601.436/636 Algorithmic Game Theory **Lecturer:** Michael Dinitz
Topic: Intro, What is AGT **Date:** 1/28/20
Scribe: Michael Dinitz

1.1 Introduction

Welcome to Algorithmic Game Theory! A quick overview of the syllabus, and a few notes:

- Who am I: I'm an assistant professor in the theory group, where I focus on the theory of algorithms, particularly approximation algorithms and algorithmic problems arising from computer networking and distributed computing. I do not specialize in algorithmic game theory, but it's an area that I occasionally dabble in (ALGOSENSORS '19, NeurIPS '18, DISC '13, SODA '13, INFOCOM '10, INFOCOM '09, SODA '09). But I think it's a central part of modern theoretical computer science, and a great example of cross-disciplinary work that actually benefits computer science (rather than just the other discipline), so I thought it was important to expose JHU students to it.
- TA: Yasamin Nazari, one of my PhD students. She works mostly on distributed algorithms.
- Prerequisite: Intro to Algorithms. This class will be extremely difficult (if not impossible) if you're not familiar with the content from that class.
- This is designed to be a PhD-level class. I'm excited that others are taking it (particularly all the undergrads!), but it's still designed for PhD students. What does this mean? Among other things, the following.
  - The course will not have as strict a timetable/schedule as an undergrad course. The schedule, including lecture topics, homework due dates, etc., will be played a bit by ear.
  - There won't be a lot of handholding. I will give the high level view and some details, but I'll also assume that you can fill in the blanks. But please come to office hours or get in touch with me if there's something you don't understand!
  - Hopefully this class won't be as much work as my undergraduate class (Intro to Algorithms), since PhD students should mostly be focused on research. But it will definitely move faster.
- Textbook: See course webpage. We'll mostly be using *Algorithmic Game Theory*, a collection of surveys from 2007, but will also be using Tim Roughgarden's *Twenty Lectures on Algorithmic Game Theory*. Both are available online (Tim's through the JHU library). But really, most of the content in this class is from lectures, not the reading.
- Course website: `http://www.cs.jhu.edu/~mdinitz/classes/AGT/Spring2020/`.
- Online discussion group: `http://piazza.com/jhu/spring2020/601436636/home`.

- Grade breakdown: 60% homework, 30% final project, 10% participation.
  - Homeworks: probably around 5. Can work in groups of up to three people, but must write up solutions separately and turn them in separately (on gradescope).
  - Final Project: I'll figure out the exact format of this in a few weeks (once enrollment stabilizes). Ideally, as a PhD-level class, this would be an real research project – I would love it if you could find some aspect of your main research area that has an AGT aspect and could study it. This might not be reasonable for MSE and undergraduate students, though. So there's a good chance that most of you will do something like finding an area of AGT that we didn't have time to talk about and writing up a survey/lecture about it, or finding some modern research papers in AGT and writing a summary/overview, etc. We'll figure this out in a few weeks.
  - Participation: participate in lecture and on piazza, come to office hours, etc.

1.2 AGT Overview

So what is algorithmic game theory? Informally, "game theory" is the study of interaction between (self-interested) agents. This has been studied in enormous depth in economics, and recently has received significant attention in CS, partly due to the many important applications (particularly with the Internet) and partly due to some really interesting phenomena that piqued a lot of intellectual interest. Historically, AGT grew out of theoretical computer scientists examining ideas from Economics and either putting a CS spin on them, or using them in a CS context. In the early days it was very much a subfield of TCS, but nowadays it is its own active area, and has much closer connections to Economics than it used to.

AGT is usually divided into three general areas of study (with significant overlap):

1. Computing Equilibria
2. Inefficiency of Equilibrium
3. Algorithmic Mechanism Design

We will study these areas in this order, but we'll definitely be spending more time on mechanism design and inefficiency of equilibria than we will on computing equilibria.

High Level Descriptions

1. **Computing Equilibria**. "Classical" game theory has been enormously successful at answering existential questions: what are "reasonable" notions of equilibria? When do they exist? But from a CS perspective, existential answers are somewhat unsatisfactory. Even supposing that some notion on equilibrium always exists, we naturally have the following questions:
   (a) Can we (efficiently) compute it?
   (b) Can we (efficiently) compute it distributedly?

(c) If agents use "natural" algorithm, will they get to equilibrium?

These problems are all technically interesting, but more importantly: is it reasonable for us to assume that agents will arrive at equilibrium if we cannot even compute one?? You may have heard the famous phrase about the "invisible hand of the market", and note that this process is by nature computational. So if finding an equilibrium is computationally difficult, are equilibria really a good model of behavior?

2. **Inefficiency of Equilibria**. Suppose that we are at some equilibrium. There are still a bunch of natural questions:

   (a) Is this equilibrium close to "optimal" (the best thing for everyone to do globally if the agents were not self-interested)?

   (b) Are *all* equilibria close to optimal?

   (c) In what classes of games are we guaranteed to be close to optimal?

   These questions can be incredibly important if we're modeling a real system. One of the original motivations behind studying these questions was routing on the Internet. Not to go into any real detail, but as the Internet became decentralized, the incentives of the various agents involved started to matter enormously. So it really started to matter whether the behavior seen from agents pursuing their incentives was actually close to the technical limits of the network. Moreover, studying "distance from optimality" is something that algorithms researchers (particularly approximation algorithms researchers) are very good at, so it was natural for TCS people to push on these questions.

3. **Mechanism Design**. In some ways this is the obvious next step: can we *design* a game so that selfish behavior leads to good outcomes? This has been called "the science of rule-making". Econ researchers have thought about mechanism design for a long time (while inefficiency of equilibria was really pioneered on the CS side), but again, many of the tools and points of view from computer science have led to new insights and ideas. We will mostly focus on some of the most important classical settings, and in particular, auctions: what should the rules of our auction be so that agents are incentivized to tell the truth?

1.3 Detailed Overviews

We're going to start being more formal next class – for now, we'll do some informal examples and overviews.

1.3.1 Game Theory Basics and Computation

A *game* (for now) consists of players, actions, and utilities/costs. Slightly more formally, let P be a set of players, and for each player $i \in P$ we will let A_i denote the possible actions of player i (sometimes called the action set). Then each player will choose one action from its action set, and each player will get a utility from the chosen actions, i.e., each player i will have a utility function $u_i : \bigotimes_{j \in P} A_j \to \mathbb{R}$. We could also have a cost function if players are trying to minimize rather than

maximize (these are equivalent by letting utility be negative cost and vice versa). To somewhat simplify notation, we will let $S = \bigotimes_{j \in P} A_j$ denote the set of all strategy vectors (i.e., the set of all "things that might happen").

Example: Prisoner's Dilemma. Two players (prisoners), each being interrogated simultaneously but in different rooms. Each player has two possible actions: stay silent or confess. If they're both silent, then they'll be in jail for two years. If one confesses and the other stays silent, then the one who confessed only gets one year in prison while the person who was silent gets five years. If they both confess, then they each get four years.

We can represent this succinctly as a matrix, where one player chooses the row and the other player chooses the column. We will use (a, b) to represent the two costs, where a is the cost to the row player and b is the cost to the column player.

	confess	silent
confess	(4,4)	(1,5)
silent	(5,1)	(2, 2)

We haven't defined any notion of equilibrium yet, but using the plain English definition, any notion should incorporate the idea of *stability*, i.e., no one should want to change what they're doing. Using that informal definition, there's only one equilibrium here: both players should confess. This is because if any player does anything else, they'll have incentive to deviate. In other words, for each player, confessing is a *dominant strategy*: no matter what the other player does, I will be better off if I confess. Slightly more formally, in a utility-maximization game, player i has a *dominant strategy* $a \in A_i$ if for all $s \in S$ we have that $u_i(a, s_{-i}) \geq u_i(s)$. This notation is standard (although confusing): s_{-i} denotes s with the i'th coordinate (the action for player i) removed, and $u_i(a, s_{-i})$ denotes s where the i'th coordinate is replaced by a. For a cost-minimization game like the prisoner's dilemma, we can use cost functions with the inequality reversed.

Clearly if every player has a dominant strategy, that is the only reasonable notion of equilibrium. But there are many games that do not have a dominant strategy for every player! For example, consider the "where to eat" game: I prefer burgers and my wife prefers pizza, but neither of us has a super strong preference and we both strongly prefer to eat together rather than separate. Using utilities (rather than costs), we can model this as the following matrix:

	pizza	burger
pizza	(5,6)	(1,1)
burger	(1,1)	(6, 5)

Note that neither player has a dominant strategy here. For example, if the column player plays burger then the row player does best by playing burger, but if the column player plays pizza then the row player is better off playing pizza. On the other hand, there are clearly two equilibria: both playing burger or both playing pizza.

Let's see an even more extreme example: rock-paper-scissors.

	rock	paper	scissors
rock	(0,0)	(-1, 1)	(1, -1)
paper	(1,-1)	(0, 0)	(-1, 1)
scissors	(-1, 1)	(1, -1)	(0,0)

What are the equilibria in this game? In order to define anything, we need to allow *mixed strategies*: players can choose from their actions randomly according to a distribution they set. If we allow such mixed strategies, it's not hard to see that there's an obvious equilibrium. Each player chooses uniformly at random (probability 1/3) from each of their actions. To see that this is an equilibrium, note that for each player, the expected utility is zero (since they each have probability 1/3 of getting utility −1, 0, and 1. Now suppose that player 1 changes its distribution to choose rock with probability p_R, paper with probability p_P, and scissors with probability p_S. Then the expected utility of player 1 is

$$\frac{1}{3}(p_P - p_S) + \frac{1}{3}(p_S - p_R) + \frac{1}{3}(p_R - p_P) = 0$$

So player 1 has no incentive to change. A similar calculation shows that player 2 is the same.

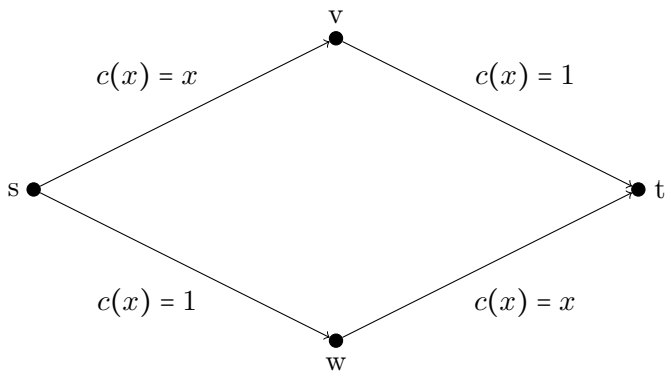
This kind of equilibrium is called a (mixed) Nash equilibrium.

Theorem 1.3.1 (Nash '51) *Every matrix game has a Nash equilibrium.*

But if we put on our computational hats, there's an obvious followup question: can we efficiently compute a Nash equilibrium. It turns out that in *zero-sum* games (like RPS) we can, via linear programming. But in more general matrix games, it is PPAD-hard to compute a Nash equilibrium (don't worry about what PPAD is for now).

1.3.2 Inefficiency of Equilibria

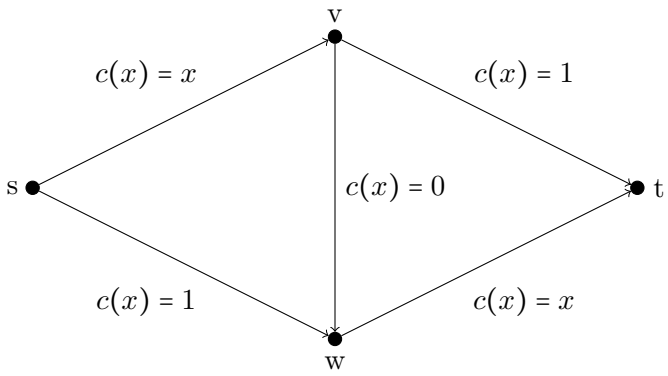
Let's consider a basic routing game. There are many drivers (N, a very large number), each of which is trying to get from s to t in the following graph.



The travel time for an edge might depend on the fraction of the traffic that uses it: if an x fraction of the drivers use an edge, then the travel time for it is the function $c(x)$ for that edge. Every driver is trying to minimize their travel time from s to t. At equilibrium, half of the drivers will

use the top path and half will use the bottom path. This means that everyone will take time 3/2 to get from s to t. Moreover, if we (as the central transportation authority) could tell people how to drive and our goal was to minimize average travel time, then we would still tell people to split 50-50. So in this case, the equilibrium is actually optimal.

Now suppose that we (as the central transportation authority) built a magical teleporter from v to w. This would give the following graph:



Now what happens? If traffic splits equally 50-50 along the top and bottom paths like before, then anyone taking the top path (taking time 3/2) would rather switch to the zig-zag path (taking time 1). In fact, the *only* equilibrium is for *everyone* to use the $s - v - w - t$ path: anyone using either the top or bottom path would be paying $1 + x$, and they could instead be paying $2x < 1 + x$. But this means that with selfish behavior, at equilibrium everyone takes time 2 to get from s to t. But this is clearly suboptimal: if we were a benevolent dictator, we would just tell half the people to use the top path and half to use the bottom path (ignoring the teleporter), for a time of 3/2. So the "cost of selfishness" is

$$\frac{2}{3/2} = 4/3$$

In general, the *Price of Anarchy* is the ratio of selfish behavior to OPT, i.e.,

$$\frac{\text{value of worst Nash}}{\text{value of OPT}}.$$

A related quantity is the *Price of Stability*:

$$\frac{\text{value of best Nash}}{\text{value of OPT}}.$$

Studying these quantities (and related quantities) in various games is the study of inefficiency of equilibria.

1.3.3 Mechanism Design

Consider a simple auction: I'm selling one item, there are n players (bidders), and player i values the item at $v_i \geq 0$. Think of this as a game: each player's actions are bids, and if player i gets the item at price p then i gets utility $v_i - p$ (if i does not get the item then they get utility 0).

Now let's take on the role of the auctioneer: once we're given the bids, how should we decide who to give the item to and at what price?

Obvious Approach: Give the item to the highest bidder and the price they bid. So given bids $b_1, b_2, \ldots b_n$, if $b_i > b_j$ for all $j \neq i \in [n]$ then the utility of player i is $v_i - b_i$ and the utility of all other players is 0 (let's assume WLOG that all bids are unique).

Question: how will/should players bid?

Answer: It's unclear! Clearly player i should never bid more than v_i, but if they got the item with bid b_i and would have gotten it with something slightly less, they should have bid slightly less. In other words, whoever has the top value (let's say player i) should try to bid the smallest they can while still getting the item. But what is this? It depends on other players' valuations (which player i doesn't know), and on whatever strategy they choose (which player i also doesn't know). So there are (at least) two issues: it's hard for players to decide what to do, and it's hard for us (the auctioneer) to figure out what's going to happen.

How can we fix both of these issues? Through something called a *second-price* auction. We'll change the rules of the auction so that we still give the item to the highest bidder, but we'll charge them the *second-highest* bid (rather than their own bid).

Theorem 1.3.2 *For each player i, bidding v_i is a dominant strategy.*

Proof: Suppose that each player $j \neq i$ bids b_j (note that player i does not know these values). How should player i set b_i?

Case 1: Suppose that v_i is less than some other bid b_j. Then bidding v_i results in zero utility for player i since it will not get the item, and any other bid results in either 0 utility (if it does not get the item) or negative utility (if it bids high enough to get it). Thus bidding v_i is the best thing to do.

Case 2: If $v_i > b_j$ for all $j \neq i$. Let i' denote the highest bidder other than i. Then bidding v_i results in utility $v_i - b_{i'} > 0$. Bidding any value larger than $b_{i'}$ results in the same utility. Bidding any value less than $b_{i'}$ results in not getting the item, and thus utility 0. Thus bidding v_i is the best thing to do.

So, no matter what the other bids are, the best thing for player i to do is bid v_i. Thus bidding v_i is a dominant strategy. ■

So the second-price auction is what's known as *truthful* or *incentive-compatible*: bidding their true value is a dominant strategy for all players, so they might as well do so. This means that it is easy for players to figure out what to do, and since they will all tell the truth it is also easy for the auctioneer to know what is happening.

Algorithmic aspects. Second-price auctions have been known to be truthful for a long time, and there's a huge body of work in economics on mechanism design. What's the new CS perspective? We'll explore this more later in the semester, but there are a few obvious things. First, what about optimization? As with equilibria, we want not just truthful mechanisms, but truthful mechanisms that are (near-)optimal. Perhaps more importantly, though: what if the auctioneer or the bidders are computationally constrained (as they would be in real life)? We'll see this in detail when we talk about *combinatorial auctions*, where there are well-known solutions from the econ literature but these solutions require solving NP-hard problems. What can we do in these kinds of cases?