601.436/636 Algorithmic Game Theory **Lecturer:** Michael Dinitz
Topic: Hardness of Computing Nash Equilibria **Date:** 2/6/20
Scribe: Michael Dinitz

4.1 Introduction

Last class we saw an exponential time algorithm (Lemke-Howson) for computing a Nash equilibrium of a bimatrix game. Today we're going to argue that we can't really expect a polynomial-time algorithm for computing Nash, even for the special case of two-player games (even though for two-player zero-sum we saw that there is such an algorithm). Formally proving this was an enormous accomplishment in complexity theory, which we're not going to do (but might be a cool final project!). Instead, I'm going to try to give a high-level point of view of what the result actually says, and not dive too deeply into how to prove it. Today's lecture is pretty complexity-theoretic, so if you haven't seen much complexity theory before, feel free to jump in with questions.

The main problem we're going to be analyzing is what I'll call Two-Player Nash: given a bimatrix game, compute a Nash equilibrium. Clearly this is a special case of the more general problem of Nash: given a simultaneous one-shot game, compute a Nash equilibrium. So if we can prove that Two-Player Nash is hard, then so is Nash.

4.2 NP and FNP

As we all know, the most common way of proving that a problem is "unlikely" to have a polynomial-time algorithm is to prove that it is NP-hard. Can we do that for computing a Nash? Recall the definition of NP based on verification:

Definition 4.2.1 (NP) *A decision problem Q is in NP if there exists a polynomial time algorithm $V(I, X)$ such that*

1. *If I is a YES instance of Q then there exists some X such that $|X|$ is polynomial in $|I|$ and $V(I, X) = YES$*
2. *If I is a NO instance of Q then $V(I, X) = NO$ for all X*

Note that in this definition Q has to be a decision problem. The obvious way of making a decision problem out of Two-Player Nash is to change the question to be "is there a Nash equilibrium." But the answer to this question is always yes! So it clearly is not NP-hard.

To fix this, one natural approach is to switch to a search version of NP, where for a YES instance we are also required to *find* the solution/witness. This is called *Functional NP* (FNP): not only do witnesses/solutions have to be able to be verified in polynomial time, but our goal is now to *find* such a witness in a YES case (as opposed to NP, where we only had to output YES).

It's easy to see that Two-Player Nash is in FNP. This is because the witness in a YES instance (which is the only type of instance in this case) can be verified in polynomial time: the witness

will be a Nash, and we can check that this is indeed a Nash by checking whether there are any pure-strategy improving deviations (or, equivalently, by using the characterization of best responses from last class).

So we've found a natural complexity class which looks like NP, but contains Two-Player Nash. Is Two-Player Nash FNP-complete? To answer this, we need a notion of reduction between FNP problems (rather than between decision problems). We'll use a straightforward definition.

Definition 4.2.2 *We say that P reduces to Q (denoted as $P \leq_p Q$) if there exist polynomial-time algorithms A and B such that*

1. *A maps instances of P to instances of Q,*
2. *If I is a YES instance of P than $A(I)$ is a YES instance of Q, and*
3. *If X is a witness for $A(I)$, then $B(x)$ is a witness of I (if I is a YES instance) or NO (if I is a NO instance).*

This definition of reduction should feel pretty natural – it is the obvious generalization of a Karp or many-one reduction used to prove reductions among NP problems. Intuitively, if we want to solve instance I of problem P (find a witness for I), we can run A to get an instance $A(I)$ of Q, use some algorithm to find a witness X of $A(I)$ (or NO), and then use B to find a witness $B(X)$ of I. So if we want to solve P, it suffices to solve Q.

All of your reduction about NP and reductions also works for FNP and these reductions. In particular, the problem FSAT (Functional SAT: given an instance of SAT, if it satisfiable then return a satisfying assignment and otherwise return NO) is FNP-complete, for the same basic reason that SAT is NP-complete.

Is it true that Two-Player Nash is FNP-complete under this type of reduction? As discussed, Two-Player Nash is in FNP, so the question is whether it is FNP-hard. Unfortunately, the answer is (likely) no.

Theorem 4.2.3 *If* Two-Player Nash *is FNP-hard then NP = coNP.*

Proof: Recall that NP is the set of problems with short certificates of YES, while coNP is the set of problems with short certificates of NO (which can be checked in polynomial time). So to prove that NP = coNP, we just need to prove that every problem in NP also has short certificates of NO. And to do this, it's enough to show it for an NP-complete problem: SAT. So our job is to find such a certificate of NO instances for SAT.

Suppose that Two-Player Nash is FNP-hard. Then there is a reduction from FSAT to Two-Player Nash. This means that there are two algorithms A and B such that:

1. A maps every SAT formula ϕ to some bimatrix game $A(\phi)$, and
2. B maps every Nash equilibrium (x, y) of $A(\phi)$ to either a satisfying assignment of ϕ (if ϕ is satisfiable) of NO (if ϕ unsatisfiable).

We can use these algorithms to give NO certificates. Suppose that ϕ is a NO instance. Then the certificate will be any Nash equilibrium (x, y) of $A(\phi)$. Such a certificate can be verified by checking that (x, y) is a Nash equilibrium, and then checking that $B(x, y)$ is NO. ■

Since most complexity theorists think that NP $\neq$ coNP, this is strong evidence that Two-Player Nash is not FNP-hard.

Intuitively, in retrospect this should not be surprising. We defined FNP because we wanted a class of search problems rather than decision problems, but at the end of the day it's still "basically" NP, where the difficulty comes from distinguishing YES from NO instances. So it shouldn't be surprising that a problem like Two-Player Nash (where every instance is YES) is not the hardest problem.

One possible approach to this is to define a new complexity class consisting of the subset of FNP consisting of problems where every instance is YES. This class is called TFNP (*Total* Functional NP). Unfortunately, there are reasons to suspect that TFNP does not have *any* complete problems (I won't go into these reasons, but there's a short discussion about this based on syntactic vs semantic complexity classes in Section 20.2.3 of Roughgarden's book).

So instead we need a different class which *does* have complete problems and where the difficulty is from search. Defining this class, and showing how Two-Player Nash fits into it, is what we'll spend the rest of the lecture on.

4.3 PPAD

To get some intuition, let's think about the argument for the existence of Nash equilibria: not Nash's original proof, but rather the proof implied by the Lemke-Howson algorithm. Recall that after all of the analysis we did of Lemke-Howson, we could interpret it as also proving the existence of a Nash based on whatever vertex of the graph H_k is at the other end of the path that has $(0, 0)$ as one endpoint.

Let's think a little more closely about the properties of the Lemke-Howson argument. I claim that when viewed more abstractly, Lemke-Howson guarantees the existence of a Nash equilibrium (and an exponential time algorithm for finding it) because:

1. There is a finite (but possibly exponential) graph.
2. Every vertex of the graph has degree at most 2.
3. There is one known source (vertex of degree 1), known as the "standard source" (in our case, $(0, 0)$).
4. Every source other than the standard source is a valid solution.
5. Given a vertex, it is computationally easy (polynomial time) to determine its neighbors in the graph.

In other words, *any* problem that has these properties has at least one solution (whatever source is at the other end of the path starting at the standard source) and a (possibly exponential) time

algorithm to find it (follow the path from the standard source). Note that it was not at all obvious that Two-Player Nash had this structure – we had to do a lot of work last lecture to show that it does!

We can define a complexity class based on these properties (this was originally done by Christos Papadimitriou in 1994):

Definition 4.3.1 (PPAD, informal) *PPAD is the class of search problems where the existence of a solution and an algorithm to find one are guaranteed by the above properties.*

PPAD stands for "Polynomial Parity Argument (Directed)". Don't worry about what this means or why. Just note that it's about *parity*. As we discussed at the end of last lecture, a corollary of Lemke-Howson is that there are an odd number of Nash equilibria. Any problem in PPAD clearly shares this property.

Another property/question that arises from PPAD is algorithmic. As mentioned, every problem in PPAD has an exponential-time algorithm. Of course, we're interested in the obvious question: does every problem in PPAD have a polynomial-time algorithm? One interesting way of thinking of this is that it's a question of "path jumping". We know we can find the start of a path where the endpoint is a solution. Can we "jump" along this path to get to the end faster than the length of the path (or find the start of a different path)?

Clearly Lemke-Howson implies that Two-Player Nash is in PPAD. Interestingly, there are many other problems which are known to be in PPAD (which is one of the reasons that Papadimitriou defined it). Most notably, a discrete version of Brouwer's fixed point theorem (known as Sperner's Lemma) is also in PPAD. Recall that Nash's original proof that every game has a Nash equilibrium is based on using ("reducing to") Brouwer's theorem. So maybe it's not so surprising that Brouwer's is also in PPAD.

A seminal result in algorithmic game theory is the following:

Theorem 4.3.2 Two-Player Nash *is PPAD-complete.*

I'm not even going to talk about how to prove this – it's incredibly complicated. But note what it means: every problem in PPAD can be reduced to Two-Player Nash, i.e., Two-Player Nash is "the hardest" problem in PPAD. In particular, Brouwer can be reduced to Two-Player Nash! So while Nash's theorem is a reduction *to* Brouwer, there is also a reduction from Brouwer to Nash. This is pretty surprising (at least to me), and shows that computing a Nash is actually a really hard problem.