Lecture Notes on Economics, AI, and Optimization
Lecture Note 1: Introduction and Examples

Christian Kroer*

January 19, 2022

1 Why Economics, AI and Optimization?

These lecture notes will examine the topics of game theory and market design. We will go over several application areas for these ideas, where each area will have real-life applications that have been deployed. So why AI and optimization? A common theme underlying all the areas that we will study is that for each area, one or more of the real applications are enabled by AI and optimization. In particular, we will repeatedly see that economic solution concepts often have some underlying convex or mixed-integer formulation of the problem, that allows us to compute solutions. Furthermore, most applications will require scaling at a level where standard optimization methods are not enough. In those settings, AI methods such as abstraction or machine learning are often used. For example, we may have a game that is way too large to even fit in memory. In that case, we can generate some coarse-grained representation of the problem using abstraction or machine learning. This coarse-grained representation is then typically what we solve with optimization methods.

2 Game Theory

The first pillar of the course will be *game theory*. GT is a tool for analyzing the behavior of *rational* agents when

2.1 Nash Equilibrium

One of the most important ideas in game theory is the famous Nash equilibrium. Roughly speaking, a Nash equilibrium of a game is a set of strategies in steady state. What is meant by steady state here is that each player is playing an optimal strategy, given what everybody else is doing. This is perhaps best illustrated with an example. Below is the bimatrix payoffs of the game of rock-paper-scissors (RPS).

	Rock	Paper	Scissors
Rock	0,0	-1,1	1,-1
Paper	1,-1	0,0	-1,1
Scissors	-1,1	1,-1	0,0

*Department of Industrial Engineering and Operations Research, Columbia University. Email: christian.kroer@columbia.edu.

In this representation, Player 1 chooses a row to play, and Player 2 chooses a column to play. Player 1 tries to maximize the first value at the resulting entry in the bimatrix, while Player 2 tries to maximize the second value.

Here is an example of something that is not a Nash equilibrium: Player 1 always plays rock, and Player 2 always plays scissors. In this case, Player 2 is not playing optimally given the strategy of Player 1, since they could improve their payoff from -1 to 1 by switching to deterministically playing paper. In fact, this argument works for any pair of deterministic strategies, and so we see that there is no Nash equilibrium consisting of deterministic strategies.

Instead, RPS is an example of a game where we need randomization in order to arrive at a Nash equilibrium. The idea is that each player gets to choose a probability distribution over their actions instead (e.g. a distribution over rows for Player 1). Now, the value that a given player receives under a pair of mixed strategies is their expected payoff given the randomized strategies. In RPS, it's easy to see that the unique Nash equilibrium is for each player to play each action with probability $\frac{1}{3}$. Given this distribution, there is no other action that either player can switch to and improve their utility. This is what we call a (mixed-strategy) Nash equilibrium.

The famous results of John Nash from 1951 is that *every* game has a Nash equilibrium:

Theorem 1. *Every bimatrix game has a Nash equilibrium.*

The attentive reader may have noticed that the RPS game has a further property: whenever one player wins, the other loses. More generally, a bimatrix game is a zero-sum game if it can be represented in the following form:

$$\min_{x \in \Delta^n} \max_{y \in \Delta^m} \; x^\top A y$$

where Δ^n, Δ^m are the n and m-dimensional probability simplexes, respectively, and A contains the payoff entries to the y-player from the bimatrix representation. This is called a bilinear saddle-point problem. The key here is that we can now represent the game as a single matrix, where the x-player wishes to minimize the bilinear term $x^\top A y$ and the y-player wishes to maximize it. Zero-sum matrix games are very special: they can be solved in polynomial time with a linear program whose size is linear in the matrix size.

A more exciting application of zero-sum games is to use it to compute an optimal strategy for two-player poker (AKA heads-up poker). In fact, as we will discuss later, this was the foundation for many recent" "superhuman AI for poker" results [1, 13, 2, 4]. In order to model poker games we will need a more expressive game class called *extensive-form games* (EFGs). These games are played on trees, where players may sometimes have groups of nodes, called *information sets*, that they cannot distinguish among. An example is shown in Figure 1.

EFGs can also be represented as a bilinear saddle-point problem:

$$\min_{x \in X} \max_{y \in Y} \; x^\top A y,$$

where X, Y are no longer probability simplexes, but more general convex polytopes that encode the sequential decisions spaces of each player. This is called the *sequence-form* representation [17], and we will cover that later. Like matrix games, zero-sum EFGs can be solved in polynomial time with linear programming, and the LP has size linear in the game tree.

It turns out that in many practical scenarios, the LP for solving a zero-sum game ends up being far too large to solve. This is especially true for EFGs, where the game tree can quickly become extremely large if the game has almost any amount of depth. Instead, iterative methods are used in practice. What is meant by iterative methods here is the class of algorithms that build a sequence of strategies $x_0, x_1, \ldots, x_T, y_0, y_1, \ldots, y_T$ where only a constant number of strategies is

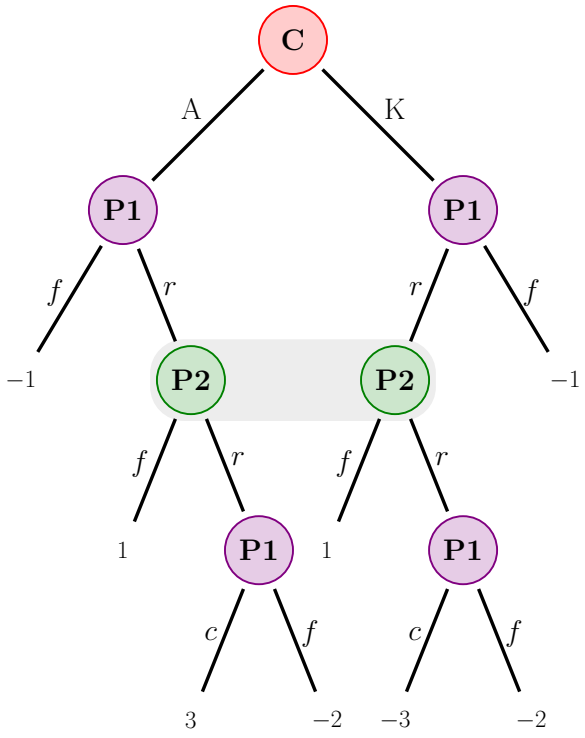

Figure 1: A poker game where P1 is dealt Ace or King. "r," "f," and "c" stands for raise, fold, and check respectively. Leaf values denote P1 payoffs. The shaded area denotes an information set: P2 does not know which of these nodes they are at, and must thus use the same strategy in both.

kept in memory, and only oracle access to Ay and $A^\top x$ is needed (this is different from writing down A explicitly!). Typically the average strategies $\bar{x}_T = \frac{1}{T}\sum_{t=1}^{T} x_t, \bar{y}_T = \frac{1}{T}\sum_{t=1}^{T} y_t$ converge to a Nash equilibrium. The reason these methods are preferred is two-fold, first by never writing down A explicitly we save a lot of memory (now we just need enough memory to store the much smaller x, y strategy vectors), secondly they avoid the expensive matrix inversions involved in the simplex algorithm and interior-point methods.

The algorithmic techniques we will learn in this section are largely centered around iterative methods. First, we will do a quick introduction to online learning and online convex optimization. We will learn about two classes of algorithms: ones that converge to an equilibrium at a rate $O(1/\sqrt{T})$. These roughly correspond to saddle-point variants of gradient-descent-like methods. Then we will learn about methods that converge to the solution at a rate of $O(1/T)$. These roughly correspond to saddle-point variants of accelerated gradient methods. Then we will also look at the practical performance of these algorithms. Here we will see that the following quote is very much true:

> In theory, theory and practice are the same. In practice, they are not.

In particular, the preferred method in practice is the CFR$^+$ algorithm [16] and later variations [3], all of which have a theoretical convergence rate of $O(1/\sqrt{T})$. In contrast, there are methods that converge at a rate of $O(1/T)$ [11**?** , 12] in theory, but these methods are actually slower than CFR$^+$ for most real games!

Being able to compute an approximate Nash equilibrium with iterative methods is only one part of how superhuman AIs were created for poker. In addition, abstraction and deep learning

methods were used to create a small enough game that can be solved with iterative methods. We will also cover how these methods are used.

Killer applications of zero-sum games include poker (as we saw), other recreational two-player games, and generative-adversarial networks (GANs). Other applications that are, as of yet, less proven to be effective in practice are robust sequential decision making (the adversary represents uncertainty), and security scenarios where we assume the world is adversarial.

2.2 Stackelberg Equilibrium

A second game-theoretic solution concept that has had extensive application in practice is what's called a *Stackelberg equilibrium*. We will primarily study Stackelberg equilibrium in the context of what is called *security games* [15].

Imagine the following scenario: we wish to protect rhinos in a national park from poaching. There are 20 different watering holes that the rhinos frequent. We have 5 teams of guards that can patrol watering holes. How can we effectively combat poaching? If we come up with a fixed patrol schedule then the poachers can observe us for a few days and learn our schedule exactly. Afterwards they can strike at a waterhole that is guaranteed to be empty at some particular time. Thus we need to design a schedule that is unpredictable, but which also accounts for the fact that some watering holes are more frequented by rhinos (and are thus higher value), travel constraints, etc.

In the security games literature, the most popular solution concept for this kind of setting is the Stackelberg equilibrium. In a Stackelberg equilibrium, we assume that we, as the leader (e.g. the park rangers), get to commit to our (possibly randomized) strategy first. Then, the follower observes our strategy and best responds. This turns out to yield a different solution concept from Nash equilibrium in the case of general-sum games.

However, if we want to help the park rangers design their schedules then we will need to be able to compute Stackelberg equilibria of the resulting game model. Again, we will see that optimization is one of the fundamental pillars of the field of security games research. A unique feature of security games is that the strategy space of the leader is typically some combinatorial polytope (e.g. a restriction on the *transportation polytope*), and the problem of computing a Stackelberg equilibrium is intimately related to optimization over the underlying polytope of the defender (see Xu [18] for some nice consequences of this observation). Because of this combinatorial nature, security games often end up being much harder to solve than zero-sum Nash equilibrium. Therefore, the focus of this section will be on combinatorial approaches to this problem, such as mixed-integer programming, and decomposition. Another crucial aspect of security games is having good models of the attacker. Thus, if time permits, we will also spend some time learning how one can model adversaries using machine learning.

Killer applications of Stackelberg games are mainly in the realm of security. They have been applied in infrastructure security (airports, coast guard, air marshals) [15], to protect wildlife [10], and to combat fare evasion. A nascent literature is also emerging in cybersecurity. Outside of the world of security Stackelberg games are also used to model things like first-mover advantage in the business world.

3 Market Design

The second pillar of the notes will be *market design*. In market design we are typically concerned with how to design the rules of the game, and how to do that in order to achieve "good" outcomes.

Thus, game theory is a key tool in market design, since it will be our way of understanding what outcomes we may expect to arise, given a set of market rules.

Market design is a huge area, and so it has many killer applications. The ones we will see in this course include Internet ad auctions and how to fairly assign course seats to students. However there are many others such as how to price and assign rideshares at Lyft/Uber, how to assign NYC kids to schools, how to enable nationwide kidney exchanges, how to allocate spectrum, etc.

3.1 Fair Allocation

For example, imagine that we are designing a mechanism for managing course enrollment. How should we decide which students get to take which courses? What do we do with the fact that our deep learning course has 100 seats and 500 people that want to take it? Overall, we would like the system to somehow be efficient, but what does that mean? We would also like the system to be fair, but it's not entirely clear what that means either.

At a loss for ideas, we come up with the following solution: we will just have a sign-up system where students can sign up until a course fills up. After that we put other students on a waitlist that we clear on a first-in first-out basis as seats become available. Is this a good system? Well, let's look at a simple example: we will have 2 students and 2 courses, each course having 1 seat. Students are allowed to take at most one course. Let's say that each student values the courses as follows:

	Course A	Course B
Student 1	5	5
Student 2	2	8

Student 1 arrives first and signs up for course B. Then Student 2 arrives and signs up for A. The total *welfare* of this assignment is $5 + 2 = 7$. This does not seem to be an efficient use of resources: we can improve our solution by swapping the courses, since Student 1 gets the same utility as before, and Student 2 improves their utility. This is what's called a *Pareto-improving* allocation because each student is at least as well off as before, and at least one student is strictly better off. One desiderata for efficiency is that no such improvement should be possible; an allocation with this property is called *Pareto efficient*.

Let's look at another example. Now we have 2 students and 4 courses, where each student takes 2 courses. Again courses have only 1 seat.

	Course A	Course B	Course C	Course D
Student 1	10	10	1	1
Student 2	10	10	1	1

Now say that Student 1 shows up first, and signs up for A and B. Then Student 2 shows up and signs up for C and D. Call this assignment x_1. Here we get that x_1 is Pareto efficient, but it does not seem fair. A fairer solution seems to be that each students get a course with value 10 and a course with value 1, let x_2 be such an allocation. One way to look at this improvement is through the notion of *envy*: each student should like their own course schedule at least as well as that of any other student. Under x_1 Student 2 envies Student 1, whereas under x_2 no student envies the other. Fairness turns out to be a complicated idea, and we will see later that there are several appealing notions that we may wish to strive for.

Instead of first-come-first-serve, we can use ideas from market design to get a better mechanism. The solution that we will learn about is based on a fake-money market: we give every student some fixed budget of fake currency (aka funny money). Then, we treat the assignment problem as

a market problem under the assigned budgets, and ask for what is called a *market equilibrium*. Briefly, a market equilibrium is a set of prices, one for each item, and an allocation of items to buyers. The allocation must be such that every item is fully allocated, and every buyer is getting an assignment that maximizes their utility given the prices and their budget. Given such a market equilibrium, we then take the allocation from the equilibrium, throw away the prices (the money was fake anyway!), and use that to perform our course allocation. This turns out to have a number of attractive fairness and efficiency properties. This system is deployed at several business schools such as Wharton (UPenn), Rotman (U Toronto), and Tuck (Dartmouth) [5, 6].

Of course, if we want to implement this protocol we need to be able to compute a market equilibrium. This turns out to be a rich research area: in the case of what is called a *Fisher market*, where each agent i has a linear utility function $v_i \in \mathbb{R}_+^m$ over the m items in the market there is a neat convex program that results in a market equilibrium [9]:

$$\begin{aligned} \max_{x \geq 0} \quad & \sum_i B_i \log(v_i \cdot x_i) \\ \text{s.t.} \quad & \sum_i x_{ij} \leq 1, \forall j \end{aligned}$$

Here x_{ij} is how much buyer i is allocated of item j. Notice that we are simply maximizing the budget-weighted logarithmic utilities, with no prices! It turns out that the prices are the dual variables on the supply constraints. We will see some nice applications of convex duality and Fenchel conjugates in deriving this relationship. We will also see that this class of markets have a relationship to the types of auction systems that are used at Google and Facebook [7, 8].

In the case of markets such as those for course seats, the problem is computationally harder and requires combinatorial optimization. Current methods use a mixture of MIP and local search [6].

4 Target Audience

These notes are targeted at senior undergraduates, master's, and Ph.D. students in operations research and computer science. They assume a basic background in convex, linear, and integer optimization. They also assume knowledge of basic computational complexity theory (e.g. that mixed-integer programming is NP-hard). Most of these things can be learned alongside the course. The course does not assume any background in game theory or mechanism design.

5 Acknowledgments

These lecture notes (the present one and the following ones) owe a large debt to several other professors that have taught courses on Economics and Computation. In particular, Tim Roughgarden's lecture notes [14] and video lectures, John Dickerson's course at UMD[1], and Ariel Procaccia's course at CMU[2] provided inspiration for course topics as well as presentation ideas.

I would also like to thank the following people for extensive feedback on the lecture notes: Ryan D'Orazio for both finding mistakes and providing helpful suggestions on presenting Blackwell approachability. Gabriele Farina for discussions around several regret minimization issues. And for helping me develop much of my thinking on regret minimization in general.

I also thank the following people who pointed out mistakes and typos in the notes: Mustafa Mert Çelikok, Ajay Sakarwal, Eugene Vinitsky.

[1] https://www.cs.umd.edu/class/spring2018/cmsc828m/

[2] http://www.cs.cmu.edu/ arielpro/15896s16/index.html

References

[1] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit hold'em poker is solved. *Science*, 347(6218):145–149, 2015.

[2] Noam Brown and Tuomas Sandholm. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

[3] Noam Brown and Tuomas Sandholm. Solving imperfect-information games via discounted regret minimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1829–1836, 2019.

[4] Noam Brown and Tuomas Sandholm. Superhuman AI for multiplayer poker. *Science*, 365 (6456):885–890, 2019.

[5] Eric Budish. The combinatorial assignment problem: Approximate competitive equilibrium from equal incomes. *Journal of Political Economy*, 119(6):1061–1103, 2011.

[6] Eric Budish, Gérard P Cachon, Judd B Kessler, and Abraham Othman. Course match: A large-scale implementation of approximate competitive equilibrium from equal incomes for combinatorial allocation. *Operations Research*, 65(2):314–336, 2016.

[7] Vincent Conitzer, Christian Kroer, Eric Sodomka, and Nicolás E Stier-Moses. Multiplicative pacing equilibria in auction markets. In *International Conference on Web and Internet Economics*, 2018.

[8] Vincent Conitzer, Christian Kroer, Debmalya Panigrahi, Okke Schrijvers, Eric Sodomka, Nicolas E Stier-Moses, and Chris Wilkens. Pacing equilibrium in first-price auction markets. In *Proceedings of the 2019 ACM Conference on Economics and Computation*. ACM, 2019.

[9] Edmund Eisenberg and David Gale. Consensus of subjective probabilities: The pari-mutuel method. *The Annals of Mathematical Statistics*, 30(1):165–168, 1959.

[10] Fei Fang, Thanh H Nguyen, Rob Pickles, Wai Y Lam, Gopalasamy R Clements, Bo An, Amandeep Singh, Brian C Schwedock, Milin Tambe, and Andrew Lemieux. Paws—a deployed game-theoretic application to combat poaching. *AI Magazine*, 38(1):23–36, 2017.

[11] Samid Hoda, Andrew Gilpin, Javier Pena, and Tuomas Sandholm. Smoothing techniques for computing Nash equilibria of sequential games. *Mathematics of Operations Research*, 35(2): 494–512, 2010.

[12] Christian Kroer, Gabriele Farina, and Tuomas Sandholm. Solving large sequential games with the excessive gap technique. In *Advances in Neural Information Processing Systems*, pages 864–874, 2018.

[13] Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

[14] Tim Roughgarden. *Twenty lectures on algorithmic game theory*. Cambridge University Press, 2016.

[15] Milind Tambe. *Security and game theory: algorithms, deployed systems, lessons learned*. Cambridge university press, 2011.

[16] Oskari Tammelin, Neil Burch, Michael Johanson, and Michael Bowling. Solving heads-up limit Texas hold'em. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[17] Bernhard von Stengel. Efficient computation of behavior strategies. *Games and Economic Behavior*, 14(2):220–246, 1996.

[18] Haifeng Xu. The mysteries of security games: Equilibrium computation becomes combinatorial algorithm design. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, pages 497–514. ACM, 2016.