CS 354: Unfulfilled Algorithmic Fantasies April 15, 2019

Lecture 5: PPAD and friends

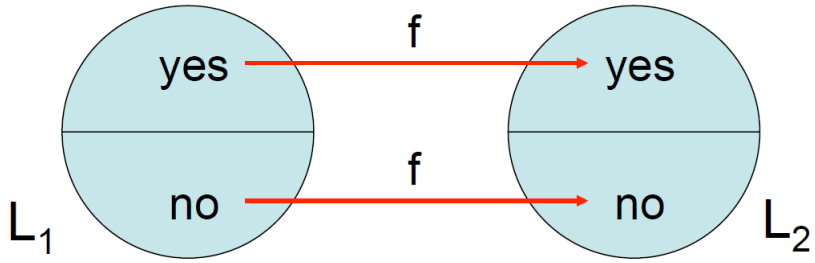
Lecturer: Aviad Rubinstein *Scribe: Jeffrey Gu*

1 Introduction

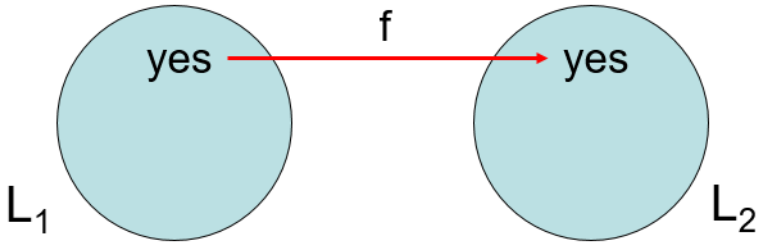
Last week, we saw **query complexity** and **communication complexity**, which are nice because you can prove lower bounds but have the drawback that the complexity is bounded by the size of the instance. Starting from this lecture, we'll start covering **computational complexity** and **conditional hardness**.

2 Reductions

The main tool for proving conditional hardness are **reductions**. For decision problems, reductions can be thought of as functions between two problems that map yes instances to yes instances and no instances to no instances:

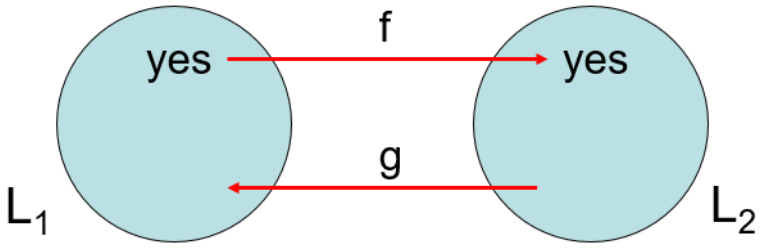


To put it succinct, "yes maps to yes" and "no maps to no". As we'll see, this notion of reduction breaks down for problems that have only "yes" instances, such as the problem of finding a Nash Equilibrium. Nash proved that a Nash Equilibrium always exists, so the Nash Equilibrium and the related ϵ-Nash Equilibrium only have "yes" instances. Now suppose that we wanted to show that Nash Equilibrium is a hard problem. It doesn't make sense to reduce from a decision problem such as 3-SAT, because a decision problem has "no" instances. Now suppose we tried to reduce from another problem with only "yes" instances, such as the End-of-a-Line problem that we've seen before (and define below), as in the above definition of reduction:



For example f may map an instance (S, P) of End-of-a-Line to an instance $f(S, P) = (A, B)$ of Nash equilibrium.

The problem with this reduction is that it allows "trivial" reductions, such as the reduction where every yes instance from L_1 is mapped to a single yes instance of L_2. To fix this, we adopt the following notion of reduction:



where g is a function that maps a solution to the instance of $F(S, P)$ of L_2 back to a solution of the orignal instance (S, P) of L_1.

3 End-of-a-Line and PPAD

In previous lectures, we've worked with the query (and communication) variants of the **end-of-a-line** problem, where given the source of a directed line graph, the goal is to find the (end) sink of the line. We've previously seen a reduction from End-of-a-Line to the problem of finding a ϵ-Nash Equilibrium via a reduction to the problem of finding a Brouwer fixed point, for the query complexity and communication complexity cases. We might suspect that there is a similar reduction under the above definition of reduction.

First, we formally define the computational variant of End-of-a-Line:

Definition 3.1 (End-of-a-Line)**.** In the End-of-a-Line problem, we have:

1. An implicit graph G of exponential size, where the in and out-degree of each vertex is at most one
2. The input is two poly-time computable circuits $S, P : \{0,1\}^n \to \{0,1\}^n$, and there is a special 0 vertex such that $S(0) \neq 0, P(0) = 0$. The two circuits are called the **successor** and **predecessor**.
3. The output is a vertex $v \neq 0$ such that $S(P(v)) \neq v$ or $P(S(v)) \neq v$. Examples of vertices in the graph that meet this condition are the end of any line, start of any lines that's not the 0 vertex, and vertices where S and P are inconsistent.

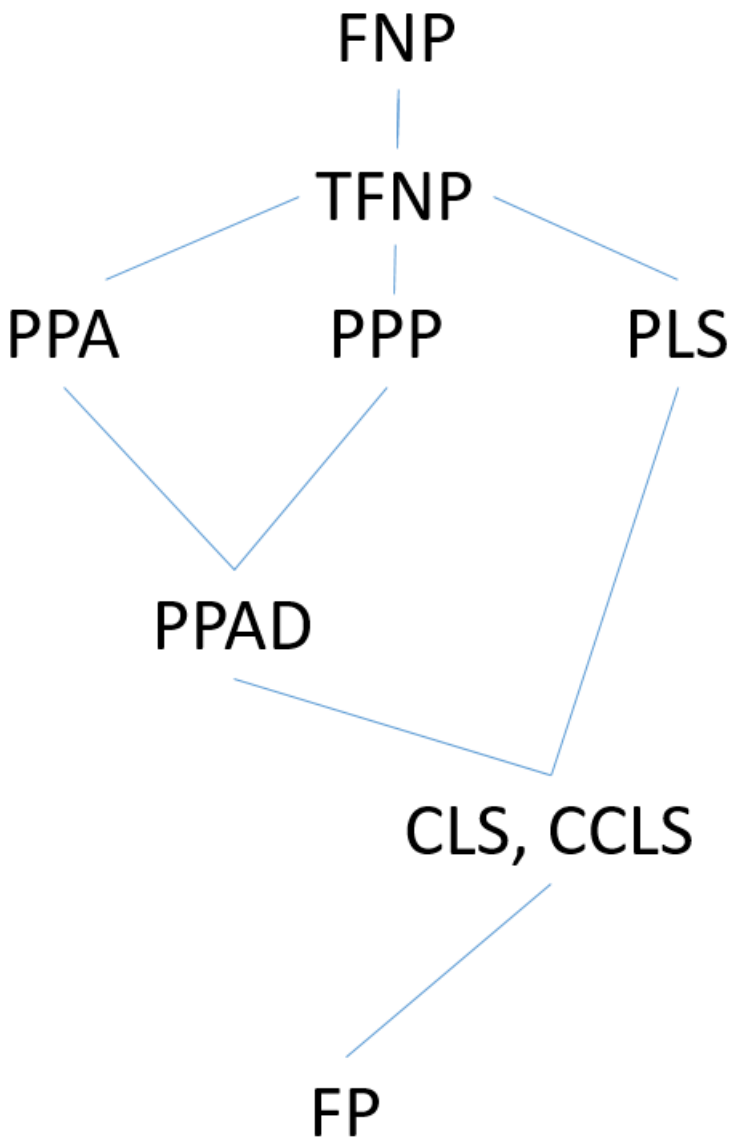
We can now define a new complexity class:

Definition 3.2 (PPAD)**.** PPAD (which stands for **Polynomial Parity Arguments on Directed graphs**) is the class of all problems reducible to End-of-a-Line.

We believe that End-of-a-Line is a hard problem, for two reasons:

- **Query Complexity**: End-of-a-Line is a hard problem in the query complexity model, i.e. when S, P are given as black-box functions. We are interested in the computational variant where we have access to the circuits. But, except for very few exceptions, we don't know any way of looking inside a circuit and extracting useful information from it.
- **Cryptography**: Under cryptographic assumptions such as **indistinguishable obfuscation** (iO) [1, 4] and **Fiat-Shamir** [2], the computational variant of End-of-a-Line is hard.

PPAD is a function problem, and it sits in a whole hierarchy of other fucntion problem complexity classes:



where the other complexity classes are

- FNP (Function Non-deterministic Polynomial): the function problem analogue of NP

- TFNP (Total Function Non-deterministic Polynomial): The subclass of FNP that is total, i.e., a solution is guaranteed to exist
- PPA (Polynomial Parity Argument: Class of total problems where the solution is guaranteed to exist via a parity argument
- PPP (Polynomial Pigeonhole Principle): Class of total problems where the solution is guaranteed to exist by the Pigeonhole Principle
- PLS (Polynomial Local Search): Class modelling problems where the goal is to find a local optima
- CLS, CCLS: continuous local search and convex continuous local search
- FP (Function Polynomial): The function problem analogue of P

4 Nash Equilibrium is PPAD-complete

Using our new notion of reduction, we can now prove a new conditional hardness result:

Theorem 4.1 ([3, 5])**.** *Nash Equilibrium is* PPAD*-complete.*

Proof. Our plan for proving that the Nash Equilibrium problem is PPAD-complete is as follows:

1. **End-of-a-Line to Brouwer**: As we've seen previously, we can reduce the End-of-a-Line problem to the Brouwer problem of finding a fixed poitn. This shows that the Brouwer problem is PPAD-complete.

   Remark. In order to define a computational variant of the Brouwer problem, we need to succinctly describe the continuous function f. It turns out that the right way to do it is to encode it as a deep ReLU network.

2. **Brouwer to Imitation Game**: As in our previous lectures, we then want to reduce Brouwer fixed point problem to the **Imitiation Game** where players Alice, and Bob have utilities

   $$U^A(x; y) = -||x - y||_2^2$$
   $$U^B(y; x) = -||f(x) - y||_x^2$$

   As we've seen before, the unique Nash equilibrium of this game is when $y = x = f(y)$. Unfortunately, we now have a problem: both Alice and Bob have infinite actions! In order to fix this problem, we can discretize this space, but after discretization both players will have exponentially many actions (roughly $(1/\epsilon)^n$ actions). So we can't immediately reduce in this way. Instead, we first reduce to a similar game with $2n$ players.

3. **Brouwer to $2n$-player Nash**: We now have n Alices and n Bobs with utility functions

   $$U^{A_i}(x_i; y_i) = -(x_i - y_i)^2$$
   $$U^{B_j}(y_j; x) = -(y_j - f_j(x))^2$$

   (Where $f_j(\cdot)$ is the j-th component of $f(\cdot)$.)

   Intuitively, we consider the dimensions separately in order to reduce the actions per player to a constant number of actions. (In fact, this can be further reduced to 2 actions per player.)

4. **2n players to 2 players**: We reduce the $2n$-player game to a 2-player game using the **Lawyers' Game**: We can think of Alice as the lawyer for all the players A_i and Bob as the lawyer for all the players B_j, and the lawyers pick moves for each of their players. So Alice and Bob's actions are

$$\text{Alice's actions: } (i \in [n]) \times (\text{Action for player } A_i)$$
$$\text{Bob's actions: } (j \in [n]) \times (\text{Action for player } B_j)$$

Then we can define Alice's utility as A_i's utility given B_j's actions:

$$U^{A_i}(x_i; y_i) = \begin{cases} -(x_i - y_i)^2 & \text{if } i = j \\ 0 & \text{othewise} \end{cases}$$

For now, it is helpful of Bob's players' utiltiy functions as defined analogously. (That doesn't quite work. We'll revisit this in the cheats section.)

5. **Forcing a large support**: Continuing on, we note that for this to be a hard problem, we need to force Alice and Bob to use all their players. Specifically, we want Alice and Bob to choose i, j approximately uniformly at random. We do this via the **Generalized Matching Pennies** or **Hide-and-Seek** game. In this game, Alice is inentivizes to choose the same index as Bob ($i = j$) whereas Bob tries to "hide". Formally:

$$U^A_{MP}(i, j) = \begin{cases} M & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} = -U^B_{MP}(j, i)$$

(Where M is a sufficiently large number .)

The bonus forces Alice and bob to pick i, j (approximately) uniformly at random since if Bob favored a player i more than uniform, Alice could respond by playing their player i more often. (And vice versa if Alice places less probability on player i.) The unique Nash equilibrium of this game is when both players play approximately uniformly at random.

6. **Analyzing the Lawyers' Game** Now we can describe Alice's expected utility as approximately the average of the utilities of the player's she's a lawyer for:

$$\text{Alice's Expected Utility} \approx \frac{1}{n} \sum_{i=1}^{n} A_i$$

Claim 4.2. *If (X, Y) is a Nash equilibrium in the 2-player game, we can extract a (x_i, y_i) Nash equilibrium for the $2n$-player game, where x_i is the marginal distribution of X on A_i's actions.*

Proof. Suppose that player A_i has a profitable deviation, then Alice would also have a profitable deviation, contradiction. □

□

5 Cheats

5.1 Bob's Utility Function

In order to define Bob's utility function, we would like to construct something analogous to Alice's utility function. The analogous utility function would be

$$U^{B_j}(y_j; x_i) = \begin{cases} -(f_j(x) - y_j)^2 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

This is actually enough to prove that finding a Nash equilibrium in a $2n$-player game is PPAD-hard.

But in order for the Lawyers' Game reduction from $2n$ players to 2 players to go through, we need like to create a **polymatrix game**, where player A_i's utility is additvely separable as a function of B_j's actions. In other words, every A_i and B_j are playing a separate two-player game. Unfortunately, here we run into a problem: $f_j(x)$ cannot be computed from x_j!

5.2 How to use the circuits S, P

In the definition of End-of-a-Line, we have two circuits S, P which we haven't touched in the above proof. There is no way we can reduce from End-of-a-Line without addressing the circuits! This means that obtaining polymatrix $2n$-players games as discussed above requires fundamentally new ideas.

The most important missing ideas is to introduce additional players that play in **gate-gadget games**. Each gate-gadget game has an output player and one or two input players. We set the utilities so that at Nash equilibrium the output player plays an action that corresponds to the gate applied to the actions chosen by the input players. When we combine all these gates together, the players implement effectively the circuits S, P. In the next lecture, we'll see a simpler example of gate gadget for a different total problem.

References

[1] Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a nash equilibrium. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1480–1498, 2015.

[2] Arka Rai Choudhuri, Pavel Hubacek, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. Finding a nash equilibrium is no easier than breaking fiat-shamir. In *STOC*, 2019.

[3] Christos H. Papadimitriou Constantinos Daskalakis, Paul W. Goldberg. The complexity of computing a nash equilibrium. *SIAM J. Comput. 39(1), 195-259*, 2009.

[4] Sanjam Garg, Omkant Pandey, and Akshayaram Srinivasan. Revisiting the cryptographic hardness of finding a nash equilibrium. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 579–604, 2016.

[5] Shang-Hua Teng Xi Chen, Xiaotie Deng. Settling the complexity of computing two-player nash equilibria. *Journal of the ACM, Volume 56(3), Aricle No. 14*, 2009.