Algorithms, Games, and Networks January 24, 2013

Lecture 4

Lecturer: Avrim Blum *Scribe: Christian Tjandraatmadja*

1 Overview

In this lecture, we finish our discussion on learning algorithms to minimize regret. We present an algorithm that minimizes swap regret and therefore approaches a correlated equilibrium. This algorithm uses as subroutine an algorithm that minimizes external regret.

After that, we discuss the Stackelberg leader strategy and then consider the hardness of problems involving Nash equilibria. We see (without proving) that finding a Nash equilibrium is PPAD-hard, define the complexity class PPAD, and finally show why it is in PPAD by presenting the Lemke-Howson algorithm, an algorithm to find a Nash equilibrium.

Readings: Sections 4.5 and 2.1-2.4 of the Algorithmic Game Theory book.

Note: Section 2 of these lecture notes is related to the final slides of Lecture 3.

2 Minimizing swap regret and finding correlated equilibria

In the previous lecture, we saw more general forms of regret and connected them to correlated equilibrium. To recap:

- *Internal regret* is regret of the form "every time I chose action i, I should have chosen action j instead". In other words, it is a rewiring of a single action to another.
- *Swap regret* generalizes internal regret so that we allow a rewiring from any action to any other action. That is, we consider a mapping from actions $\{1, \ldots, n\}$ to $\{1, \ldots, n\}$.
- *Correlated equilibrium* is a distribution over entries in the matrix such that if a trusted party picks an entry under the distribution and tells each player its action, that player has no incentive to deviate given that others also do not deviate.
- *Coarse-correlated equilibrium* is similar to correlated equilibrium except that each player decides whether to commit to the advice before seeing it or play normally. In this case, each player has no incentive not to commit given that others commit.
- If all players run a low swap regret algorithm, then the empirical distribution of play is an approximate correlated equilibrium.

2.1 An algorithm for minimizing swap regret

In this section, we describe an algorithm by Blum and Mansour (2005) that converts any low external regret algorithm (for example, the Randomized Weighted Majority algorithm) into a low swap regret algorithm. The algorithm, illustrated by Figure 1 below, is as follows.

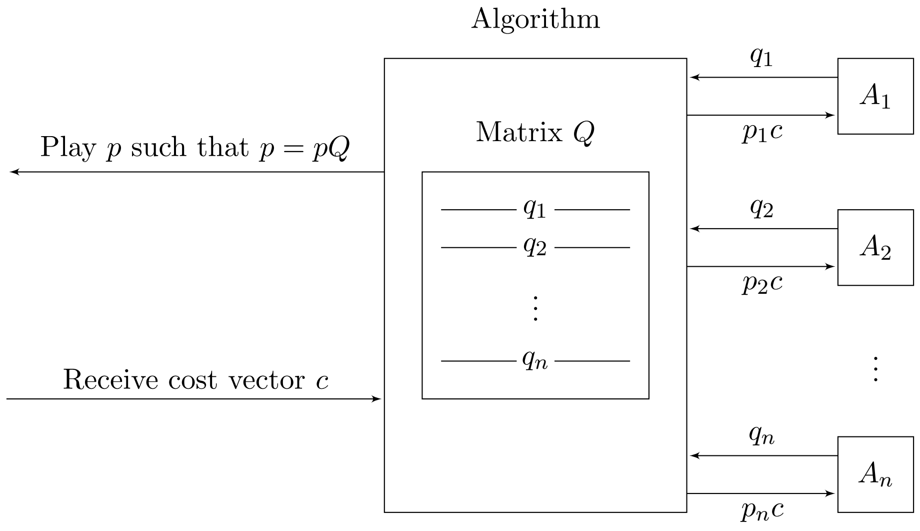


Figure 1: The structure of the swap regret minimization algorithm.

Instantiate n copies $A_1, \ldots, A_n$ of an external regret minimization algorithm (n is the number of actions). We can interpret A_j for each j as responsible for the expected regret over times we play action j instead of a better action. Each algorithm A_j will propose a distribution q_j over the actions. Let Q be the matrix composed of rows q_j. In the first step, all proposed distributions q_j are uniform and equal (in other words, all entries in Q are $\frac{1}{n}$).

Given Q, compute p such that $p = pQ$. This is possible because if we interpret Q as the transition matrix of a Markov chain, then p is its stationary distribution, which we know exists and is efficiently computable. This choice of p enables us to view p as not only the distribution to select an action j directly, but also as the distribution to select an algorithm A_j that is used to select an action (since this gives us the distribution $\sum_j p_j q_j = pQ$).

After playing distribution p, the world gives us back a cost vector c. Split c back to the algorithms A_j, feeding each algorithm $p_j c$ and repeating the process.

Now let us see why this algorithm guarantees us low swap regret. In T rounds, each algorithm A_j naturally guarantees

$$\underbrace{\sum_{t=1}^{T} (p_j^t c^t) q_j^t}_{\text{expected cost}} \leq \underbrace{\min_i \sum_{t=1}^{T} p_j^t c_i^t}_{\substack{\text{best fixed action cost} \\ \text{(``best expert'')}}} + \underbrace{R}_{\substack{\text{regret term} \\ \text{of the algorithm}}}$$

This can be trivially rearranged to

$$\sum_{t=1}^{T} p_j^t(q_j^t c^t) \le \min_i \sum_{t=1}^{T} p_j^t c_i^t + R$$

Summing the inequalities for all algorithms j, we get

$$\sum_{t=1}^{T} p^t Q^t c^t \le \sum_{j=1}^{n} \min_i \sum_{t=1}^{T} p_j^t c_i^t + nR$$

To see how the above inequality bounds swap regret, first recall that we play $p = pQ$. This means the left hand side $\sum_{t=1}^{T} p^t Q^t c^t = \sum_{t=1}^{T} p^t c^t$ is the total expected cost of our algorithm. The right hand side term $\sum_{j=1}^{n} \min_i \sum_{t=1}^{T} p_j^t c_i^t$ is the total expected cost after applying the best action mapping: for each action j, apply the probability of playing j at each round to the best action i, and sum up the expected costs of every (remapped) action to obtain the total expected cost.

This means that if we have an algorithm with external regret R, we can build an algorithm with swap regret nR. In particular, the Randomized Weighted Majority yields regret $O(\sqrt{T \log n})$, which implies we have an algorithm with swap regret $O(n\sqrt{T \log n})$.

2.2 Another way to approach a correlated equilibrium

The algorithm we described approaches a correlated equilibrium, but we can also find one by solving a linear program. Consider a two-player game (R, C) (where R and C are the payoff matrices of the row and column players respectively).

Our goal is to find a distribution p over the matrix. Let p_{ij} be the set of variables that represent this distribution and denote by p_i the probability of choosing row i, that is, $\sum_j p_{ij}$. By the definition of a correlated equilibrium, for each row i, the conditional expected payoff of playing i given that the trusted party chose row i (with probability p_i) is at least as good as playing some other row i'. This corresponds to the following constraint:

$$\sum_j \frac{p_{ij}}{p_i} R_{ij} \ge \sum_j \frac{p_{ij}}{p_i} R_{i'j} \quad \forall \text{ rows } i, i'$$

This constraint can be linearized simply by multiplying both sides by p_i, resulting in

$$\sum_j p_{ij} R_{ij} \ge \sum_j p_{ij} R_{i'j} \quad \forall \text{ rows } i, i'$$

Similarly, the analogous constraint is valid for the column player.

$$\sum_i p_{ij} C_{ij} \ge \sum_i p_{ij} C_{ij'} \quad \forall \text{ columns } j, j'$$

Finally, p should of course also satisfy the constraints of being a probability distribution: $p_{ij} \geq 0, \forall i, j$ and $\sum_i \sum_j p_{ij} = 1$.

If there are more than two players, we can use an ellipsoid algorithm or just run a swap regret minimization algorithm for each player. So we see that we can approach correlated equilibrium with efficient algorithms, which is a nice feature of correlated equilibria.

3 The Stackelberg leader strategy

Consider the *ultimatum game*: there are two players, the splitter and the chooser. A third party puts \$10 on the table and the splitter decides how to split the money between the two. The chooser can either accept or reject. If the chooser accepts, each player receives its part. Otherwise, the money is burned.

If we think of this game considering psychology, the chooser may reject a low amount because people want to be treated fairly. However, think of it in terms of game theory and let the payoff be only the money. In this case, the chooser would never reject any nonzero amount, so the splitter would offer \$1. On the other hand, suppose the chooser locks in a program that only accepts \$9. Now the splitter is stuck on offering \$9. This is a kind of game in which if you can reliably commit to a strategy first, you have an advantage.

This is called a *Stackelberg leader strategy*: a (pure or mixed) strategy where if you reliably announce and commit to it and the other players do best response, then you are best off (if mixed, the distribution is announced but not the choice itself).

This strategy does not need to be a Nash equilibrium. Here is another example: suppose you are a big player R in a market and you want to choose a price so that a competitor C does not enter the market. As described in Figure 2, if you price low, you make less, but you drive the competitor out of the market.

R \ C	Compete	Leave
Price high	3 3	1 6
Price low	0 2	1 4

Figure 2: An example of a game in which a Stackelberg leader strategy applies.

A Stackelberg leader strategy in this case is to commit to pricing low in order to force the competitor to leave, which is better than competing in either case. Note that this is not a Nash equilibrium because you have incentive to price high.

Finding a Stackelberg leader strategy can be done efficiently using linear programming. If we are the row player, for each column j, find a distribution p that maximizes our expected gain such that j is best response, and then choose the best distribution.

A practical example of this notion appears in security games. Police at the Los Angeles International Airport must decide where to place units in order to catch smugglers. Their solution is based on a Stackelberg leader strategy: they choose the placement assuming smugglers know what they are doing and do best response.

4 Hardness of computing Nash equilibria

Consider two-player games. There are two types of results about hardness of Nash equilibria:

1. Some problems regarding Nash equilibria with special properties are NP-hard. For example, is there a Nash equilibria with payoff at least v? Is there one that uses the first row? Is there one that does not use the first row? Is there more than one?
2. Finding a Nash equilibrium is PPAD-hard.

Results of the first type are generally developed through a reduction to 3-SAT. A rough idea of the reduction is as follows. A game is *symmetric* if $C = R^T$, where R and C are the payoff matrices for row and column players respectively.

Given a 3-SAT formula, create a symmetric game with one row and one column for each literal, clause and variable, and one additional action f. The idea is to build this game so that all Nash equilibria are either a pure equilibrium (f, f) or a mixed equilibrium that corresponds to a satisfying assignment ($\frac{1}{n}$ for each variable or its negation, where n is the number of variables). The construction of this game ensures that only satisfying assignments remain as equilibria and the remaining ones drift to (f, f).

4.1 The complexity class PPAD

In this lecture we do not prove that finding a Nash equilibrium is PPAD-hard, but we show that it is in PPAD. In order to do that, let us first define PPAD in terms of a PPAD-complete problem. The complexity class PPAD is composed of the problems that are polynomially reducible to the *end of the line problem*, described as follows and illustrated by Figure 3 in the next page.

Given two circuits C_{next} and C_{prev} (known, not oracles), each with n input bits and n output bits, define a directed graph G with 2^n nodes, each corresponding to an n-bit value, such that there is an arc from u to v if and only if $C_{\text{next}}(u) = v$ and $C_{\text{prev}}(v) = u$. Note that each node has at most one in-arc and one out-arc. This means that G is a collection of directed paths and cycles.

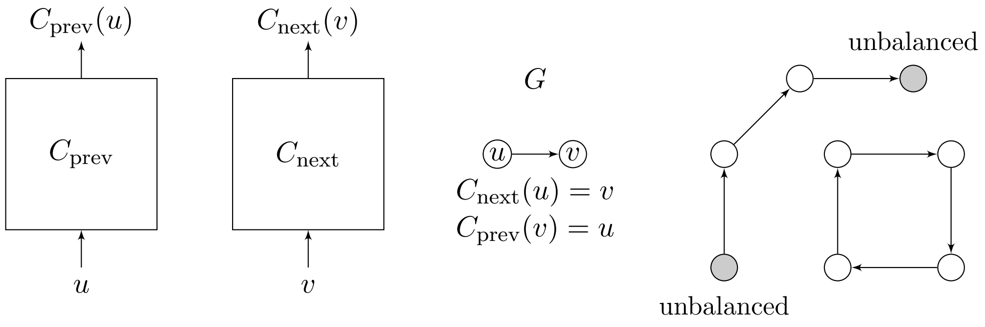


Figure 3: The end of the line problem: find an unbalanced node.

We say that a node is *unbalanced* if its in-degree different than its out-degree. In this case, they are nodes at the start or the end of a directed path. Suppose the circuits define a graph such that 0^n is an unbalanced node. The end of the line problem is to find another unbalanced node. Note that there must exist at least one other unbalanced node at the other end of the path containing 0^n.

At first glance this problem may seem trivial: if we follow the path containing 0^n, we reach an unbalanced node. However, this algorithm is not guaranteed to be polynomial-time since the path could be exponentially long.

In the next section, we show an algorithm that solves the problem of finding a Nash equilibrium. We will see that it is essentially solving an instance of the end of the line problem, implying that the problem is in PPAD.

4.2 The Lemke-Howson algorithm

Note: This algorithm is described both in Sections 2.3 and 3.4 in the Algorithmic Game Theory book. In this lecture, we follow the description on Section 2.3.

In this section, we describe an algorithm by Lemke and Howson (1964) to find a Nash equilibrium for two-player games. It is convenient to first convert a general game to a symmetric game. Given a two-player game (R, C), a standard way to do this is to define a game (A, A^T) such that

$$A = \begin{bmatrix} 0 & R \\ C^T & 0 \end{bmatrix}$$

Claim 1 *If $((x, y), (x, y))$ is a symmetric Nash equilibrium in the game (A, A^T) (where x refers to the first rows $\begin{bmatrix} 0 & R \end{bmatrix}$ or columns $\begin{bmatrix} 0 \\ C^T \end{bmatrix}$ of A and y to the remaining ones), then $\left(\frac{x}{X}, \frac{y}{Y}\right)$ is a Nash equilibrium in the game (R, C) where $X = \sum_i x_i$ and $Y = \sum_i y_i$.*

Proof: The row player's payoff is $x^T R y + y^T C^T x$, which can be rewritten as $x^T R y + x^T C y$.

The column player's payoff is the same. Since this is a symmetric Nash equilibrium, neither player wants to deviate. In particular, neither wants to deviate internally on x or y, that is, fix x and deviate on y or vice versa. This means that neither of the terms $x^T R y$ or $x^T C y$ in the expression may increase through deviation for either player. Thus if you scale x and y to a probability distribution, it has to be a Nash equilibrium (scaling does not affect the equilibrium). ■

Now that we have an $n \times n$ symmetric game, our problem is to find a symmetric Nash equilibrium. Let z be a vector of n variables $z = (z_1, \ldots, z_n)$. Consider the polytope defined by the following $2n$ linear constraints:

$$\begin{aligned} A_i z \leq 1 \quad & \forall i \\ z_j \geq 0 \quad & \forall j \end{aligned}$$

Assume A is composed of only nonnegative entries and has full rank (that is, vertices are not degenerate: each vertex is the intersection of n constraints at equality). Thus, we have a bounded polytope with no zero rows or columns and z cannot be arbitrarily large.

If we define $x_i = \frac{z_i}{Z}$ where $Z = \sum_i z_i$ (assuming $Z \neq 0$), we can rewrite the first inequality as $A_i x \leq \frac{1}{Z}$. This allows us to interpret this inequality as follows: if the other player is playing x, then $A_i x$ is our expected payoff for playing i, so the inequality is tight at x if and only if i is a best response. In other words, z can be viewed as how the other player is playing.

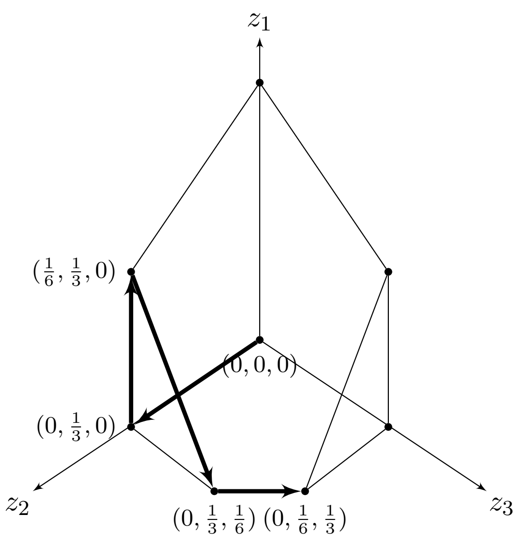
Let us see how this polytope relates to Nash equilibria. A strategy i is *represented* if $A_i z = 1$ or $z_i = 0$ (or both), that is, at least one of the constraints is tight at z. Suppose we have a z where all strategies are represented. Then we claim either $z = (0, \ldots, 0)$ (the origin, where x is not defined), or (x, x) is a symmetric Nash equilibrium. To see why this is true, a key point is that for every i such that $z_i \neq 0$, $A_i z$ must be 1. In other words, for every row i we put nonzero probability, the strategy i must be a best response for x, which implies that (x, x) is a Nash equilibrium.

The general idea of the algorithm is to start at origin zero and move along the edges of the polytope until it reaches a vertex where all strategies are represented, that is, a Nash equilibrium. More precisely, the algorithm (illustrated by Figure 4) is the following.

Lemke-Howson algorithm: Start at $z = (0, \ldots, 0)$. At the first step, relax some $z_j = 0$, moving along an edge until we hit a face $A_i z = 1$ or $z_i = 0$. If all strategies are represented, then we are done. Otherwise, strategy i is represented twice: both $A_i z = 1$ and $z_i = 0$. Afterwards there are two possible constraints to relax, but one of them returns to the previous vertex. Therefore, relax the other constraint. Repeat this process until it reaches a vertex where all strategies are represented.

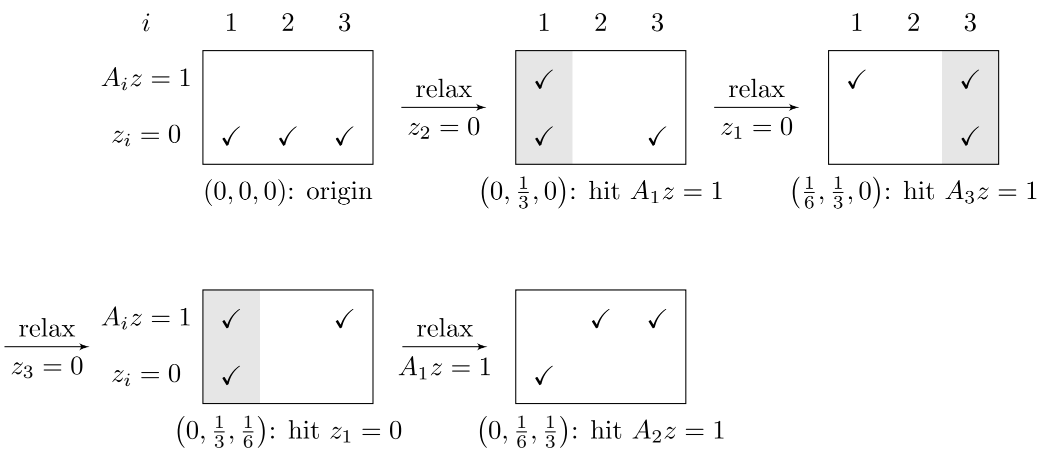
Since the polytope is finite, we must either reach the desired vertex or cycle. It is left to argue that we cannot cycle through edges of the polytope: we can view each vertex as being adjacent to at most two other vertices, each reached by relaxing each constraint of the doubly represented strategy (if not the origin or end vertex). (There of course may be more adjacent vertices in the polytope itself, but we only consider the two corresponding to the doubly represented strategy.) Therefore we cannot have a vertex adjacent to more than two vertices, which would be necessary for the algorithm to cycle. (Note also that we cannot go back to the origin either because we never visit a previously visited vertex.)

Hence, we must eventually reach a vertex corresponding to a symmetric Nash equilibrium. We can also see this as solving an instance of the end of the line problem: at each step, relaxing one constraint goes back and relaxing the other moves forward. Therefore, finding a Nash equilibrium is in PPAD.



z_1
$(\frac{1}{6}, \frac{1}{3}, 0)$
$(0,0,0)$
$(0, \frac{1}{3}, 0)$
z_2
$(0, \frac{1}{3}, \frac{1}{6})$ $(0, \frac{1}{6}, \frac{1}{3})$
z_3

$$A = \begin{bmatrix} 0 & 3 & 0 \\ 0 & 0 & 3 \\ 2 & 2 & 2 \end{bmatrix}$$



i
1 2 3
$A_i z = 1$
$z_i = 0$
✓ ✓ ✓
$(0,0,0)$: origin
$\xrightarrow[z_2 = 0]{\text{relax}}$
$(0, \frac{1}{3}, 0)$: hit $A_1 z = 1$
$\xrightarrow[z_1 = 0]{\text{relax}}$
$(\frac{1}{6}, \frac{1}{3}, 0)$: hit $A_3 z = 1$
$\xrightarrow[z_3 = 0]{\text{relax}}$
$(0, \frac{1}{3}, \frac{1}{6})$: hit $z_1 = 0$
$\xrightarrow[A_1 z = 1]{\text{relax}}$
$(0, \frac{1}{6}, \frac{1}{3})$: hit $A_2 z = 1$

Figure 4: A simulation of the Lemke-Howson algorithm over the example in Section 2.3 of the AGT book. Each checkmark indicates whether the corresponding constraint is satisfied with equality or not. The algorithm terminates when all strategies are represented, that is, each column has a checkmark.