600.363 Introduction to Algorithms / 600.463 Algorithms I **Lecturer:** Michael Dinitz
Topic: Algorithms and Game Theory **Date:** 12/4/14

25.1 Introduction

Today we're going to spend some time discussing game theory and algorithms. There are a lot of different ways and places that algorithms and game theory intersect – we're only going to discuss a few of them. There is actually an incredibly active research area in theoretical CS called "algorithmic game theory", but we're only going to scratch the surface and discuss some of the more classical results.

25.2 Two-player zero-sum

One of the oldest settings in game theory are two-player zero-sum games. We briefly discussed these when we talked about linear programming, but let's go into a bit more detail. Remember our example from there: we considered penalty kicks from soccer. There was a matrix of payoffs, where the shooter was the row player and the goalie was the column player:

	Left	Right
Left	(0,0)	(1,-1)
Right	(1, -1)	(0,0)

In other words, if the shooter kicks it left/right and the goalie dives in the opposite direction then there's a goal: the shooter gets 1 and the goalie gets -1. If they go in the same direction, though, there is no goal and no player gets any payoff.

In general, there is a matrix M with n rows and m columns, and each entry is of the form $(x, -x)$. Such an entry at position (i, j) means that if the row player chooses action i and the column player chooses action j then the row player receives payoff x and the column player receives payoff $-x$. Saying that the game is "zero-sum" just means that in every row the sum of the two payoffs is 0 (note that x could be negative, so the column player isn't necessarily restricted to negative payoffs, but we'll usually think of x as being positive).

Given such a game, what should a player (either row or column) do? A natural thing to try for is a (randomized) strategy that maximizes the expected payoff even when the opposing player does the best they can against it. In other words, we could like to maximize (over the choice of all of our randomized strategies) the minimum (over all possible strategies for our opponent) of our expected payoff. Intuitively, this is the strategy we should play if our opponent knows us well – if we play anything else then they will have some opposing strategy where we do worse. Such a strategy is called a *minimax* strategy. We saw how to calculate this strategy using an LP in the class on linear programming.

So what about the above example? It's pretty clear that both players actually have the same minimax strategy: choose between left and right with probability $1/2$ each. Then the shooter has expected payoff of $1/2$ and the goalie has expected payoff of $-1/2$.

Recall that our example from the LP class involved a goalie who was weaker on the left. We considered the following game.

	Left	Right
Left	$(\frac{1}{2}, -\frac{1}{2})$	(1,-1)
Right	(1, -1)	(0,0)

In this game the minimax strategy for the shooter is $(2/3, 1/3)$, which guarantees expected gain of at least $2/3$ no matter what the goalie does. The minimax strategy for the goalie is also $(2/3, 1/3)$, which guarantees expected loss of at most $2/3$ no matter what the shooter does.

Theorem 25.2.1 (Minimax Theorem (von Neumann)) *Every 2-player zero-sum game has a unique value V such that the minimax strategy for the row player guarantees expected gain of at least V, and the minimal strategy for the column player also guarantees expected loss of at most V.*

This theorem is somewhat counterintuitive. For example, it implies that if both players are optimal, it doesn't hurt to just publish your strategy. Your opponent can't take advantage of it.

We've already seen how to use LPs to calculate the value of a game, but two-player zero-sum games can also provide a useful framework for thinking *about* algorithms, and in particular for thinking about bounds on algorithms. Think of a game in which each row is a different algorithm for some problem and each column is a different possible input, and the entry at (i, j) is the cost of algorithm i on input j. For example, consider sorting. Then each row is a different sorting algorithm and each column is a different input (i.e., permutation of $\{1, 2, \ldots, n\}$), and each entry is the time the algorithm takes to sort the input. Of course, this matrix is massive and we can't actually write it out, but we can think about it.

As algorithm designers, what we're trying to do is find a good row in this matrix. Since we're looking at worst-case guarantees, by "good" we mean a row which minimizes the maximum cost in that row. Essentially a minimax strategy! But more interestingly, suppose that we're trying to prove *lower bounds* on algorithms, like e.g. the sorting lower bound. Then we are essentially trying to prove that in every row i, there is at least one column j so that the (i, j) entry is bad. This is exactly what the sorting lower bound is – for any possible sorting algorithm (decision tree), we found an input on which it performs poorly.

But now let's make our lives more difficult – suppose we want a lower bound on *randomized* algorithms? Now we can think of each entry of this huge matrix as not just an algorithm, but an algorithm paired with a particular instantiation of randomness. For example, there would be a row for each possible order of pivot selection in randomized quicksort, so there would be $n!$ rows for randomized quicksort. Now it's easy to see that a randomized algorithm is just a distribution over these rows, and the worst-case expected cost of the randomized algorithm is the maximum over all columns of the expectation of the cost to the row player when using this strategy, assuming the column player plays the designated column.

But this is exactly the value of the game! So we can use the minimax theorem to say that this value is exactly equal to the minimax value of the column player. In other words, the value of the best randomized algorithm (worst-case over inputs) is equal to the value of the best *deterministic* algorithm over a worst-case *distribution* over inputs. So instead of analyzing a randomized algorithm over a worst-case input, we can analyze a deterministic algorithm against a worst-case *distribution* over inputs. This is known as *Yao's minimax principle*, due to Andrew Yao, and is the standard way of proving lower bounds on randomized algorithms.

25.3 General games and Nash equilibria

In general games we can remove both of the restriction: we allow more than 2 players (although 2 is often a useful number to consider), and payoffs don't need to add to 0. Among other things, this means that games no longer have a unique value. And instead of minimax strategies, we have the notion of a *Nash equilibrium.* Informally, a Nash equilibrium is a strategy for each player (these might be randomized strategies) such that no player has any incentive to deviate. Let's do a simple example: two people walking down the sidewalk, deciding which side to walk on.

	Left	Right
Left	(1,1)	(-1,-1)
Right	(-1, -1)	(1,1)

This game has three Nash equilibria: both people walk on the left, both people walk on the right, or both people decide random with probabilities $(1/2, 1/2)$. Note that the first two equilibria give both players payoff of 1, while the third equilibrium gives both players expected payoff of 0. Nevertheless, it is an equilibrium: if that's what both players are doing, then neither player has any incentive to change their strategy.

25.3.1 Computing Nash

Nash proved that if the number of players is finite and the number of possible actions for each player is finite, then there is always at least 1 Nash equilibrium. However, his proof is nonconstructive – it doesn't give an algorithm for actually finding an equilibrium. This hasn't seemed to bother economists and mathematicians, but it should bother us! After all, the whole point of an equilibrium from an economics point of view is that it's a good solution concept – the claim is that markets/systems will naturally end up at equilibrium. But if it's hard to compute an equilibrium, then how can we possibly expect a massive distributed system like a market to end up at equilibrium?

So we have the following algorithmic question: given a game, can we compute a Nash equilibrium? This is actually a bit hard to formalize. Since an equilibrium always exists the decision question "does this game have an equilibrium" is certainly not NP-complete. And since equilibria don't come with a notion of "value" like minimax, we can't create a less trivial problem like "does this game have an equilibrium with large value". Instead, we have to use a different complexity class: PPAD. Without going into any details, PPAD is a class of problems in which the answer is always "yes", but where (we believe) it is hard to actually find a solution in general.

Theorem 25.3.1 (Daskalakis, Goldberg, Papadimitriou) *Computing a Nash equilibrium is PPAD-complete.*

From a computational point of view, this means that Nash equilibria are in fact not good solution concepts in general, since there's no real reason assume that a game will end up at such an equilibrium. Of course, for specific games/markets we might be able to prove that it is easy to compute a Nash equilibrium, or even that natural distributed algorithms (e.g., best-response dynamics) converge quickly to an equilibrium. But in general we don't believe that this is the case. Among other things, this has motivated other notions of equilibrium which are actually computable, such as correlated and coarse-correlated equilibria (which were defined previously by economists but were only recently shown to have natural distributed algorithms).

25.3.2 Braess's Paradox

Nash equilibria can behave somewhat strangely and counterintuitively. The most famous example of this is Braess's Paradox, which comes up in routing games. In a routing games (at least the simplest versions), we are given a graph, together with a source and destination. We assume there are a huge number of players(say $1/\epsilon$), each of which is trying to get from the source to the destination as quickly as possible. So each player is responsible for ϵ traffic, and its actions are the possible paths from the source to the destination. However, the length of an edge might be some function of the total traffic along the edge, rather than just a number.

Let's do an example. Consider the following graph:

The only Nash equilibrium is for half of the players to choose the top path and half to choose the bottom path. If more than half choose the top or more than half choose the bottom, then they could obtain shorter travel time by switching. So at equilibrium, every player has travel time equal to 3/2.

But now suppose that the government decides to invest in a new road. This is such a great road that it takes almost no time to travel across it, no matter how many players use it. We might think that this can only make things better – clearly if the road is between the source of the destination then it's a huge help (travel time gets cut to 0), and if its between any two other points then it still provides a ton of extra capacity. Unfortunately, this is not the case. If we place it between the two non-source/destination nodes, then in fact the only Nash equilibrium is where all players use the new zig-zag path, giving a travel time of 2 for each player!

This effect is called Braess's Paradox. Note that it is entirely due to game-theoretic behavior: if we could tell every player what to do then we could force them all to simply ignore the new road. But since players are selfish, adding this fancy new road actually decreased the quality of the system. Nowadays, people use the term "Braess's Paradox" to mean such a situation, even outside of routing games. For example, I recently wrote a paper showing that in wireless networks, improving technology (e.g. improving the signal-to-noise ratio that we can decode, or allowing nodes to choose their broadcast power, or allowing fancy decoding techniques like interference cancellation) can actually result in worse Nash equilibria, and thus worse actual performance.

25.3.3 Price of Anarchy

Braess's paradox has been known for a while, but (like with the computational issues involving Nash) it didn't seem to bother economists too much. I'm not quite sure why this is, but maybe if you assume that games/markets are "natural" then there's not much point in comparing them to centralized solutions. But comparing to optimal solutions is exactly what we do all the time in theoretical CS! We did this, for example, with approximation and online algorithms.

This motivates the definition of the *price of anarchy*, which was introduced by Koutsoupias and Papadimitriou in 1999. For a given game, let OPT denote the "value" of the best solution, which is typically (although not always) the social welfare, i.e. the sum over all players of the value obtained by the player. So, for example, in the routing game we looked at above $OPT = 3/2$. For a fixed equilibrium s, let $W(s)$ denote the "value" of the equilibrium, which again might be defined differently for different games but (for example) in the routing game is equal to the average trip length when players use equilibrium s. So before the new road there was only one possible s and $W(s) = 3/2$, and after the new road there is still only one possible s but it is a different equilibrium and now $W(s) = 2$. Let $\mathcal{S}$ denote the set of all equilibria.

Definition 25.3.2 *The* price of anarchy *of a minimization game is* $\max_{s \in \mathcal{S}} W(s)/OPT$*, and the PoA of a maximization game is* $\min_{s \in \mathcal{S}} W(s)/OPT$*.*

In other words, the price of anarchy of a game is the ratio between the *worst* Nash equilibrium and the optimum value. We don't have time to go into it, but analyzing the price of anarchy of various games has been a popular area for the last 10 years or so. We now understand many classes of games quite well. For example, take routing games: it is known (thanks to Tim Roughgarden) that as long as edge lengths are a linear function of the traffic across them (as they were in our example) the Price of Anarchy is always at most 4/3. This is exactly the gap that we observed! So in fact our simple exapmle of Braess's paradox is the worst possible.