CS295 Introduction to Algorithmic Game Theory

Instructor: Ioannis Panageas Scribed by: Yiwen Ma, Yushen Sun

Lecture 5. Computing NE in two player games.

1 A trivial algorithm

The most natural way to find a Nash Equilibrium (NE) in a two player game is "brute force". As long as this is a finite game, we should be able to enumerate all the possible "supports" of the Nash. For those supports, if we can find a strategy combination of the players, that should be the NE of the game.

1.1 Initial Idea: If the support of the Nash is known

Let's first consider the case when we know the support of a Nash. To figure out what's the situation under such case, let's make some assumptions first.

Assumptions:

- R and C are pay-off matrices of row and column players respectively, with size $m \times n$
- S and T are supports of the Nash

Then we have some properties:

- For x
  - $x_i \leq 0$ for $\forall i \in [n]$
  - $x_i = 0$ for $\forall i \notin S$
  - $\Sigma_{i \in S} x_i = 1$
  - $(C^T x)_i \leq (C^T x)_j$ for $\forall i \in T, j \in [m]$
- For y
  - $y_i \leq 0$ for $\forall i \in [m]$
  - $y_i = 0$ for $\forall i \notin T$
  - $\Sigma_{i \in T} y_i = 1$
  - $(Ry)_i \leq (Ry)_j$ for $\forall i \in S, j \in [n]$

1.2 Linear Program: Derive the Nash Equilibrium

From the properties we derived previously, it's natural to come into a conclusion that given a set of support S and T, it actually generates a linear program problem. As long as we can find a feasible solution (x, y) for the linear program, it is a Nash Equilibrium.

Fortunately, the possible combinations of S and T is finite, which is $2^n \times 2^m = 2^{m+n}$. Therefore, we can enumerate each possible set of supports, and solve the corresponding linear program problem, until we find a feasible solution (x, y).

However, the drawbacks of this algorithm is also obvious. For the worst case, the number of linear program problems we need to solve is $O(2^{m+n})$, which means this algorithm is super expensive in terms of calculation. Therefore, this algorithm only reveals the feasibility to derive Nash Equilibrium, but not a practical or general way to compute NE.

2 Lemke-Howson Algorithm

2.1 Introduction

Lemke-Howson Algorithm is an effective algorithm that finds a Nash Equilibrium(NE) in two player games. The basic idea is to maintain a single guess of the supports, and in each iteration we change the guess only a little bit. In the algorithm we assume that the two payoff matrices have non-negative entries. And no loss of generality, NE are invariant under shifting.

2.2 Preparations

In the algorithm we represent the two polytopes P_1 and P_2 by:

$$P_1 = \{x \in \mathbf{R}^n : \forall i \in [n] \quad x_i \geq 0 \& \forall j \in [m] \quad (x^T C)_j \leq 1\}$$
$$P_2 = \{y \in \mathbf{R}^m : \forall i \in [m] \quad y_i \geq 0 \& \forall j \in [n] \quad (Ry)_j \leq 1\}$$

Def. Then we define the normalization of x and y as:

$$nrml(x) = (\Sigma_{i \in [n]} x_i)^{-1} x$$
$$nrml(y) = (\Sigma_{i \in [m]} y_i)^{-1} y$$

Def. We define "label" as:

x has label i if $x_i = 0$ or $(x^T C)_i = 1$. y has label j if $y_i = 0$ or $(Ry)_j = 1$.

Lemma. Let $x^* \in P_1$, $y^* \in P_2$, x^*, y^* have all labels together and assume x^*, y^* are not zero vectors. It holds that $(nrml(x^*), nrml(y^*))$ is a Nash Equilibrium.
Proof.

- For each $i \in [n]$, either $x_i^* = 0$ or $(Ry^*)_i = 1$ (i is best response of row player to $nrml(y^*)$).
- For each $j \in [m]$, either $y_j^* = 0$ or $({x^*}^T C)_j = 1$ (j is best response of row player to $nrml(x^*)$).

So we can conclude that:

$$\text{if } x_i^* > 0 \Longrightarrow (Ry^*)_i \geq (Ry^*)_j \quad \forall j \in [n]$$

$$\text{if } y_j^* > 0 \Longrightarrow ({x^*}^T C)_i \geq ({x^*}^T C)_j \quad \forall j \in [m]$$

So we can conclude that x^*, y^* is a Nash Equilibrium. Since $(nrml(x^*), nrml(y^*))$ is the normalization form we defined, $(nrml(x^*), nrml(y^*))$ is also a Nash Equilibrium. And x^*, y^* satisfy LP(Supp(x^*),Supp(y^*)), which we discussed in part 1.

Def. We define "Vertex" as: A vertex of polytope P_1 is given by n linearly independent equalities (the rest constrains of P_1 are strict inequalities). A vertex of polytope P_2 is given by m linearly independent equalities (the rest constrains of P_2 are strict inequalities). For $P_1 \bigcup P_2$ is n+m. This is the non-degenerate case.

2.3 The algorithm

Algorithm 1 Lemke-Howson Algorithm

1: Initialize $x = 0$ and $y = 0$.
2: $k = k_0 = 1$.
3: **Loop**
4: In P_1 find the neighbor vertex x' of x with label k' instead of k. Remove label k and add label k'.
5: **Set** x=x'.
6: **If** k'=1 **STOP**.
7: In P_2 find the neighbor vertex y' of y with label k" instead of k'. Remove label k' and add label k".
8: **Set** y=y'.
9: **If** k"=1 **STOP**.
10: **Set** k=k".

Theorem. The Lemke-Howson algorithm outputs a Nash Equilibrium.
Proof. Define a graph with vertices in $P_1 \bigcup P_2$. Each vertex (x,y) has:

- One duplicate label. This vertex is adjacent to exactly two other vertices, since we can remove the duplicate label from x and pivot in P_1, or remove the duplicate label from y and pivot in P_2.

- They have all labels exactly once. This vertex has only one neighbor (remove label 1 from whichever vector has it.)

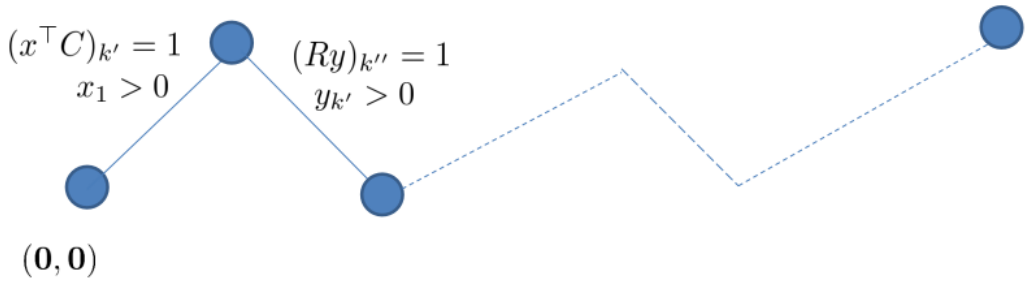

Figure 1: Process of Lemke-Howson algorithm

Since each vertex in the graph has degree 1 or 2, the graph is a union of cycles and paths. Then we know that:

1. Lemke-Howson algorithm begins at the configuration (0,0).
2. (0,0) has all labels and is therefore an endpoint of a path component.
3. The algorithm will terminate at the other endpoint of the path.
4. The other point is not (0,0) and cannot be (x,0) or (0,y).

According to the Lemma in 2.2, we can conclude that the point is a Nash Equilibrium.

3 The Odd-number Corollary of Lemke-Howson Algorithm

3.1 Definition

The definition of the odd-number corollary says that **for non-degenerate games, the number of NE is odd**.

3.2 Proof

From the analysis of Lemke-Howson Algorithm, we can notice that finding NE with Lemke-Howson Algorithm is actually finding some "special nodes" in a graph. More specifically, the graph is composed of simply cycles and paths, and these "special nodes" must exist at the end of a path.

In other words, a NE must be a node in the graph which has odd degree (i.e. number of neighbours). Then we can try to calculate the number of nodes with odd degree. In fact, the number of such nodes must be even, given that:

$$\Sigma d_v = 2 \cdot E$$

which means the total number of degrees in a graph is even. Therefore we have even number of odd-degree nodes including $(0,0)$.However, $(0,0)$ cannot be a NE, so it's excluded, which means that there will be odd-number of NE for this graph, or the game.

4 Approximating a Nash Equilibrium

We've already known that it's very expensive to get the exact NE of a game. Therefore, a reasonable idea is to look for approximates of NE. There exists a theorem which points out the existence of approximates of NE, which says **For any two-player game, let $\epsilon > 0$, there always exists a k-uniform ϵ-approximate NE for** $k = \frac{12logn}{\epsilon^2}$.The definition of k-uniform is as follow:

for strategy x, if every coordinate of x is multiple of $\frac{1}{k}$, then x is k-uniform

Besides, it has already been proved that we can get such approximation of Nash Equilibrium with $n^{O(\frac{logn}{\epsilon^2})}$ time, which we think makes it more feasible to apply Nash Equilibrium into analysis of practical problems.

References

[1] C. E. Lemke and J. T. Howson. *Equilibrium points of bimatrix games.* SIAM Journal on Applied Mathematics. 12 (2): 413–423. doi:10.1137/0112033.