CS364A: Algorithmic Game Theory
Lecture #20: Mixed Nash Equilibria and PPAD-Completeness*

Tim Roughgarden†

December 4, 2013

Today we continue our study of the limitations of learning dynamics and polynomial-time algorithms for converging to and computing equilibria. Recall that we have sweeping positive results for coarse correlated and correlated equilibria, which are tractable in arbitrary games. We have only partial positive results for pure Nash equilibria of routing and congestion games, and last lecture we developed the theory of PLS-completeness to explain our limited success. In this lecture we focus on mixed Nash equilibria (MNE). Our positive results so far have been limited to the special case of two-player, zero-sum games (Lecture 18). This lecture develops theory that suggests that there cannot be significantly more general positive results.

1 The Problem: Computing a MNE of a Bimatrix Game

Formally, we study the problem of computing a MNE of a bimatrix game.[1] The input is two $m \times n$ payoff matrices A and B — one for the row player, one for the column player. In zero-sum games, $B = -A$. The goal is to compute mixed strategies $\hat{\mathbf{x}}, \hat{\mathbf{y}}$ such that

$$\hat{\mathbf{x}}^T A \hat{\mathbf{y}} \geq \mathbf{x}^T A \hat{\mathbf{y}} \tag{1}$$

for all row mixed strategies $\mathbf{x}$ and

$$\hat{\mathbf{x}}^T B \hat{\mathbf{y}} \geq \hat{\mathbf{x}}^T B \mathbf{y} \tag{2}$$

for all column mixed strategies $\mathbf{y}$.

†Department of Computer Science, Stanford University, 462 Gates Building, 353 Serra Mall, Stanford, CA 94305. Email: tim@cs.stanford.edu.

[1] An alternative generalization of two-player zero-sum games is three-player zero-sum games. Such games include two-player non-zero-sum games as a special case — do you see why?

There is no known polynomial-time algorithm for computing a MNE of a bimatrix game, and many smart people of tried to come up with one. This lecture develops the relevant complexity theory for arguing that the problem may be inherently intractable. The goal is to prove that the problem is complete for a suitable complexity class. But which class? This is a tricky question, and it took years for the leading thinkers in the field to figure out the answer. Before we explain the solution, let's understand why plain old NP-completeness is not the right intractability notion for equilibrium computation. The discussion in the next section also applies to the PLS problems discussed last lecture, thereby justifying our development of PLS-completeness as a weaker analog of NP-completeness.

2 NP Search Problems (FNP)

NP problems are decision problems, where the correct answer to an instance is either "yes" or "no". Equilibrium computation problems are not decision problems; the output should be a bona fide equilibrium. To address this typechecking error, we work with the closely related class FNP, for "functional NP." FNP problems are just like NP problems except that, for "yes" instances, we demand that a solution be produced. These are also call *search* problems.

More formally, an algorithm for an FNP problem takes as input an instance of an NP problem, like a SAT formula or an undirected graph. The responsibility of the algorithm is to output a solution, or "witness," like a satisfying assignment or a Hamiltonian cycle, provided one exists. If there is no solution, the algorithm should output "no." Reductions between search problems are defined as in the last lecture via two polynomial-time algorithms, the first algorithm A mapping instances x of one problem to instances $A(x)$ of another, the second algorithm B mapping solutions of $A(x)$ to solutions to x (and "no" to "no").

Your intuition for NP works fine for FNP. For example, the functional version of SAT is an FNP-complete problem. The proof of Cook's theorem — which effectively constructs the algorithm A above from an arbitrary NP problem to SAT — establishes a bijective correspondence between the witnesses of the given NP verifier for the given instance and the satisfying assignments of the constructed SAT formula. The algorithm B is a straightforward implementation of this correspondence.

The class PLS of local search problems, defined last lecture, is a subset of FNP. The witnesses of a PLS problem are its local optima, and the third algorithm in the PLS problem description acts as an efficient verifier of witnesses. In fact, the third algorithm of a PLS problem does considerably more than is asked of an NP verifier — in addition to certifying local optima, when a solution is not locally optimal, the PLS algorithm does not merely say "no," it offers an alternative solution with superior objective function value.

The problem of computing a MNE of a bimatrix game also belongs to FNP. This assertion amounts to proving that, given mixed strategies $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ in a bimatrix game (A, B), it can be checked efficiently whether or not $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ constitute an MNE of the game. This is not entirely obvious, since the equilibrium conditions (1) and (2) reference an infinite number of mixed strategies. Fortunately, it is enough to check only pure-strategy deviations.

Equivalently, a mixed strategy $\hat{\mathbf{x}}$ by the row player is a best response to a column strategy $\hat{\mathbf{y}}$ if and only if $\hat{\mathbf{x}}$ randomizes only over rows with maximum expected payoff (w.r.t. $\hat{\mathbf{y}}$); see also the Exercises.[2]

3 NP Search Problems with Guaranteed Witnesses ($TFNP$)

Could computing a MNE of bimatrix game be FNP-complete? Being as hard as every problem in FNP would constitute strong evidence of intractability. Intriguingly, FNP-completeness would have astonishing consequences.

Theorem 3.1 ([8]) *If the problem of computing a MNE of bimatrix game is FNP-complete, then $NP = coNP$.*

While $NP = coNP$ doesn't directly imply $P = NP$, it is thought to be an equally unlikely state of affairs. For example, if $NP = coNP$, then there are short, efficiently verifiable proofs for the $coNP$-complete UNSAT problem. Convincing someone that a SAT formula is satisfiable is easy enough — just exhibit a satisfying assignment — but how would you quickly convince someone that none of the exponentially many truth assignments are satisfying? Most researchers believe that there is no way to do it — that $NP \neq coNP$. If this is indeed the case, then Theorem 3.1 implies that the problem of computing a MNE of a bimatrix game is not FNP-complete.

Proof of Theorem 3.1: The proof is a bit of a mind-binder, but it is not long. Assume there is a reduction, in the same sense of the PLS reductions described last lecture, from the functional SAT problem to the problem of computing a MNE of a bimatrix game. By the definition of a reduction, there exist the following two algorithms:

1. A polynomial-time algorithm A that maps every SAT formula φ to a bimatrix game $A(\varphi)$.

2. A polynomial-time algorithm B that maps every MNE $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ of a game $A(\varphi)$ to:

   (a) a satisfying assignment $B(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ of $A(\varphi)$, if one exists;

   (b) the string "no," otherwise.

We claim that the existence of these algorithms A and B imply that $NP = coNP$. Indeed, consider an arbitrary "no" instance φ of SAT, and an arbitrary MNE $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ of the game $A(\varphi)$.[3] We claim that $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ is a short, efficiently verifiable proof of the unsatisfiability

[2] To be completely rigorous, we also need to argue that there are MNE whose description (in bits) has length polynomial in the input size. This follows from the observation that, given the supports of a MNE — the strategies played with nonzero probability — suitable mixing probabilities can be recovered by solving a linear system of equations. See the Problems for more details.

[3] Crucially, Nash's Theorem ensures that $A(\varphi)$ has at least one MNE. Also, as noted above, every bimatrix game has an MNE whose description length is polynomial in that of the game. We discuss the proof of Nash's Theorem in Section 7.

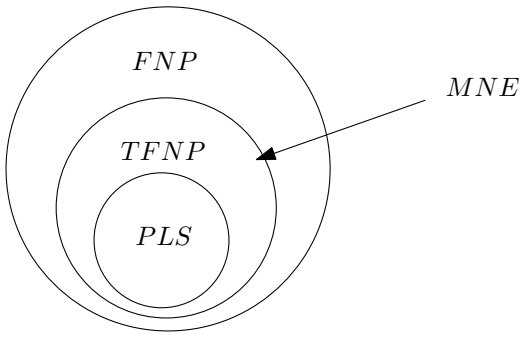

Figure 1: MNE are in the TFNP complexity class.

of φ; this implies $NP = coNP$. Indeed, given an alleged unsatisfiability certificate $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ for φ, one performs two checks: (i) compute the game $A(\varphi)$ using algorithm A and verify that $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ is a MNE of $A(\varphi)$; (ii) use the algorithm B to verify that $B(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ is the string "no." If $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ passes both of these tests, then correctness of the algorithms A and B implies that φ is indeed unsatisfiable. ■

What's really going on in the proof of Theorem 3.1 is a mismatch between an FNP-complete problem like $FSAT$, where an instance may or may not have a witness, and a problem like computing an MNE, where Nash's theorem guarantees at least one (polynomial-length) witness in every instance. While the correct answer to an instance of FSAT might well be "no;" a correct answer to an instance of computing a MNE of a game is always a witness (a MNE). The subset of FNP problems for which every instance has at least one witness is called $TFNP$, for "total functional NP." The proof of Theorem 3.1 shows more generally that if *any* $TFNP$ problem is FNP-complete, then $NP = coNP$. In particular, since every instance of a PLS problem has at least one witness — local search has to stop somewhere, necessarily at a local optimum — PLS is a subset of $TFNP$. Hence no PLS problem, such as computing a PNE of a routing or congestion game, can be FNP-complete unless $NP = coNP$. See Figure 1. This justifies last lecture's development of PLS-completeness, a weaker analog of FNP-completeness tailored for local search problems.

4 Complete Problems: Syntactic vs. Semantic Complexity Classes

Membership in $TFNP$ precludes proving that computing a MNE of a bimatrix game is FNP-complete. The sensible refined goal, then, is to prove that the problem is $TFNP$-complete — as hard as any other problem in $TFNP$. Unfortunately, $TFNP$-completeness is also too ambitious a goal. The reason is that $TFNP$ does not seem to have complete problems. To explain, think about the complexity classes you know about that *do* have complete problems — NP of course, and also P and $PSPACE$. What do these complexity classes have in common? They are "syntactic," meaning that membership can be characterized via

acceptance by some concrete computational model — polynomial-time or polynomial-space Turing machines, for example. In this sense, there is a "generic reason for membership" in these complexity classes.

$TFNP$ has no obvious generic reason for membership, and as such is called a "semantic class."[4] For example, the problem of computing a MNE of a bimatrix game belongs to $TFNP$ because of Nash's theorem guaranteeing the existence of a MNE — essentially, for topological reasons (see Section 7). Another problem in $TFNP$ is factoring — given a positive integer, output its factorization. Here, membership in $TFNP$ has an algebraic or number-theoretic explanation. Can the existence of a MNE and of an integer factorization be regarded as separate instantiations of some "generic" $TFNP$ argument? No one knows the answer.

5 $PPAD$: A Syntactic Subclass of $TFNP$

Papadimitriou [11] proposed a path to progress: identify *subclasses* of $TFNP$ such that:

1. The class contains interesting computational problems not known to be in P (like computing a MNE).

2. The class has complete problems; roughly equivalently, the class has a "syntactic" definition in the form of a generic reason for membership.

We have already seen an example of such a subclass: the complexity class PLS from last lecture. For example, computing a local optimum of a maximum cut instance is a PLS-complete problem that is not known to be polynomial-time solvable. Our definition of PLS is syntactic in that PLS problems are precisely those defined by a particular computational model, as induced by the three polynomial-time algorithms that define such a problem. The generic reason for membership in PLS (and hence $TFNP$) is that local search, as defined by these three given algorithms, is guaranteed to eventually terminate at a local optimum.

The right complexity class to study the computation of MNE in bimatrix games is called $PPAD$.[5] We describe $PPAD$ primarily via analogy with the class PLS, which is similar in spirit but different in details.

We can think of a local search problem as the problem of computing a sink vertex of a directed acyclic graph (Figure 2), where nodes represent feasible solutions and arcs represent improving local moves. Note that the number of nodes in this directed graph is generally exponential in the size of a problem instance — for example, in a maximum cut instance, nodes correspond to cuts of the original graph. The three algorithms that define a PLS problem enable a straightforward path-following algorithm in this directed graph — the first algorithm gives a starting node and third algorithm provides the next arc to traverse from a non-sink node.

[4] There are many other interesting examples, such as $NP \cap coNP$.

[5] Allegedly standing for "polynomial parity argument, directed version" [11].

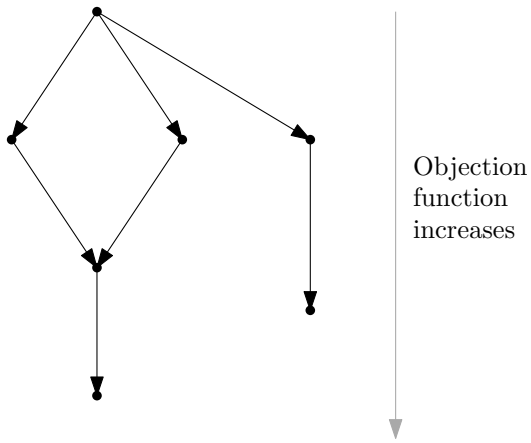

Figure 2: A PLS problem is characterized by a DAG with an objective function.

$PPAD$ problems are those solvable by a particular class of naive directed path-following algorithms — this is the generic reason for membership. In this sense, $PPAD$ problems are similar to PLS problems, which are solvable by following a path through a directed acyclic graph. $PPAD$ problems can be thought of as directed graphs (Figure 3), where nodes again correspond to "intermediate solutions," , and arcs to "next moves." Like with PLS problems, the number of nodes in generally exponential in the parameters of interest. Rather than being directed acyclic, a $PPAD$ directed graph has in- and out-degree at most 1, and at least one source node. Unlike a PLS problem, there is no objective function, and a $PPAD$ directed graph can possess cycles. Like a PLS problem, a $PPAD$ problem is formally specified by three algorithms — this is a "syntactic definition." The first algorithm suggests an initial solution that is a source vertex. The third algorithm either verifies that the current intermediate solution is a witness — by definition, a source or sink vertex different from the starting point — or else returns the next intermediate solution. Naive path-following — invoking the first algorithm once and the third algorithm until the other end of the path is reached — is guaranteed to terminate at a witness.[6] This path might have exponential length, however, so the non-trivial question is whether or not a witness of a $PPAD$ problem can be computed in polynomial time.

What do $PPAD$ problems have to do with computing MNE, or anything else for that matter? To get a feel for this complexity class, we discuss next a canonical example of a $PPAD$ problem.

[6]The primary role of the second algorithm is to keep the third algorithm honest — this is technically necessary so that naive path-following is guaranteed to terminate, no matter what the given three algorithms are. Recall that in a PLS problem, the second algorithm, which computes the objective function value, can be used to verify that the third algorithm suggests a superior solution. If the third algorithm fails to produce a better solution, we just interpret its output as "locally optimal." In a $PPAD$ problem, the second algorithm enforces that the corresponding directed graph has in-degree at most 1. It takes as input an intermediate solution and outputs the previous one. If the third algorithm declares that y should follow x, but the second algorithm does not agree that x should precede y, then we interpret x as a sink (and hence a witness).

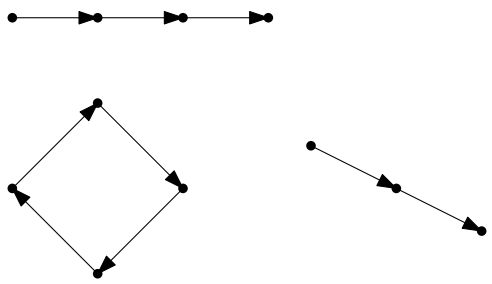

Figure 3: PPAD problems are characterized by a graph where each node has in- and out-degree at most 1.

6 A Canonical $PPAD$ Problem: Sperner's Lemma

This section presents a computational problem that is clearly well matched with the $PPAD$ complexity class. The next section describes its relevance to computing a MNE of a bimatrix game.

Consider a subdivided simplex in the plane; see Figure 4. A *legal coloring* of its vertices colors the top vertex red, the left vertex green, and the right vertex blue. A vertex on the boundary must have one of the two colors of the endpoints of its side. Internal vertices are allowed to possess any of the three colors. A triangle is *trichromatic* if all three colors are represented at its corners.

Sperner's Lemma asserts that for every legal coloring, there is an odd number of trichromatic triangles (and hence at least one). The proof is constructive. Define a graph G that has one node per triangle, plus a source node outside the simplex. The graph G has one edge for each red-green side of a triangle. See Figure 4. Every trichromatic triangle corresponds to a degree-one node of G. Every triangle with one green and two red corners or two green and one red corner corresponds to a node with degree two in G. The source node of G has degree equal to the number of red-green edges on the left side of the simplex, which is an odd number. Because every graph has an even number of nodes of odd degree, G has an odd number of trichromatic triangles, which proves Sperner's Lemma.[7] Moreover, starting from the source vertex, naive path-following is guaranteed to terminate at a trichromatic triangle. Thus, computing a trichromatic triangle of a legally colored subdivided simplex is a $PPAD$ problem.[8]

[7]The same result and proof extend by induction to higher dimensions — every subdivided simplex in $\mathcal{R}^n$ with vertices legally colored with $n+1$ colors has an odd number of panchromatic subsimplices, with a different color on each corner.

[8]We are omitting some details. The graph of a $PPAD$ problem is directed, while the graph G we defined here is undirected. There is, however, a canonical way to direct the edges of this graph G.

Also, the source node of a $PPAD$ problem is supposed to have out-degree 1, while that of the graph G above has some odd degree $2k+1$. This can be corrected by splitting the source node of G into $k+1$ nodes, a source with out-degree 1 and k nodes with in- and out-degree 1.

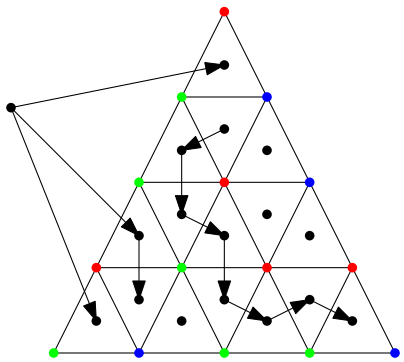

Figure 4: Every triangle with an end point is a trichromatic triangle.

7 MNE and $PPAD$

What does computing a MNE have to do with $PPAD$, the class of FNP problems solvable by a particular form of directed path-following? There are two fundamental connections.

First, Sperner's Lemma turns out to be the combinatorial heart of Nash's proof that every finite game has at least one MNE, and its proof yields a path-following algorithm for computing an approximate MNE (in exponential time). The reduction of Nash's theorem to Sperner's Lemma has two parts. The first part is use Sperner's Lemma to prove Brouwer's fixed-point theorem. Brouwer's fixed-point theorem states that every continuous function f that maps a compact convex subset C of $\mathcal{R}^n$ to itself has at least one fixed point: a point $x \in C$ with $f(x) = x$. All of the hypotheses — that f is continuous, C is bounded, C is closed, and C is convex — are necessary (see the Exercises).

Suppose we want to prove Brouwer's fixed-point theorem when C is a simplex in $\mathcal{R}^2$. Let $f : C \to C$ be continuous. Subdivide C into small triangles. Color a triangle corner green if $f(x)$ is farther from the left corner of C than x; red if $f(x)$ is farther from the top corner of C than x; and blue if x is farther from the right corner of C than x. If two of these conditions apply to x, either corresponding color can be used. (If none of them apply, x is a fixed point and there's nothing left to prove.) This results in a legal coloring of the subdivision. By Sperner's Lemma, there is at least one trichromatic triangle — a triangle whose corners are being pulled in different directions. Taking a sequence of finer and finer subdivisions, we get a sequence of ever-smaller trichromatic triangles. Because C is compact, the centers of these triangles contain a sequence $x^1, x^2, \ldots$, that converges to a point x^* in C. Because f is continuous, in the limit, $f(x^*)$ is at least as far from each of the three corners of C as x^*. This means that x^* is a fixed point of f.[9]

Nash [10] gave an elegant reduction from the existence of MNE in finite games to

[9]Here's the idea for extending the fixed-point theorem to all convex compact subsets of $\mathcal{R}^n$. First, since Sperner's Lemma extends to higher dimensions, so does Brouwer's fixed-point theorem for the special case of simplices (by the same argument). Second, every pair C_1, C_2 of compact convex subsets of equal dimension are homeomorphic — that is, there is a bijection $f : C_1 \to C_2$ with f and f^{-1} continuous — and homeomorphisms preserve fixed-point theorems (see the Exercises).

Brouwer's fixed point theorem. Here is a sketch. Consider a k-player game with strategy sets $S_1, \ldots, S_k$ and payoff functions $\pi_1, \ldots, \pi_k$. The relevant compact convex set is $C = \Delta_1 \times \cdots \times \Delta_k$, where Δ_i is the simplex representing the mixed strategies of player i. We want to define a continuous function $f : C \to C$ — from mixed strategy profiles to mixed strategy profiles — such that fixed-points of f are MNE of the given game. We define f separately for each component $f_i : C \to \Delta_i$. A natural first idea is to set f_i to be a best response of player i to the mixed strategy profiles of the other players. This does not lead to a continuous, or even well defined, function, so instead we use a "regularized" version. We set

$$f_i(x_i, \mathbf{x}_{-i}) = \operatorname*{argmax}_{x_i' \in \Delta_i} g_i(x_i', \mathbf{x}),$$

where

$$g_i(x_i, \mathbf{x}) = \underbrace{\mathbf{E}_{s_i \sim x_i', \mathbf{s}_{-i} \sim \mathbf{x}_{-i}}[\pi(\mathbf{s})]}_{\text{linear in } x_i'} - \underbrace{\|x_i' - x_i\|_2^2}_{\text{strictly convex}}.$$

The first term of the function g_i encourages a best response while the second "penalty term" discourages big changes to i's mixed strategy. Because the function g_i is strictly concave in x_i', f_i is well defined. The function $f = (f_1, \ldots, f_k)$ is continuous (see Exercises). It should be clear that every MNE of the given game is a fixed point of f. For the converse, suppose that $\mathbf{x}$ is not a MNE, with player i able to increase its expected payoff by deviating from x_i and x_i'. A simple computation shows that, for sufficiently small $\epsilon > 0$, $g_i((1-\epsilon)x_i + \epsilon x_i', \mathbf{x}) > g_i(x_i, \mathbf{x})$, and hence $\mathbf{x}$ is not a fixed point of f (see the Exercises).

There is also a second way to prove that computing a MNE of a bimatrix game is a $PPAD$ problem, via the Lemke-Howson algorithm (see [15]). The Lemke-Howson algorithm reduces computing a MNE of a bimatrix game to a path-following problem, much in the way that the simplex algorithm reduces computing an optimal solution of a linear program to following a path of improving edges along the boundary of the feasible region. The biggest difference between the Lemke-Howson algorithm and the simplex method is that the former is not guided by an objective function; all known proofs of its inevitable convergence use parity arguments akin to the one in the proof of Sperner's Lemma, thereby showing that the problem lies in $PPAD$.

The two connections between $PPAD$ and computing a MNE are incomparable. The Lemke-Howson argument applies only to games with two players, but it shows that the problem of computing an *exact* MNE of a bimatrix game belongs to $PPAD$. The path-following algorithm derived from Sperner's Lemma applies to games with any finite number of players, but only shows that the problem of computing an *approximate* MNE is in $PPAD$. In fact, with 3 or more players, the problem of computing an exact MNE of a game appears to be strictly harder than $PPAD$ problems [5].

The upshot of all this is that $PPAD$ is a subclass of $TFNP$ that contains the problem of computing a MNE of a bimatrix game. Moreover, $PPAD$ is syntactically defined, with the generic reason for membership being solvability by a naive path-following argument from a source in a directed graph with all in- and out-degrees at most 1. As such, we expect the class to admit complete problems. In fact, we have finally identified the right complexity

class for building evidence that the problem of computing a MNE of a bimatrix game is computationally intractable: it is a $PPAD$-complete problem.

Theorem 7.1 ([2, 3]) *Computing a MNE of a bimatrix game is a $PPAD$-complete problem.*

Theorem 7.1 is one of the greatest hits of algorithmic game theory. Its proof is far too technical to describe here; for overviews in order of increasing levels of detail, see [13, §4.2], [12, pp. 41–45], and [4].

8 Discussion and Open Questions

One interpretation of Theorem 7.1, which is not without controversy, is that the seeming intractability of the Nash equilibrium concept renders it unsuitable for general-purpose behavioral prediction. If no polynomial-time algorithm can compute a MNE of a game, then we don't expect a bunch of strategic players to find one quickly, either. More generally, in classes of games of interest, polynomial-time tractability of computing an equilibrium can be used as a necessary condition for its predictive plausibility.

Intractability is not necessarily first on the list of the Nash equilibrium's drawbacks. For example, its non-uniqueness already limits its predictive power in many settings. But the novel computational intractability critique in Theorem 7.1 is one that theoretical computer science is particularly well suited to contribute.

If we don't analyze the Nash equilibria of a game, then what should we analyze? Theorem 7.1 suggests shining a brighter spotlight on computationally tractable classes of games and equilibrium concepts. For example, our convergence results for no-regret dynamics motivate identifying properties that hold for all correlated or coarse correlated equilibria.

One natural equilibrium concept whose computational complexity remains poorly understood is ϵ-approximate MNE of bimatrix games. After translating and scaling all player payoffs so that they lie in $[0, 1]$, such an equilibrium is, by definition, a pair of mixed strategies so that neither player can increase its payoff by more than ϵ via a unilateral deviation. It is known that an ϵ-approximate MNE of a bimatrix game can be computed in polynomial time when $\epsilon \approx \frac{1}{3}$ [14] and in quasi-polynomial time when ϵ is an arbitrarily small constant (see [7] and the Problems).[10] An interesting open question is whether or not an ϵ-approximate MNE of a bimatrix game can be computed in polynomial time for arbitrarily small constants ϵ.

Another fundamental question that is poorly understood is: how hard are $PPAD$ problems, anyways? In the absence of an unconditional proof about whether or not $PPAD$ problems are polynomial-time solvable, it is important to relate the assumption that $PPAD \not\subseteq P$ to other complexity assumptions stronger than $P \neq NP$ — for example, to cryptographic assumptions like the existence of one-way functions.

[10] This quasi-polynomial-time algorithm enumerates approximations of all MNE, and in particular can identify an approximate MNE with total payoff close to that of the best MNE. Intriguingly, this harder optimization problem can be connected to the infamous planted clique problem [1, 6, 9].

References

[1] P. Austrin, M. Braverman, and E. Chlamtac. Inapproximability of NP-complete variants of Nash equilibrium. *Theory of Computing*, 9:117–142, 2013.

[2] X. Chen, X. Deng, and S.-H. Teng. Settling the complexity of two-player Nash equilibria. *Journal of the ACM*, 56(3), 2009.

[3] 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.

[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] K. Etessami and M. Yannakakis. On the complexity of Nash equilibria and other fixed points. *SIAM Journal on Computing*, 39(6):2531–2597, 2010.

[6] E. Hazan and R. Krauthgamer. How hard is it to approximate the best Nash equilibrium? *SIAM Journal on Computing*, 40(1):79–91, 2011.

[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 (EC)*, pages 36–41, 2003.

[8] N. Megiddo and C. H. Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 81(2):317–324, 1991.

[9] L. Minder and D. Vilenchik. Small clique detection and approximate Nash equilibria. In *APPROX-RANDOM*, pages 673–685, 2009.

[10] J. F. Nash. Equilibrium points in N-person games. *Proceedings of the National Academy of Science*, 36(1):48–49, 1950.

[11] C. H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994.

[12] C. H. Papadimitriou. The complexity of finding Nash equilibria. In N. Nisan, T. Roughgarden, É. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*, chapter 2, pages 29–51. Cambridge University Press, 2007.

[13] T. Roughgarden. Computing equilibria: A computational complexity perspective. *Economic Theory*, 42(1):193–236, 2010.

[14] H. Tsaknakis and P. G. Spirakis. An optimization approach for approximate Nash equilibria. *Internet Mathematics*, 5(4):365–382, 2008.

[15] B. von Stengel. Equilibrium computation for two-player games in strategic and extensive form. In N. Nisan, T. Roughgarden, É. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*, chapter 3, pages 53–78. Cambridge University Press, 2007.