IEOR8100: Economics, AI, and Optimization
Lecture Note 2: Intro to Game Theory and Regret

Christian Kroer[*]

February 12, 2020

1 Nash Equilibrium

In this lecture we begin our study of Nash equilibrium. First we will learn the basic definitions, and then we will get started on regret minimization, which will be an essential tool for will computing Nash equilibrium later.

1.1 General-Sum Games

A *normal-form game* consists of:

- A set of players $N = \{1, \dots, n\}$
- A set of strategies $S = S_1 \times S_2 \times \cdots \times S_n$
- A utility function $u_i : S \to \mathbb{R}$

We will use the shorthand s_{-i} to denote the subset of a strategy vector s that does not include player i's strategy.

As a first solution concept we will consider *dominant-strategy equilibrium* (DSE). In DSE, we seek a strategy vector $s \in S$ such that each s_i is a best response *no matter what s_{-i} is.* A classic example is the *prisoner's dilemma*: two prisoners are on trial for a crime. If neither confesses (stay silent) to the crime then they will each get 1 year in prison. If one person confesses and the other does not, then the confessor gets no time, but their co-conspirator gets 9 years. If both confess then they both get 6 years.

	Silent	Confess
Silent	-1,-1	-9,0
Confess	0,-9	-6,-6

In this game, confessing is a DSE: it yields greater utility than staying silent no matter what the other player does. A DSE rarely exists in practice, but it can be useful in the context of mechanism design, where we get to decide the rules of the game. It is the idea underlying e.g. the second-price auction which we will cover later.

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

Consider some strategy vector $s \in S$. We say that s is a *pure-strategy Nash equilibrium* if for each player i and each alternative strategy $s'_i \in S_i$:

$$u_i(s) \geq u_i(s_{-i}, s'_i),$$

where s_{-i} denotes all the strategies in s except that of i. A DSE is always a pure-strategy Nash equilibrium, but not vice versa. Consider the *Professor's dilemma*,[1] where the professor chooses a row strategy and the students choose a column strategy:

		Students	
		Listen	Sleep
Prof.	Prepare	$10^6, 10^6$	-10,0
	Slack off	0,-10	0,0

In this game there is no DSE, but there's clearly two pure-strategy Nash equilibria: the professor prepares and students listen, or the professor slacks off and students sleep. But these have quite different properties. Thus equilibrium selection is an issue for general-sum games. There are at least two reasons for this: first, if we want to predict the behavior of players then how do we choose which equilibrium to predict? Second, if we want to prescribe behavior for an individual player, then we cannot necessarily suggest that they player some particular strategy from a Nash equilibrium, because if the others player do not play the same Nash equilibrium then it may be a terrible suggestion.

Moreover, pure-strategy equilibria are not even guaranteed to exist, as we saw in the previous lecture with the rock-paper-scissors example.

To fix the existence issue we may consider allowing players to randomize over their choice of strategy (as in rock-paper-scissors where players should randomize uniformly). Let $\sigma_i \in \Delta^{|S_i|}$ denote player i's probability distribution over their strategy, this is called a *mixed strategy.* Let a strategy profile be denoted by $\sigma = (\sigma_1, \dots, \sigma_n)$. By a slight abuse of notation we may rewrite a player's utility function as

$$u_i(\sigma) = \sum_{s \in S} u_i(s) \prod_i \sigma_i(s_i)$$

A (mixed-strategy) Nash equilibrium is a strategy profile σ such that for all pure strategies σ'_i (σ'_i is pure if it puts probability 1 on a single strategy):

$$u_i(\sigma) \geq u_i(\sigma_{-i}, \sigma'_i).$$

Now, Nash's theorem says that

Theorem 1. *Any game with a finite set of strategies and a finite set of players has a mixed-strategy Nash equilibrium.*

Now, since our goal is the prescribe or predict behavior, we would also like to be able to compute a Nash equilibrium. Unfortunately this turns out to be computationally difficult:

Theorem 2 ([4])**.** *The problem of computing a Nash equilibrium in general-sum finite games is PPAD-complete.*

[1]Example borrowed from Ariel Procaccia's slides

We won't go into detail on what the complexity class PPAD is for now, but suffice it to say that it is weaker than the class of NP-complete problems (it is not hard to come up with a MIP for computing a Nash equilibrium, for example), but still believed to take exponential time in the worst case.

As a sidenote, one may make the following observation about why Nash equilibrium does not "fit" in the class of NP-complete problems: typically in NP-completeness we ask questions such as "does there exist a satisfying assignment to this Boolean formula?" But given a particular game, we already know that a Nash equilibrium exists. Thus we cannot ask about the type of existence questions typically used in NP-complete problems, but rather it is only the task of finding one of the solutions that is difficult. This can be a useful notion to keep in mind when encountering other problems that have guaranteed existence. That said, once one asks for additional properties such as "does there exist a Nash equilibrium where the sum of utilities is at least v?" one gets an NP-complete problem [5, 3].

1.2 Zero-Sum Games

In the special case of a two-player zero-sum game, we have $u_1(s) = -u_2(s) \forall s \in S$. In that case, we can represent our problem as the bilinear saddlepoint problem we saw in the last lecture:

$$\min_{x \in \Delta^n} \max_{y \in \Delta^m} \langle x, Ay \rangle.$$

A first observation one may make is that the minimization problem faced by the x-player is a convex optimization problem, since the max operation is convexity-preserving. This suggests that we should have a lot of algorithmic options to use. This turns out to be true: unlike the general case, we can compute a zero-sum equilibrium in polynomial time using linear programming (LP).

In fact, we have the following stronger statement, which is essentially equivalent to LP duality:

Theorem 3 (von Neumann's minimax theorem)**.** *Every two-player zero-sum game has a unique value v, called the* value of the game*, such that*

$$\min_{x \in \Delta^n} \max_{y \in \Delta^m} \langle x, Ay \rangle = \max_{y \in \Delta^m} \min_{x \in \Delta^n} \langle x, Ay \rangle = v.$$

Next lecture we will prove a more general version of this theorem.

Because zero-sum Nash equilibria are min-max solutions, they are the best that a player can do, given a worst-case opponent. This guarantee is the rationale for saying that a given game has been *solved* if a Nash equilibrium has been computed. Some games such as rock-paper-scissors are trivially solvable as we know that uniform distribution is the only equilibrium. However, this notion has also been applied to heads-up limit Texas hold'em, the smallest poker variant played by humans. In 2015 [1] *essentially solved* that game. Their notion of essentially solved is based on having computed a strategy that is statistically indistinguishable from a Nash equilibrium in a lifetime of human-speed play. The statistical notion was necessary because their solution was computed using iterative methods that only converge to an equilibrium in the limit (but in practice get quite close very rapidly). The same argument is also used in constructing AIs for even larger two-player zero-sum poker games where we can only try to approximate the equilibrium.

Note that this guarantee does not hold in general-sum games, where we have no payoff guarantees if our opponent does not play their part of the same Nash equilibrium that we play. Interestingly, [2] used the same methods that are used for computing two-player zero-sum Nash and applied them in the context of a 6-player poker AI. This AI ended up beating professional human players, in spite of the methods having no guarantees on performance, nor even of converging to a general-sum Nash equilibrium.

Here is another interesting property of zero-sum Nash equilibrium: it is interchangeable. Meaning that if you take an equilibrium (x, y) and another equilibrium (x', y') then (x, y') and (x', y) are also equilibria. This is easy to see from the minimax formulation.

2 Regret Minimization

Now we'll get started on how to compute Nash equilibrium. The fastest methods for computing large-scale zero-sum Nash equilibrium are based on what's called *regret minimization*. In the simplest regret-minimization setting we imagine that we are faced with the task of choosing among a finite set of n actions. After choosing an action, a loss between 0 and 1 is revealed for each action. This scenario is then repeated iteratively. The key is that the losses may be adversarial, and we would like to come up with a decision-making procedure that does at least as well as the single best action in hindsight. We will be allowed to choose a distribution over actions, rather than a single action, at each decision point. Classical example applications would be picking stocks, picking which route to take to work in a routing problem, or weather forecasting. To be concrete, imagine that we have n weather-forecasting models that we will use to daily forecast the weather. We would like to decide which model is best to use, but we're not sure how to pick the best one. In that case, we may run a regret-minimization algorithm, where our "action" is to pick a model, or a probability distribution over models, to forecast the weather with. If we spend enough days forecasting, then our average prediction will be as good as the best single model in hindsight.

As can be seen from the above examples, regret minimization methods are widely applicable beyond equilibrium. The next 2-3 lectures will be on regret minimization, with connections to equilibrium computation covered as we go along.

2.1 Setting

Formally, we are faced with the following problem: at each time step $t = 1, \dots, T$:

1. We recommend a decision $x_t \in \Delta^n$
2. A loss vector $g_t \in [0, 1]^n$ is revealed to us, and we pay the loss $\langle g_t, x_t \rangle$

Our goal is to develop an algorithm that recommends good decisions. A natural goal would be to do as well as the best sequence of actions in hindsight. But this turns out to be too ambitious, as the following example shows

Example 1. *We have 2 actions a_1, a_2. At timestep t, if our algorithm puts probability greater than $\frac{1}{2}$ on action a_1, then we set the loss to $(1, 0)$, and vice versa we set it to $(0, 1)$ if we put less than $\frac{1}{2}$ on a_1. Now we face a loss of at least $\frac{T}{2}$, whereas the best sequence in hindsight has a loss of 0.*

Instead, our goal will be to minimize *regret*. The regret at time t is how much worse our sequence of actions is, compared to the best single action in hindsight:

$$R_t = \sum_{\tau=1}^{t} \langle g_\tau, x_\tau \rangle - \min_{x \in \Delta^n} \sum_{\tau=1}^{t} \langle g_\tau, x \rangle$$

We say that an algorithm is a *no-regret algorithm* if for every $\epsilon > 0$, there exists a sufficiently-large time horizon T such that $\frac{R_T}{T} \le \epsilon$.

Let's see an example showing that randomization is necessary. Consider the following natural algorithm: at time t, choose the action that minimizes the loss seen so far, where e_i is the vector of all zeroes except index i is 1:

$$x_{t+1} = \text{argmax}_{x \in \{e_1, \dots, e_n\}} \sum_{\tau=1}^{t} \langle g_\tau, x \rangle. \tag{FTL}$$

This algorithm is called *follow the leader* (FTL). Note that it always chooses a deterministic action. The following example shows that FTL, as well as any other deterministic algorithm, cannot be a no-regret algorithm

Example 2. *At time t, say that we recommend action i. Since the adversary gets to choose the loss vector after our recommendation, let them choose the loss vector be such that $g_i = 1$, $g_j = 0 \forall j \neq i$. Then our deterministic algorithm has loss T at time T, whereas the cost of the best action in hindsight is at most $\frac{T}{n}$.*

It is also possible to derive a lower bound showing that any algorithm must have regret at least $O(\sqrt{T})$ in the worst case, see e.g. [6] Example 17.5.

2.2 The Hedge Algorithm[2]

We now show that, while it is not possible to achieve no-regret with deterministic algorithms, it is possible with randomized ones. We will consider the *Hedge* algorithm. It works as follows:

- At $t = 1$, initialize a weight vector w^1 with $w_i^1 = 1$ for all actions i
- At time t, choose actions according to the probability distribution $p_i = \frac{w_i^t}{\sum_j w_j^t}$
- After observing g_t, set $w_i^{t+1} = w_i^t \cdot e^{-\eta g_{t,i}}$, where η is a stepsize parameter

The stepsize η controls how aggressively we respond to new information. If $g_{t,i}$ is large then we decrease the weight w_i more aggressively.

Theorem 4. *Consider running Hedge for T timesteps. Hedge satisfies*

$$R_T \leq \frac{\eta T}{2} + \frac{\log n}{\eta}$$

Proof. Let g_t^2 denote the vector of squared losses. Let $Z_t = \sum_j w_j^t$ be the sum of weights at time t. We have

$$\begin{aligned}
Z_{t+1} &= \sum_{i=1}^{n} w_i^t e^{-\eta g_{t,i}} \\
&= Z_t \sum_{i=1}^{n} x_{t,i} e^{-\eta g_{t,i}} \\
&\leq Z_t \sum_{i=1}^{n} x_{t,i} (1 - \eta g_{t,i} + \frac{\eta^2}{2} g_{t,i}^2) \\
&= Z_t (1 - \eta \langle x_t, g_t \rangle + \frac{\eta^2}{2} \langle x_t, g_t^2 \rangle) \\
&\leq Z_t e^{-\eta \langle x_t, g_t \rangle + \frac{\eta^2}{2} \langle x_t, g_t^2 \rangle}
\end{aligned}$$

[2]We did not have time to cover this section in lecture 2, it will be covered at the beginning of lecture 3.

where the first inequality uses the second-order Taylor expansion $e^{-x} \leq 1 - x + \frac{x^2}{2}$ and the second inequality uses $1 + x \leq e^x$.

Telescoping and using $Z_1 = n$, we get

$$Z_{T+1} \leq n \prod_{t=1}^{T} e^{-\eta\langle x_t, g_t\rangle + \frac{\eta^2}{2}\langle x_t, g_t^2\rangle} = n e^{-\eta \sum_{t=1}^{T}\langle x_t, g_t\rangle + \frac{\eta^2}{2}\sum_{t=1}^{T}\langle x_t, g_t^2\rangle}$$

Now consider the best action in hindsight i^*. We have

$$e^{-\eta \sum_{t=1}^{T} g_{t,i^*}} = w_{i^*}^{T+1} \leq Z_{T+1} \leq n e^{-\eta \sum_{t=1}^{T}\langle x_t, g_t\rangle + \frac{\eta^2}{2}\sum_{t=1}^{T}\langle x_t, g_t^2\rangle}$$

Taking logs gives

$$-\eta \sum_{t=1}^{T} g_{t,i^*} \leq \log n - \eta \sum_{t=1}^{T}\langle x_t, g_t\rangle + \frac{\eta^2}{2}\sum_{t=1}^{T}\langle x_t, g_t^2\rangle.$$

Now we rearrange to get

$$R_T \leq \frac{\log n}{\eta} + \frac{\eta}{2}\sum_{t=1}^{T}\langle x_t, g_t^2\rangle \leq \frac{\log n}{\eta} + \frac{\eta T}{2},$$

where the last inequality follows from $x_t \in \Delta^n$ and $g_t \in [0,1]^n$. □

If we know T in advance we can now set $\eta = \frac{1}{\sqrt{T}}$ to get that Hedge is a no-regret algorithm.

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 multiplayer poker. *Science*, 365 (6456):885–890, 2019.

[3] Vincent Conitzer and Tuomas Sandholm. New complexity results about Nash equilibria. *Games and Economic Behavior*, 63(2):621–641, 2008.

[4] Constantinos Daskalakis, Paul W Goldberg, and Christos H Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

[5] Itzhak Gilboa and Eitan Zemel. Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*, 1(1):80–93, 1989.

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