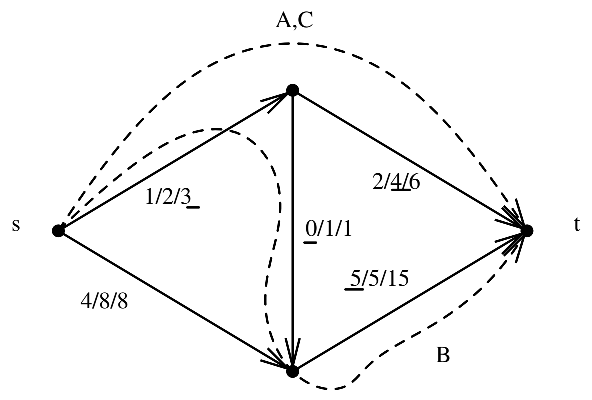

Figure 1.1: Network Congestion Game

- n players
- For each player i, a strategy set $S_i \subseteq 2^E$
- For each $e \in E$, $d_e : \mathbb{N} \to \mathbb{Z}_0$, an increasing delay function for resource e that maps the number of users of e to the delay

A strategy profile consists of one strategy per player: $S = (s_1, s_2, \ldots, s_n), s_i \in S_i$. We define a function $n_e(S)$ to give the number of players using resource e under strategy profile S: $n_e(S) = |\{i : e \in s_i\}|$. A player's utility for a strategy profile is the negation of the delay the player experiences: $-u_i(S) = d_i(S) = \sum_{e \in s_i} d_e(n_e(S))$.

The network congestion game example above is a special case of a congestion game.

1.2.1 Nash Equilibria in Congestion Games

Theorem 1.1. Rosenthal '73 [1] *Every congestion game has a pure Nash Equilibrium.*

We call a strategy profile a state. Now, start from some arbitrary state. If someone has a better move versus the rest of the strategy profile, then that player has an incentive to deviate. Allowing that player to deviate gives us a new state. Letting the states be nodes in a graph, we draw an edge with label i from state s^1 to state s^2 if player i has incentive to deviate from s^1 and s^2 is the state resulting from this single deviation. This is a huge graph, and every node is a pure strategy profile.

We will show that this graph is a directed acyclic graph and use the fact that every DAG has a sink to prove the game has a pure Nash Equilibrium. We will show it is a DAG by associating a potential with each node and showing that the potential decreases along every edge, so there can be no cycles.

Proof. For each state (strategy profile) s, we define the *potential* $\phi(s)$ by:

$$\phi(s) = \sum_e \sum_{j=1}^{n_e(s)} d_e(j)$$

and

$$\phi_e(s) = \sum_{j=1}^{n_e(s)} d_e(j)$$

We will show that if i wants to change (increase payoff/decrease delay), then $d_i(s) - d_i(s') = \phi(s) - \phi(s')$, where $s'/-i = s/-i$ (the states are the same except for i).

$$d_i(s) - d_i(s') = \sum_e (d_i^e(s) - d_i^e(s')) = \sum_e (\phi_e(s) - \phi_e(s'))$$

Now, we think about the problem as if the players come in some order. Each player experiences a *pseudodelay* on an edge e of just the players who traverse e and come before (and including) that player in the order. Clearly,

$$\phi_e(s) = \sum_j \text{pseudodelay}_e(j) = \sum_j d_e(n_e^{\leq j}(s))$$

where $n_e^{\leq j}(s)$ is the number of people up to j who use e. Then

$$\phi(s) = \sum_j \sum_{e \in s_j} d_e(n_e^{\leq j}(s))$$

The sum of the pseudodelay values does not depend on the order of players, so we look at the order where i (the deviator) comes last. No other player's pseudodelay changes, since they do not "see" i on edges anyway. The last player's pseudodelay always equals the actual delay. The sum of the other players' pseudodelays then is some constant c, and

$$\phi(s) = c + \sum_{e \in s_i} d_e(n_e(s))$$

and

$$\phi(s') = c + \sum_{e \in s'_i} d_e(n_e(s'))$$

meaning

$$d_i(s) - d_i(s') = \phi(s) - \phi(s')$$

So we start at some value of potential and decrease by an integer when a player deviates, and the Nash dynamics converges. □

Basically, we are looking for a local optimum of the potential function. Can we do this in polytime?

1.2.2 PLS - Polynomial-time Local Search

We introduce the complexity class PLS (defined in [2]). PLS lies somewhere between P and NP. A general *local search* problem consists of the following:

- a set I of instances
- for each instance $x \in I$, a set F_x of feasible solutions
- an oracle $c(x, s)$, which checks if $s \in F_x$ and if so gives an integer cost of that feasible solution

- for any s, there is some neighborhood $N_x(s) \subseteq F_x$
- a function $g(x,s)$ that finds a better solution: it either says that s is locally optimal (in its neighborhood) or returns $s' \in N_x(s)$ such that $c(x,s') < c(x,s)$
- a subroutine $h(x)$ that returns some $s \in F_x$, giving a starting point

A local search problem is in PLS if c, g, h are all polynomial time algorithms.

In a PLS problem, we are given x and must find a local optimum, or we are given x and some starting point and must find a reachable optimum.

There is a $1+\epsilon$ polytime approximation scheme for a PLS problem. We only make the transition from s to $g(x,s) = s'$ if the cost decreases by more than ϵ. However, this will approximate the minimum of the potential function, but not necessarily the Nash equilibrium. Using our congestion game notation, we want $\Delta d_i(s) \leq \epsilon d_i(s)$, but we are only guaranteed $\Delta\phi(s) \leq \epsilon\phi(s)$ (by our Δ values, we mean the amount the function would change given any state transition). We want to say that no player has more than an ϵ incentive to deviate, but the potential could be much more than an individual player's delay.

1.2.3 The Congestion Game Problem is PLS-complete

For any congestion game x and corresponding starting point, there is some local optimum because all DAGs have sinks. We will show that the congestion game problem is in fact PLS-complete. We do this by reducing from POS NAE 3SAT-FLIP, which is known to be PLS-complete. POS NAE 3SAT/FLIP is POSITIVE WEIGHTED NOT ALL EQUAL 3SAT/FLIP. In an instance of this problem, we are given a 3SAT instance with no negations, and each clause c has a weight w_c. We want to find an assignment with not all literals in a clause the same (so the clause and its complement will both be satisfied) such that a single flip of a variable will not increase the weight of satisfied clauses (so the neighborhood of a feasible solution is all solutions that can be obtained by flipping the value of one variable). We seek to minimize the weight of unsatisfied clauses.

First, we show our congestion game construction. Given an instance of POSE NAE 3SAT/FLIP, we construct a congestion game as follows. We have a set of resource $E = \{c, c' : c \text{ is a clause}\}$, giving us 2 resources per clause. Our players are our variables. Each player x has 2 strategies $S_x = \{\{c : x \in c\}, \{c' : x \in c\}\}$, the first corresponding to **true** and the second to **false**. We define the delay on a resource by $d_{c'} = d_c = 0/0/w_c$. So, if at most 2 players pass through a clause, the delay is 0. Any clause not satisfied contributes $3w_C$ to the total delay. A Nash equilibrium of this congestion game corresponse to a local optimum for the POS NAE 3SAT-FLIP problem. This is a reduction to an asymmetric generalized conegestion game, meaning that the players do not necessarily all have the same strategies.

We now show that the class of symmetric congestion games (same strategies for all players) is also PLS-complete by showing how to do an equivalent reduction to the symmetric case. In this case, our resources are $E \cup \{x : x \text{ is a variable}\}$, where E is from above. The common strategy set $S = \bigcup_x S_x$, where $S_x = \{x \cup \{c : x \in c\}, x \cup \{c' : x \in c\}\}$. The delays are $d_{c'} = d_c = 0/0/w_c/w_c/\ldots$ and $d_x = 0/M/M/\ldots$, where M is a very large number. Clearly, any equilibrium of this game with have exactly one player choosing a strategy from S_x (to avoid the cost M), corresponding to an equilibrium in the asymetric game.

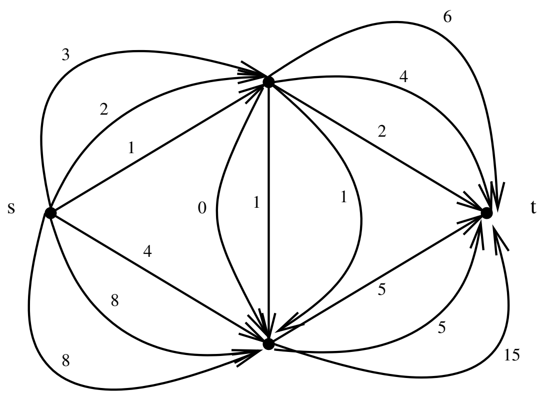

Figure 1.2: Triplicated Edges for Min Cost Flow

1.2.4 Relationships between Different Classes of Congestion Games

We have shown that asymetric congestion games and symmetric congestion games are PLS-complete. It is harder to show (and will not be shown here), but asymmetric network games are also PLS-complete. We now look at symmetric network games.

1.2.5 The Symmetric Network Game Problem is in P

We show that we can find pure Nash equilibiria in poly-time in symmetric network games, those in which all players have the same origin and destination. We construct a new graph in order to reduce to the min-cost flow problem. In this new graph, all edges have capacity 1, and we want to minimize total cost given some flow. We triplicate each edge in the original graph, with the costs of a triple increasing as defined above (see Figure 1.2). The cost of any particular flow is exactly the potential we defined. We know how to solve min-cost flow, and the solution is always integral. This solution gives us a global optimum, so is of course a local optimum.

If we allow split flow, an approach like we just described gives us an approximate solution.

1.3 General Potential Games

No games other than congestion games are potential games (meaning those in which a defection by a player improves the player's payoff by exactly the change in the potential function). We now look at a slightly different type of game.

In this game we have:

- N players
- w_{ij} a symmetric integer warmth function defined for pairs of players i, j
- Each player i, picks either red or blue; say,$s_i \in \{+1, -1\}$

- A strategy profile $s = (s_1, \dots, s_n)$ of all the players' moves/colors
- For each player i, a utility $u_i(S) = \text{sgn}(\sum_j s_i \cdot s_j \cdot w_{ij})$, so a player does well if most friends are on the same team and most enemies on the other
- A potential $\phi(s) = \sum_{i,j} s_i \cdot s_j \cdot w_{ij}$

The potential increases whenever someone defects, but the amount it changes by is not the same as the amount the player utilities change by. So it is not a potential game like we saw above, but rather a *general potential game* (meaning one where the sign changes are the same), which is all we really need for local search. This class is exactly PLS.

1.4 Example of games with pure Nash not in the classes above

1.4.1 Player-specific edge weights

We have:

- n players
- m parallel edges (strategies)
- for each edge e, a delay function $d_e : 2^n \rightarrow \mathbb{R}$, a nondecreasing function of the specific set of players choosing e (not the number)

So every player that chooses a particular edge gets the same delay, but the value of the delay depends on who is using it.

Proof. Consider a state s and the multiset of numbers $\{d_1(s), d_2(s), \dots, d_m(s)\}$, the delays of each edge. Whenever a player defects from an edge, this multiset decreases lexigraphically. This is true because every player on an edge experiences the same delay. When a player moves, the delay of other players on the edge the player moved from decreases, and the delay of players on the edge moved to increases. However, the player would only switch to improve, so the delay decreased from is more than the delay increased to; this change gives the lexigraphic decrease. □

1.4.2 Caching Game

In this game, there are n players deciding whether to cache a file or get a copy from the nearest server. A player x can either pay M to cache or $d(x, y)$ to get a copy from server y. As M decreases from infinity, eventually you will start caching. Others will get the file from you. Once you start caching, you will never stop (as M decreases further). Suppose, at some later point, it is cheaper to get a copy from y (who starts caching after you) and pay $d(x, y) < M$. Then y is being stupid and should have instead got the file from you and paid $d(x, y)$. So if we bring M down to a target value, we have a Pure Nash equilibrium.

References

[1] R.W. Rosenthal. A class of games possessing pure-strategy nash equilibria. *International Journal of Game Theory*, 2:65-67, July 2002.

[2] D.S. Johnson, C.H. Papadimitriou, and M. Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37:79-100, 1988.