Algorithmic Game Theory	Lecture Date: September 20, 2011

Lecture: Complexity of Finding a Nash Equilibrium

Lecturer: Christos Papadimitriou *Scribe: Miklos Racz, Yan Yang*

In this lecture we are going to talk about the complexity of finding a Nash Equilibrium. In particular, the class of problems **NP** is introduced and several important subclasses are identified. Most importantly, we are going to prove that finding a Nash equilibrium is **PPAD**-complete (defined in Section 2). For an expository article on the topic, see [4], and for a more detailed account, see [5].

1 Computational complexity

For us the best way to think about computational complexity is that it is about *search problems*. The two most important classes of search problems are **P** and **NP**.

1.1 The complexity class NP

NP stands for non-deterministic polynomial. It is a class of problems that are at the core of complexity theory. The classical definition is in terms of yes-no problems; here, we are concerned with the search problem form of the definition.

Definition 1 (Complexity class **NP**). *The class of all search problems. A* search problem *A is a binary predicate $A(x, y)$ that is efficiently (in polynomial time) computable and balanced (the length of x and y do not differ exponentially). Intuitively, x is an instance of the problem and y is a solution. The search problem for A is this:*

"Given x, find y such that $A(x, y)$, or if no such y exists, say "no"."

The class of all search problems is called **NP**. Examples of **NP** problems include the following two.

- SAT = SAT(φ, x): given a Boolean formula φ in conjunctive normal form (CNF), find a truth assignment x which satisfies φ, or say "no" if none exists.
- NASH = NASH$(G, (x, y))$: given a game G, find mixed strategies (x, y) such that (x, y) is a Nash equilibrium of G, or say "no" if none exists. NASH is in **NP**, since for a given set of mixed strategies, one can always efficiently check if the conditions of a Nash equilibrium hold or not.

Note, however, that a big difference between SAT and NASH is that we know that the latter *always* has a solution, by Nash's theorem [8].

1.2 Completeness and reduction

SAT is actually **NP**-complete, which informally means that it is a "the hardest search problem". The concept of *reduction* makes this notion precise, and helps us to see the relations of different problems in terms of complexity.

Definition 2 (Reduction)**.** *We say problem A reduces to problem B if there exist two functions f and g mapping strings to strings such that*

- *f and g are efficiently computable functions, i.e. in polynomial time in the length of the input string;*
- *if x is an instance of A, then $f(x)$ is an instance of B such that:*
  - *x is a "no" instance for problem A if and only if $f(x)$ is a "no" instance for problem B;*
  - *$B(f(x), y) \Rightarrow A(x, g(y))$.*

The point is that we indirectly solve problem A by reducing it to problem B: we take an instance of A, compute $f(x)$, which we feed into our algorithm for problem B, and finally, we transform the solution for problem B back to a solution for problem A, using function g. Thus if A reduces to B, then B is at least as hard as A.

Definition 3 (**NP**-completeness)**.** *A problem in* **NP** *is* **NP***-complete if all problems in* **NP** *reduce to it.*

It can be shown that SAT is **NP**-complete. However, nobody believes that NASH is **NP**-complete. The reason for this is that NASH always has a solution, which we know from Nash's theorem, and **NP**-complete problems draw their hardness from the possibility that a solution might not exist. For problems that are **NP**-complete, the "yes" instances have certificates (solutions verifiable in polynomial time), but the "no" instances (most likely) do not. It is very unlikely that there is a reduction from SAT to NASH, since if we have an instance of SAT which is unsatisfiable, the reduction gives us an instance of NASH such that some solution to this NASH instance implies that the SAT instance has no solution.

Despite the fact that NASH is very unlikely to be **NP**-complete, there are many variants of NASH (in which the existence obstacle is not there) that are indeed **NP**-complete. For example, if one seeks a Nash equilibrium that maximizes the sum of player utilities, or one that uses a given strategy with positive probability, then the problem becomes **NP**-complete [6, 3]. Another variant is 2NASH: given a game and a Nash equilibrium, find another one, or output "no" if none exist. Here there is no existence theorem, and indeed, we have the following result.

Lemma 1.1. 2NASH *is* **NP***-complete.*

Proof. To prove **NP**-completeness of 2NASH, we provide a reduction from 3SAT to 2NASH. (Every instance of 3SAT is in CNF, with three literals per clause.) Let φ be an instance of 3SAT consisting of n literals $\{x_i\}_{i=1}^n$ and m clauses $\{c_j\}_{j=1}^m$. We then construct a two player symmetric game, where each player has $2n+m+1$ possible strategies: two for each literal (denoted by $\{\pm x_i\}_{i=1}^n$), one for each clause (denoted by $\{c_j\}_{j=1}^m$), and one extra strategy, called d (for default). To specify the game, it remains to define the payoffs.

We start by specifying the payoffs when at least one of the players plays d. If they both play d, then both of them get a payoff of 6. If one of them plays d and the other plays a clause c_j, $j \in [m]$, then the player playing d gets a payoff of 5, while the other player gets 0. If one of them plays d and the other plays a literal λ, then both players get a payoff of 0. Consequently it is clear that (d, d) is a pure Nash equilibrium. We now construct the rest of the payoffs such that there exists another Nash equilibrium if and only if there exists a truth assignment x that satisfies the formula φ.

We first define two generalizations of the well-known *rock-paper-scissors* game:

- GRPS$_n$: generalized rock-paper-scissors on n objects. Given n objects (n odd), we arrange them in a cycle, which defines the "beating order". Each object loses to its left neighbor and beats its right neighbor (with payoffs 1 and -1 for the winner and loser, respectively); playing any other pair of objects results in a tie, giving a payoff of 0 to both players. There is only one mixed Nash equilibrium, randomizing uniformly among all n objects, giving each of them a weight of $1/n$.

- BWGRPS_n: black and white generalized rock-paper-scissors on n objects. We now have two colored versions of each object: each one is available in black and white. If the players play two different objects, then the payoff is defined exactly according to GRPS_n, disregarding the colors. However, if both players play the same object but with different colors, then both of them get executed. (Less drastically: both receive a huge negative payoff.) Finally if both play the same object and the same color, then there is a tie with payoff 0. So in any Nash equilibrium the two players first agree at the beginning on a coloring for each object, and then randomize uniformly among all n objects with this specific coloring.

Now in the $2n \times 2n$ submatrix of the payoff matrix where both players play a literal, let the payoff matrix be given by BWGRPS_n—a color assignment of the objects corresponds to a truth assignment of the literals—but at the end let us give each player a payoff of 2 (this can be done by shifting the payoffs of the game from 1, -1 to 3, 1). If the payoff matrix were just this submatrix, then a Nash equilibrium would correspond to a truth assignment and both players randomizing uniformly with weight $1/n$ among these literals. The average payoff for both players would be 2.

If both players play a clause, then let them both receive a payoff of 2.

Finally, let us define the payoffs when one player plays a literal, and the other plays a clause. We define the payoffs symmetrically, so let us assume that player 1 plays literal λ and player 2 plays clause c_j. The payoffs are $(0, n)$ if $\bar{\lambda} \in c_j$ (i.e. if λ falsifies c_j), and $(0, 0)$ otherwise. Since each literal is played with weight $1/n$ by player 1 (if played at all), player 2 receives an extra average payoff of $\frac{1}{n} \times n = 1$ for each literal that falsifies the clause. So if all literals are falsified in a given clause, then player 2 gets an average payoff of 3 playing this clause, which is more than what he/she gets by playing a literal. Consequently, if the truth assignment chosen by the players before playing the BWGRPS_n game does not satisfy the original Boolean formula φ, then randomizing uniformly on literals of this truth assignment does not give a Nash equilibrium, since player 2 has an incentive to play a clause. However, if the chosen truth assignment does satisfy φ, then neither player has an incentive to deviate, and we have a mixed Nash equilibrium, which immediately gives us a truth assignment satisfying our formula φ.

Note that every Nash equilibrium other than the pure equilibrium (d, d) must have its support in the $2n \times 2n$ submatrix of the payoff matrix where both players play a literal, since if e.g. player 2 plays a clause or d, then player 1 has an incentive to play d, in which case player 2 has an incentive to play d, thus leading us to the pure equilibria (d, d).

So we have shown that this game has a second Nash equilibrium (other than (d, d)) if and only if φ is satisfiable; moreover, if there exists a second Nash equilibrium and we can find it, then we immediately have a truth assignment satisfying our formula φ. $\square$

We note that this proof can be modified to prove the **NP**-completeness of other variants of NASH: Does the game have at least 3 Nash equilibria? Does it have one in which the payoffs are the same? Does it have one in which the sum of the payoffs is more than 3? How many Nash equilibria are there? (This last one is #**P**-complete.) And so on.

1.3 TFNP, a subclass of NP

Due to the fact that NASH always has a solution, we are interested more generally in the class of search problems for which every instance has a solution. We call this class **TFNP** (which stands for *total function non-deterministic polynomial*). Clearly NASH $\in$ **TFNP** $\subseteq$ **NP**.

Is NASH **TFNP**-complete? Probably not, because **TFNP** probably has no complete problems — intuitively because the class needs to be difined on a more solid basis than an uncheckable universal statement such as "every instance has a solution." However, it is possible to define subclasses of **TFNP** which do have complete problems. Notice that for every problem in **TFNP** there is a corresponding theorem that proves that every instance of the problem has a solution. The idea: subdivide **TFNP** according to the method of proof.

We can distinguish several classes based on the proof. First of all, if the proof is constructive and can be implemented in polynomial time, then the problem lies in **P**. The empirical observation is that for every problem that lies in **TFNP** but has not been proven to lie in **P**, the proof of existence of a solution uses some simple combinatorial lemma that is exponentially nonconstructive. There are essentially four such known arguments:

- "If a graph has an odd degree node then it has another one." This is the *parity argument*, which gives rise to the class **PPA** (polynomial parity argument).
- "If a directed graph has an unbalanced node (a vertex with different in-degree and out-degree), then it has another one." This is the *parity argument for directed graphs*, which gives rise to the class **PPAD**, which is our main focus, as we will see shortly.
- "Every directed acyclic graph has a sink." This gives rise to the class **PLS** (polynomial local search).
- "Any function mapping n elements to $n-1$ elements has at least one collision." This is the *pigeonhole principle*, which gives rise to the class **PPP**.

A remark on how these classes relate to each other. All of these four classes contain **P**, are contained in **TFNP**, and furthermore **PPAD** is contained in both **PPA** and **PPP**.

The question remains: how can we actually define these classes? In the next section we focus specifically on the **PPAD** class.

2 NASH is PPAD-complete

We concentrate on the class **PPAD** because of the following main theorem.

Theorem 2.1. NASH *is* **PPAD**-*complete.*

The rest of this section is organized as follows. First, we define the **PPAD** class and also what it means to be **PPAD**-complete, and then we show that NASH is indeed **PPAD**-complete. Finally, we conclude the section with some philosophical remarks.

2.1 The class PPAD

Informally, **PPAD** is the class of all search problems which always have a solution and whose proof is based on the parity argument for directed graphs. To formulate this in a precise way, we first define the search problem END OF THE LINE.

The setup is the following. We are given a graph G where the in-degree and the out-degree of each node is at most 1. (I.e. there are four kinds of nodes: sources, sinks, midnodes, and isolated vertices.) It turns out such graphs capture the full power of the argument. Our graph G is exponential in size, since otherwise we would be able to explore the structure of the graph (in particular, we can identify sources and sinks) efficiently; to be specific, suppose G has 2^n vertices, one for every bit string of length n. How do we specify this graph?

Since the graph is exponential in size, we cannot list all edges. Instead, the edges of G will be represented by two Boolean circuits, of size polynomial in n, each with n input bits and n output bits. The circuits are denoted P and S (for potential predecessor and potential successor). There is a directed edge from vertex u to vertex v if and only if $v = S(u)$ and $u = P(v)$, i.e. given input u, S outputs v and, vice-versa, given input v, P outputs u. Also, we assume that the specific vector $00 \cdots 0$ has no predecessor (the circuit P is so wired that $P(0^n) = 0^n$). The search problem END OF THE LINE is the following:

"Given (S, P), find a sink or another source."

Clearly END OF THE LINE $\in$ **NP**, since it is easy to check if a given node is a source or a sink. Moreover, END OF THE LINE is a total problem (i.e. it always has a solution), so END OF THE LINE $\in$ **TFNP**. We are now ready to define the class **PPAD**.

Definition 4 (The class **PPAD**)**.** *The class* **PPAD** *contains all search problems in* **TFNP** *that reduce to* END OF THE LINE*.*

Note that the phrase "in **TFNP**" is unnecessary in the definition, since every problem in **NP** that reduces to END OF THE LINE is in **TFNP**, because reducing means that this argument can prove existence for you.

2.2 Nash $\in$ PPAD

Our first task is to show that Nash is in **PPAD**, i.e. that Nash reduces to END OF THE LINE. But we had done just this in the last lecture when we proved Nash's theorem. We showed that Nash's theorem is implied by Brouwer's theorem, which in turn is implied by Sperner's lemma, which follows from the parity argument for directed graphs. Consequently, if we define the search problems Brouwer and Sperner naturally, then Nash reduces to Brouwer, which reduces to Sperner, which reduces to END OF THE LINE.

2.3 Nash is PPAD-complete

We now show that Nash is in fact **PPAD**-complete, i.e. that END OF THE LINE reduces to Nash. We do this via a series of reductions: we first reduce END OF THE LINE to a 3-dimensional discrete variant of Brouwer, then reduce this to Nash for separable network games (Nash for sng), and finally reduce this to Nash for two player games. Pictorially:

$$\begin{array}{c} \text{END OF THE LINE} \\ \downarrow \\ \text{3D DISCRETE BROUWER} \\ \downarrow \\ \text{NASH FOR SNG} \\ \downarrow \\ \text{NASH FOR TWO PLAYER GAMES} \end{array}$$

We now sketch these reductions, similarly to [4]; for complete proofs we refer to [5].

2.3.1 Brouwer is PPAD-complete

We now give the main idea of reducing END OF THE LINE to (a 3-dimensional discrete version of) Brouwer. We are given an instance of END OF THE LINE, i.e. a pair of circuits (S, P) that define the graph G on 2^n vertices. Our goal is to encode this graph in terms of a continuous, easy-to-compute Brouwer function F.

The domain of F will be the 3-dimensional unit cube, and the behavior of F will be defined in terms of its behavior on a very fine rectilinear mesh of grid points in the cube. Each grid point lies at the center of a tiny "cubelet", and away from these grid points the behavior of F is determined by interpolation.

Each grid point x receives one of 4 "colors", $\{0, 1, 2, 3\}$, that represent the value of the 3-dimensional displacement vectors $F(x) - x : (-1, -1, -1), (1, 0, 0), (0, 1, 0)$, or $(0, 0, 1)$. Our task is to find a "panchromatic cubelet," and that would be close to a fixpoint.

We are now ready to encode our graph G into such a Brouwer function F. We represent every vertex u as two small segments on two edges of the cube, whose endpoints are denoted by u_1, u_1', and u_2, u_2', respectively—see Figure 1. If G has an edge from vertex u to vertex v, then we can create a long path in the 3-dimensional unit cube, going from u_1 to u_1', then to u_2 and u_2', and finally to v_1 and v_1', as indicated in Figure 1. (We can choose the rectilinear mesh fine enough so that these lines do not cross or come close to each other.) Now the coloring is done in such a way that most grid points get color 0, but for every edge from vertex u to vertex v in G, the corresponding long path in the unit cube connects grid points that get colors 1,2, and 3. Moreover, there is a circuit F that computes the displacement (color) of each center of a cubelet, based only of the location of the cubelet. F looks at the location and determines, making calls to circuits S and P, whether the cublet lies on the path, and if so, in which direction does the path traverse it, and if this is the case it outputs the correct color among $1, 2, 3$. Otherwise (and if the cubelet does not lie on one of the three faces adjacent to the origin), then its color/displacement is 0. Importantly, *all 4 colors are adjacent to each other (giving an approximate fixed point) only at sites that correspond to an "end of the line" of G.*

2.3.2 From Brouwer to Nash

We now sketch how to reduce Brouwer to Nash: we have to simulate the Brouwer function with a game.

The **PPAD**-complete class of Brouwer functions that appear above have the property that their function F can be efficiently computed using arithmetic circuits that are built up from standard operators such as addition, multiplication and comparison. The circuits can be written down as a "data flow graph", with one of these operators at every node. The key is to simulate each standard operator with a game, and then compose these games according to the "data flow graph" so that the aggregate game simulates the Brouwer function, and a Nash equilibrium of the game corresponds to a(n approximate) fixed point of the Brouwer function.

We now demonstrate how to simulate addition with a game. Consider the graph in Figure 2, which shows which players affect other players' payoffs. Each of the four players will have two possible strategies, 0 and 1. The payoffs are defined as follows. If W plays strategy 0, i.e. plays to the left (denoted by L), then the payoff matrix for W based on what players X and Y play is given by

	0	1
0	0	1
1	1	2

If W plays strategy 1, i.e. plays to the right (denoted by R), then W gets a payoff of 0 if Z plays L, and a payoff of 1 if Z plays R. Player Z gets a payoff of 1 if he plays L and also W plays L, otherwise he gets a payoff of 0—Z wants W to "look the other way".

In this game the mixed strategy of every player is just a number as well: x is the probability that player X plays strategy 1, y is the probability that player Y plays strategy 1, etc. It turns out that in Nash equilibrium we have $z = \min\{1, x + y\}$, which means that we can do arithmetic.

We can simulate other standard operators such as subtraction, multiplication and comparison, via similar games. Then we can put these basic games together into a big game where any Nash equilibrium corresponds

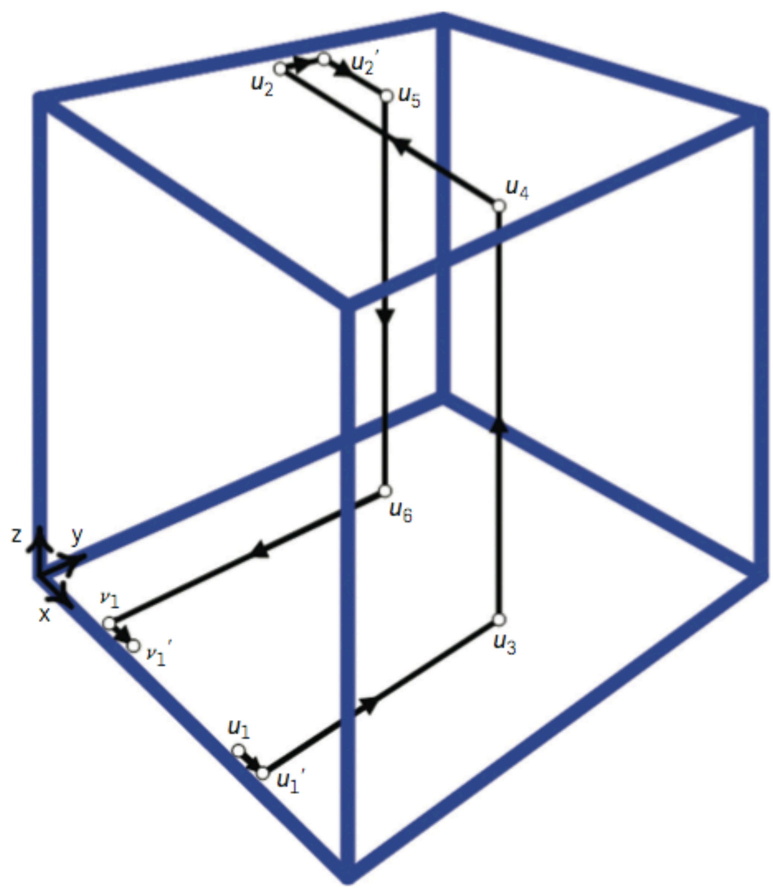

Figure 1: Embedding the END OF THE LINE graph in a cube. Figure from [4].

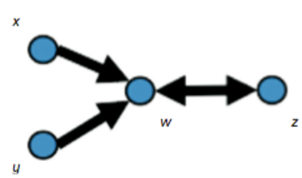

Figure 2: The four players of the addition game, and the graph showing which players affect other players' payoffs. Figure from [4].

to a fixed point of the Brouwer function. However, there is one catch: our comparison simulator is "brittle" in the sense that it can only deal with strict inequality—if the inputs are equal then it outputs anything. This is a problem because our computation of F is now faulty on inputs that cause the circuit to make a comparison of equal values. We can overcome this problem by computing the Brouwer function at a grid of many points near the point of interest, and averaging the results. This makes the computation of F "robust", but introduces a small error in the computation of F. So in fact any Nash equilibrium we find corresponds to an *approximate* fixed point of the Brouwer function.

So far we have shown that the Nash equilibrium problem for network games is $PPAD$-complete. We now show that it is possible to simulate this game with a two-player game, as follows.

We color the players (the vertices of the graph) by two colors, red and blue, so that no two players who play together have the same color. (This can be done, because the little arithmetic games that compose the graph have this structure: input/output nodes who play against internal nodes, and not between them.) Assume that there are n red and n blue players (we may add a few players who do nothing, and n is odd). How can we simulate this game with two players? The idea is to have two "lawyers", Red and Blue, who correspond to the red and blue players, their "clients". The lawyers' task is to then deal with the disputes (described by games) between people from the two sides. Since only players with different colors have disputes the lawyers can deal with the disputes amongst themselves. Since all clients have 2 possible actions, the lawyers have $2m$ possible actions. We just need to define the utilities. We define them in the natural way, e.g. $u_R(1_{r_3}, 0_{b_5}) = u_{r_3}(1, 0_{b_5})$ if players r_3 and b_5 play a game, and 0 otherwise.

The only problem is that we have to convince the lawyers to be fair, to devote equal amount of time (probability mass, that is) to each client, i.e. to play every game with probability $1/n$. To do this, we have the lawyers play, on the side, a high stakes GRPS_n game. "On the side" means that the huge payoffs of the new game are added to the payoffs of the original lawyers' game, depending on which client each lawyer chooses (that is, the objects are the identities of the clients). This allows us to make the lawyers arbitrarily close to fair. This is made precise in the following lemma.

Lemma 2.2. *If (x, y) is an equilibrium in the lawyers' game from above, then*

$$\left| x_{r_i,0} + x_{r_i,1} - \frac{1}{n} \right| \leq \frac{u_{\max} n^2}{M} \qquad \forall i \in [n],$$

where M is the payoff of the high stakes $GRPS_n$, and $u_{\max}$ is the largest (in absolute value) payoff in the original game.

So by making M large enough, we can force the lawyers to be as fair as we want.

Therefore if we can solve the lawyers' game exactly, then we can get an approximate equilibrium in the original game. Note that we need some kind of "relaxation" like this in the reduction, since for two player games we know (e.g. from the Lemke-Howson algorithm) that any Nash equilibrium is rational (i.e. has rational probabilities in the mixed strategies), but for more players we can have irrational numbers in a mixed strategy of a Nash equilibrium.

2.4 Philosophical remarks

To cut a long story short, in this section we have shown that NASH is *intractable* in the refined sense defined above. What does this say about Nash equilibrium as a concept of predictive behavior?

Nash's theorem is important because it is a *universality* result: all games have a Nash equilibrium. In this sense it is a credible form of prediction. However, if finding this Nash equilibrium is intractable (i.e. there exist some specialized instances when one cannot find it in any reasonable amount of time), then it loses universality, and therefore loses credibility as a predictor of behavior.

3 Approximate Nash equilibria

If finding a Nash equilibrium is intractable, it makes sense to ask if one can find an approximate Nash equilibrium, and what are the limits to this?

An *approximate Nash equilibrium* informally means that every player has a "low incentive to deviate". More formally, an ε-Nash equilibrium is a profile of mixed strategies where any player can improve his expected payoff by at most ε by switching to another strategy.

The work of several researchers [4, 5, 1, 2] has now ruled out a *fully polynomial-time approximation scheme* for computing approximate Nash equilibria. The main open question is whether there exists a *polynomial-time approximation scheme* (PTAS) for computing approximate Nash equilibria.

We now present a sub-exponential (though super-polynomial) algorithm to compute an approximate Nash equilibrium for two players, which is due to Lipton, Markakis and Mehta [7]. (The algorithm also works for games with a constant number of players.)

Their idea is the following. Suppose you actually knew a Nash equilibrium and were able to sample from it. Playing t games, the sample strategies for the two players are $X_1, X_2, \ldots X_t$, and $Y_1, Y_2, \ldots, Y_t$, respectively. (These samples are unit vectors representing the strategies.) From these t samples, we can then form mixed strategies for the two players, by averaging the samples:

$$X = \frac{1}{t}\left(X_1 + \cdots + X_t\right)$$
$$Y = \frac{1}{t}\left(Y_1 + \cdots + Y_t\right).$$

The following lemma then tells us that the set of mixed strategies (X, Y) is close to a Nash equilibrium.

Lemma 3.1. *Let $t = \frac{8 \log n}{\varepsilon^2}$ and define X and Y as above. Then (X, Y) is an ε-approximate Nash equilibrium with high probability.*

The form of t in the lemma follows from the Chernoff bound.

However, there is a catch to this procedure: this does not tell us how to find this approximate Nash equilibrium, since we are not able to sample from the Nash equilibrium. (In that case we would already be done.) The solution to this problem is based on the observation that any such pair of mixed strategies consists of vectors where the entries have denominators equal to t. So we can simply go through all possible vectors with denominators equal to t, and we will find an ε-Nash equilibrium. This gives us an $O\left(n^{\log n/\varepsilon^2}\right)$ algorithm for finding an ε-Nash equilibrium for two players.

References

[1] X. Chen and X. Deng. Settling the Complexity of 2-Player Nash Equilibrium. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society, 2006.

[2] X. Chen, X. Deng, and S.H. Teng. Computing Nash Equilibria: Approximation and Smoothed Complexity. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society, 2006.

[3] V. Conitzer and T. Sandholm. Complexity Results about Nash Equilibria. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 765–771, 2003.

[4] C. Daskalakis, P.W. Goldberg, and C.H. Papadimitriou. The Complexity of Computing a Nash Equilibrium. *Communications of the ACM*, 52(2):89–97, 2009.

[5] C. Daskalakis, P.W. Goldberg, and C.H. Papadimitriou. The Complexity of Computing a Nash Equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

[6] I. Gilboa and E. Zemel. Nash and Correlated Equilibria: Some Complexity Considerations. *Games and Economic Behavior*, 1(1):80–93, 1989.

[7] R.J. Lipton, E. Markakis, and A. Mehta. Playing Large Games Using Simple Strategies. In *Proceedings of the 4th ACM Conference on Electronic Commerce*, pages 36–41. ACM, 2003.

[8] J. Nash. Non-cooperative games. *The Annals of Mathematics*, 54(2):286–295, 1951.