Lecture 20

Mixed Nash Equilibria and $\mathcal{PPAD}$-Completeness

This lecture continues our study of the limitations of learning dynamics and polynomial-time algorithms for converging to and computing equilibria, with a focus on mixed Nash equilibria (MNE). The theory of $\mathcal{PPAD}$-completeness, which resembles that of $\mathcal{PLS}$-completeness except for certain details, provides evidence that the problem of computing a MNE of a general two-player game is computationally intractable. This suggests that the positive results in Lecture 18 for two-player zero-sum games cannot be extended to a significantly larger class of games.

Section 20.1 formally defines the problem of computing a MNE of a bimatrix game. Section 20.2 explains why $\mathcal{NP}$-completeness is not the right intractability notion for the problem, or for the $\mathcal{PLS}$ problems studied in Lecture 19. Section 20.3 formally defines the complexity class $\mathcal{PPAD}$, the class for which computing a MNE is a complete problem. Section 20.4 describes a canonical $\mathcal{PPAD}$ problem inspired by Sperner's lemma, while Section 20.5 explains why computing a MNE of a bimatrix game is a $\mathcal{PPAD}$ problem. Section 20.6 discusses the ramifications of the $\mathcal{PPAD}$-completeness of this problem.

20.1 Computing a MNE of a Bimatrix Game

A two-player game that is not necessarily zero-sum is called a *bimatrix* game. A bimatrix game can be specified by two $m \times n$ payoff matrices $\mathbf{A}$ and $\mathbf{B}$, one for the row player, one for the column player. In zero-sum games, $\mathbf{B} = -\mathbf{A}$. We consider the problem of computing a mixed Nash equilibrium (MNE) of a bimatrix game, or equivalently mixed strategies $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ over the rows and columns such that

$$\hat{\mathbf{x}}^\top \mathbf{A} \hat{\mathbf{y}} \geq \mathbf{x}^\top \mathbf{A} \hat{\mathbf{y}} \tag{20.1}$$

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

$$\hat{\mathbf{x}}^\top \mathbf{B}\hat{\mathbf{y}} \geq \hat{\mathbf{x}}^\top \mathbf{B}\mathbf{y} \tag{20.2}$$

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

There is no known polynomial-time algorithm for computing a MNE of a bimatrix game, despite significant effort by many experts. This lecture develops the appropriate 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? Before providing the solution in Section 20.3, Section 20.2 explains why plain old $\mathcal{NP}$-completeness is not the right intractability notion for equilibrium computation problems.

20.2 Total $\mathcal{NP}$ Search Problems ($\mathcal{TFNP}$)

20.2.1 $\mathcal{NP}$ Search Problems ($\mathcal{FNP}$)

An $\mathcal{NP}$ problem is defined by a polynomial-time verifier of alleged solutions to a given instance, and the inputs accepted by the verifier are called the *witnesses* for the instance. $\mathcal{NP}$ problems are traditionally defined as decision problems, where the correct answer to an instance is either "yes" or "no," depending on whether or not the instance has at least one witness.

Equilibrium computation problems are not decision problems, as the output should be a bona fide equilibrium. To address this typechecking error, we work with the complexity class $\mathcal{FNP}$, which stands for "functional $\mathcal{NP}$." $\mathcal{FNP}$ problems are just like $\mathcal{NP}$ problems except that, for "yes" instances, a witness must be produced. These are also called *search* problems.

An algorithm for a $\mathcal{FNP}$ problem takes as input an instance of an $\mathcal{NP}$ problem, like an encoding of a logical formula or an undirected graph. The responsibility of the algorithm is to output a witness for the instance, like a satisfying truth assignment or a Hamiltonian cycle, provided one exists. If no witnesses exist for the instance, then the algorithm should output "no." $\mathcal{FP}$ denotes the subclass of $\mathcal{FNP}$ problems that can be solved by a polynomial-time algorithm.

A reduction from one search problem L_1 to another one L_2 is defined as in Section 19.2.3 via two polynomial-time algorithms, the first algorithm $\mathcal{A}_1$ mapping instances x of L_1 to instances $\mathcal{A}_1(x)$ of

L_2, the second algorithm $\mathcal{A}_2$ mapping witnesses of $\mathcal{A}_1(x)$ to witnesses of x (and "no" to "no").[1]

The class $\mathcal{PLS}$ of local search problems, defined in Section 19.2, is a subset of $\mathcal{FNP}$. The witnesses of an instance of a $\mathcal{PLS}$ problem are its local optima, and the third algorithm in the $\mathcal{PLS}$ problem description acts as an efficient verifier of witnesses. In fact, the third algorithm of a $\mathcal{PLS}$ problem does considerably more than is asked of an $\mathcal{NP}$ verifier. When this algorithm is given a solution that is not locally optimal, it does not merely say "no," and instead offers an alternative solution with superior objective function value.

The problem of computing a MNE of a bimatrix game also belongs to $\mathcal{FNP}$. This assertion boils down to an efficient solution to the problem of checking whether or not given mixed strategies $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ of a given bimatrix game $(\mathbf{A}, \mathbf{B})$ constitute a MNE. While the equilibrium conditions (20.1) and (20.2) reference an infinite number of mixed strategies, it is enough to check only the pure-strategy deviations (cf., Exercise 13.1), and this can be done in polynomial time.[2]

20.2.2 $\mathcal{NP}$ Search Problems with Guaranteed Witnesses

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

Theorem 20.1 (Computing a MNE Not $\mathcal{FNP}$-Complete) *The problem of computing a MNE of a bimatrix game is not $\mathcal{FNP}$-complete unless $\mathcal{NP} = co\mathcal{NP}$.*

While $\mathcal{NP} = co\mathcal{NP}$ doesn't immediately imply that $\mathcal{P} = \mathcal{NP}$, experts regard it as an equally unlikely state of affairs. For example, if $\mathcal{NP} = co\mathcal{NP}$, then the $co\mathcal{NP}$-complete unsatisfiability problem has short and efficiently verifiable proofs of membership. Convincing someone that a formula in propositional logic is satisfiable is easy

[1] Intuition for the class $\mathcal{NP}$ works fine for $\mathcal{FNP}$. For example, the proofs showing that the decision version of the satisfiability problem (SAT) is $\mathcal{NP}$-complete also show that the functional version of SAT is $\mathcal{FNP}$-complete.

[2] To be completely rigorous, we also need to argue that there is a MNE whose description length (in bits) is polynomial in that of $\mathbf{A}$ and $\mathbf{B}$ (Exercise 20.1).

enough—just exhibit a satisfying truth assignment. But how would you quickly convince someone that none of the exponentially many truth assignments satisfy a formula? Most researchers believe that there is no way to do it, or equivalently that $\mathcal{NP} \neq co\mathcal{NP}$. If this is indeed the case, then Theorem 20.1 implies that the problem of computing a MNE of a bimatrix game is not $\mathcal{FNP}$-complete.

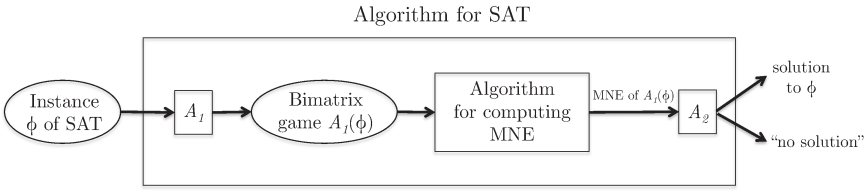


Figure 20.1: A reduction from the functional SAT problem to the problem of computing a MNE of a bimatrix game. Such a reduction would yield a polynomial-time verifier for the unsatisfiability problem.

Proof of Theorem 20.1: The proof is short but a bit of a mind-bender. Suppose there is a reduction, in the same sense of the $\mathcal{PLS}$ reductions described in Section 19.2.3, from the functional SAT problem to the problem of computing a MNE of a bimatrix game. By definition, the reduction comprises two algorithms:

1. A polynomial-time algorithm $\mathcal{A}_1$ that maps every SAT formula ϕ to a bimatrix game $\mathcal{A}_1(\phi)$.

2. A polynomial-time algorithm $\mathcal{A}_2$ that maps every MNE $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ of a game $\mathcal{A}_1(\phi)$ to a satisfying assignment $\mathcal{A}_2(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ of ϕ, if one exists, and to the string "no" otherwise.

See also Figure 20.1.

We claim that the existence of these algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ imply that $\mathcal{NP} = co\mathcal{NP}$. In proof, consider an unsatisfiable SAT formula ϕ, and an arbitrary MNE $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ of the game $\mathcal{A}_1(\phi)$.[3] We claim that $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ is a short, efficiently verifiable proof of the unsatisfiability of ϕ, implying that $\mathcal{NP} = co\mathcal{NP}$. Given an alleged certificate $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ that ϕ is unsatisfiable, the verifier performs two checks: (1) compute the

[3]Crucially, $\mathcal{A}_1(\phi)$ has at least one MNE (Theorem 20.5), including one whose description length is polynomial in that of the game (Exercise 20.1).

game $\mathcal{A}_1(\phi)$ using algorithm $\mathcal{A}_1$ and verify that $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ is a MNE of $\mathcal{A}_1(\phi)$; (2) use the algorithm $\mathcal{A}_2$ to verify that $\mathcal{A}_2(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ is the string "no." This verifier runs in time polynomial in the description lengths of ϕ and $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$. If $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ passes both of these tests, then correctness of the algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ implies that ϕ is unsatisfiable. ■

What's really going on in the proof of Theorem 20.1 is a mismatch between a $\mathcal{FNP}$-complete problem like the functional version of SAT, where an instance may or may not have a witness, and a problem like computing a MNE, where every instance has at least one witness. While the correct answer to a SAT instance might well be "no," a correct answer to an instance of MNE computation is always a MNE.

The subset of $\mathcal{FNP}$ problems for which every instance has at least one witness is called $\mathcal{TFNP}$, for "total functional $\mathcal{NP}$." The proof of Theorem 20.1 shows more generally that if *any* $\mathcal{TFNP}$ problem is $\mathcal{FNP}$-complete, then $\mathcal{NP} = co\mathcal{NP}$. In particular, since every instance of a $\mathcal{PLS}$ problem has at least one witness—local search has to stop somewhere, necessarily at a local optimum—$\mathcal{PLS}$ is a subset of $\mathcal{TFNP}$. Hence no $\mathcal{PLS}$ problem, such as computing a PNE of a congestion game, can be $\mathcal{FNP}$-complete unless $\mathcal{NP} = co\mathcal{NP}$.

Theorem 20.2 ($\mathcal{PLS}$ Problems Not $\mathcal{FNP}$-Complete)
No $\mathcal{PLS}$ problem is $\mathcal{FNP}$-complete, unless $\mathcal{NP} = co\mathcal{NP}$.

Theorem 20.2 justifies the development in Lecture 19 of $\mathcal{PLS}$-completeness, a weaker analog of $\mathcal{FNP}$-completeness tailored for local search problems.

20.2.3 Syntactic vs. Semantic Complexity Classes

Membership in $\mathcal{TFNP}$ precludes proving that computing a MNE of a bimatrix game is $\mathcal{FNP}$-complete (unless $\mathcal{NP} = co\mathcal{NP}$). The sensible refined goal is to prove that the problem is $\mathcal{TFNP}$-complete, and hence as hard as any other problem in $\mathcal{TFNP}$.

Unfortunately, $\mathcal{TFNP}$-completeness is also too ambitious a goal. The reason is that $\mathcal{TFNP}$ does not seem to have complete problems. To explain, think about the complexity classes that *are* known to have complete problems—$\mathcal{NP}$ of course, and also classes like $\mathcal{P}$ and $\mathcal{PSPACE}$. What do these complexity classes have in common? They are "syntactic," meaning that membership can be characterized via acceptance by some concrete computational model, such as

polynomial-time or polynomial-space deterministic or nondeterministic Turing machines. In this sense, there is a generic reason for membership in these complexity classes.

Syntactically defined complexity classes always have a "generic" complete problem, where the input is a description of a problem in terms of the accepting machine and an instance of the problem, and the goal is to solve the given instance of the given problem. For example, the generic $\mathcal{NP}$-complete problem takes as input a description of a verifier, a polynomial time bound, and an encoding of an instance, and the goal is to decide whether or not there is a witness, meaning a string that causes the given verifier to accept the given instance in at most the given number of steps.

$\mathcal{TFNP}$ 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 $\mathcal{TFNP}$ because of the topological arguments that guarantee the existence of a MNE (see Section 20.5). Another problem in $\mathcal{TFNP}$ is factoring: given a positive integer, output its factorization. Here, membership in $\mathcal{TFNP}$ has a number-theoretic explanation. Can the guaranteed existence of a MNE of a game and of a factorization of an integer be regarded as separate instantiations of some "generic" $\mathcal{TFNP}$ argument? No one knows the answer.

20.2.4 Where We're Going

Section 20.3 defines a subclass of $\mathcal{TFNP}$, known as $\mathcal{PPAD}$, that characterizes the computational complexity of computing a MNE of a bimatrix game. Figure 20.2 summarizes the conjectured relationships between $\mathcal{PPAD}$ and the other complexity classes discussed.

The definition of $\mathcal{PPAD}$ is technical and may seem unnatural, but it is justified by the following result.

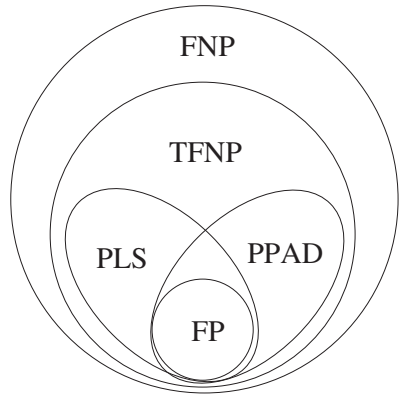
Theorem 20.3 (Computing a MNE Is $\mathcal{PPAD}$-Complete)
Computing a MNE of a bimatrix game is a $\mathcal{PPAD}$-complete problem.

The known proofs of Theorem 20.3 are much too long and complex to describe here (see the Notes). The remainder of this lecture focuses

[4] There are many other interesting examples, such as $\mathcal{NP} \cap co\mathcal{NP}$ (Exercise 20.2).

Complexity Class	Informal Definition
$\mathcal{FP}$	Polynomial-time solvable search problems
$\mathcal{FNP}$	Polynomial-time verifiable witnesses
$\mathcal{TFNP}$	Witnesses guaranteed to exist
$\mathcal{PLS}$	Solvable by local search
$\mathcal{PPAD}$	Solvable by directed path-following

(a) Recap of complexity classes



(b) Suspected relationships

Figure 20.2: Summary of complexity classes. The inclusions $\mathcal{PLS} \cup \mathcal{PPAD} \subseteq \mathcal{TFNP} \subseteq \mathcal{FNP}$ follow from the definitions. Every problem in $\mathcal{FP}$ can be viewed as a degenerate type of $\mathcal{PLS}$ or $\mathcal{PPAD}$ problem by treating the transcript of a correct computation as a legitimate (and efficiently verifiable) witness.

on the definition of and intuition behind the complexity class $\mathcal{PPAD}$, and explains why computing a MNE of a bimatrix game is a $\mathcal{PPAD}$ problem.

*20.3 $\mathcal{PPAD}$: A Syntactic Subclass of $\mathcal{TFNP}$

Our goal is to provide evidence of the computational intractability of the problem of computing a MNE of a bimatrix game by proving that it is complete for a suitable complexity class $\mathcal{C}$, where $\mathcal{C}$ is plausibly a strict superset of $\mathcal{FP}$. Section 20.2 argues that $\mathcal{C}$ needs to be a subset of $\mathcal{TFNP}$ that also has complete problems. Roughly equivalently, the class $\mathcal{C}$ should have a "syntactic" definition, in the form of a

generic reason for membership in $\mathcal{TFNP}$.

We already know one example of a subclass of $\mathcal{TFNP}$ that appears larger than $\mathcal{FP}$ and also admits complete problems, namely $\mathcal{PLS}$ (Section 19.2). For example, computing a local optimum of a maximum cut instance is a $\mathcal{PLS}$-complete problem that is not known to be polynomial-time solvable. The definition of $\mathcal{PLS}$ is syntactic in that every $\mathcal{PLS}$ problem is specified by the descriptions of three algorithms (Section 19.2.2). Among $\mathcal{PLS}$ problems, the common reason for membership in $\mathcal{TFNP}$ is that the generic local search procedure, using the three given algorithms as "black boxes," is guaranteed to eventually stop with a witness (a local optimum).

The right complexity class for studying the computation of MNE in bimatrix games is called $\mathcal{PPAD}$.[5] Before defining the class formally, we describe it by analogy with the class $\mathcal{PLS}$, which is similar in spirit but different in the details. The connection to computing MNE is far from obvious, and it is taken up in Section 20.5.

Recall that we can view a local search problem as one of searching a directed acyclic graph for a sink vertex (Figure 19.3). Vertices of this graph correspond to feasible solutions, such as outcomes in a congestion game, and the number of vertices can be exponential in the description length of the instance. Every $\mathcal{PLS}$ problem can be solved by a generic local search procedure that corresponds to following outgoing edges until a sink vertex is reached.

$\mathcal{PPAD}$ problems, like $\mathcal{PLS}$ problems, are those solvable by a particular generic path-following procedure. Every $\mathcal{PPAD}$ problem can be thought of as a directed graph (Figure 20.3), where the vertices again correspond to "solutions" and directed edges to "moves." The essential difference between $\mathcal{PLS}$ and $\mathcal{PPAD}$ is that the graph corresponding to a $\mathcal{PLS}$ problem is directed acyclic, while the graph corresponding to a $\mathcal{PPAD}$ problem is directed with all in- and out-degrees at most 1. Also, by definition, the graph corresponding to a $\mathcal{PPAD}$ problem has a canonical *source* vertex, meaning a vertex with no incoming edges. Traversing outgoing edges starting from the canonical source vertex cannot produce a cycle, since every vertex has in-degree at most 1 and the canonical source vertex has no incoming edges. We conclude that there is a sink vertex reachable from the

[5] The letters in $\mathcal{PPAD}$ stand for "polynomial parity argument, directed version."

canonical source vertex. Unlike a $\mathcal{PLS}$ problem, there is no objective function, and a $\mathcal{PPAD}$ directed graph can possess cycles. The witnesses of a $\mathcal{PPAD}$ problem are, by definition, all of the solutions that correspond to a sink vertex or to a source vertex other than the canonical one.

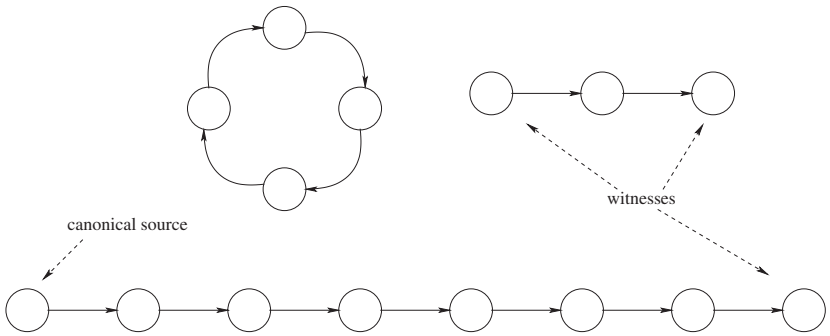


Figure 20.3: A $\mathcal{PPAD}$ problem corresponds to a directed graph with all in- and out-degrees at most 1.

Formally, as with $\mathcal{PLS}$, the class $\mathcal{PPAD}$ is defined syntactically as the problems that can be specified by three algorithms.

Ingredients of a $\mathcal{PPAD}$ Problem

1. The first polynomial-time algorithm takes as input an instance and outputs two distinct solutions, corresponding to the canonical source vertex and its successor.

2. The second polynomial-time algorithm takes as input an instance and a solution x other than the canonical source vertex and its successor, and either returns another solution y, the *predecessor* of x, or declares "no predecessor."

3. The third polynomial-time algorithm takes as input an instance and a solution x other than the canonical source vertex, and either returns another solution y, the *successor* of x, or declares "no successor."

These three algorithms implicitly define a directed graph. The vertices correspond to the possible solutions. There is a directed edge

(x, y) if and only if the second algorithm outputs x as the predecessor of y and the third algorithm agrees that y is the successor of x. There is also a directed edge from the canonical source vertex to its successor, as defined by the first algorithm; this ensures that the canonical source vertex is not also a sink vertex. Every vertex of the graph has in- and out-degree at most 1. The witnesses are all of the solutions that correspond to vertices with in-degree or out-degree 0, other than the canonical source vertex.[6]

Every $\mathcal{PPAD}$ problem has at least one witness, and a witness can be computed by a generic path-following procedure that uses the given three algorithms as "black boxes." This procedure effectively traverses outgoing edges, beginning at the canonical source vertex, until a sink vertex (a witness) is reached.[7] In this sense, the problems in $\mathcal{PPAD}$ have a generic reason for membership in $\mathcal{TFNP}$.

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

*20.4 A Canonical $\mathcal{PPAD}$ Problem: Sperner's Lemma

This section presents a computational problem that is clearly well matched with the $\mathcal{PPAD}$ complexity class. Section 20.5 describes its relevance to computing a MNE of a bimatrix game.

Consider a subdivided triangle in the plane (Figure 20.4). A *legal coloring* of its vertices colors the top corner vertex red, the left corner vertex green, and the right corner 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 small triangle is *trichromatic* if all three colors are represented at its vertices.

Sperner's lemma asserts that for every legal coloring, there is at least one trichromatic triangle.[8]

[6] As in the definition of a $\mathcal{PLS}$ problem, there is some canonical interpretation when an algorithm misbehaves. For example, the output of the third algorithm is interpreted as "no successor" if it does not stop within a specified polynomial number of steps, and also if it outputs the canonical source vertex or its successor.

[7] The purpose of the second algorithm is to keep the third algorithm honest, and ensure that the latter never creates an in-degree larger than 1.

[8] The same result and proof extend by induction to higher dimensions. Every

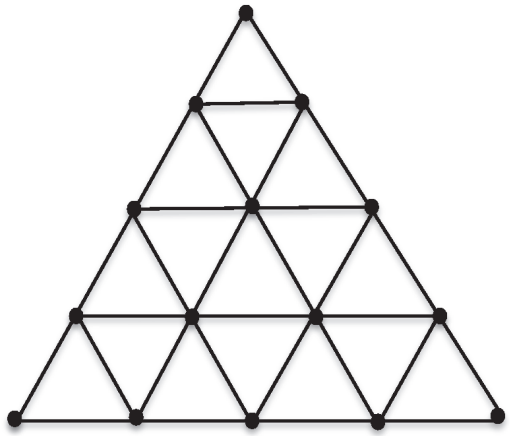


Figure 20.4: A subdivided triangle in the plane.

Theorem 20.4 (Sperner's Lemma) *For every legal coloring of a subdivided triangle, there is an odd number of trichromatic triangles.*

Proof: The proof is constructive. Define an undirected graph G that has one vertex corresponding to each small triangle, plus a source vertex that corresponds to the region outside the big triangle. The graph G has one edge for each pair of small triangles that share a side with one red and one green endpoint. Every trichromatic small triangle corresponds to a degree-one vertex of G. Every small triangle with one green and two red corners or two green and one red corners corresponds to a vertex with degree two in G. The source vertex of G has degree equal to the number of red-green segments on the left side of the big triangle, which is an odd number. Because every undirected graph has an even number of vertices with odd degree, there is an odd number of trichromatic triangles. ■

The proof of Sperner's lemma shows that following a path from a canonical source vertex in a suitable graph leads to a trichromatic triangle. Thus, computing a trichromatic triangle of a legally colored subdivided triangle is a $\mathcal{PPAD}$ problem.[9]

subdivided simplex in $\mathbb{R}^n$ with vertices legally colored with $n+1$ colors has an odd number of panchromatic subsimplices, with a different color at each vertex.

[9]We're glossing over some details. The graph of a $\mathcal{PPAD}$ problem is directed,

*20.5 MNE and $\mathcal{PPAD}$

What does computing a MNE have to do with $\mathcal{PPAD}$, the subclass of $\mathcal{FNP}$ problems that are solvable by a particular generic path-following procedure? There are two fundamental connections.

20.5.1 Sperner's Lemma and Nash's Theorem

Sperner's lemma (Theorem 20.4) turns out to be the combinatorial heart of Nash's theorem, stating that every finite game has at least one MNE.

Theorem 20.5 (Nash's Theorem) *Every finite game has at least one mixed Nash equilibrium.*

The reduction of Nash's theorem to Sperner's lemma has two parts. The first part is to use Sperner's lemma to prove Brouwer's fixed-point theorem. The latter theorem states that every continuous function f that maps a convex compact subset C of $\mathbb{R}^n$ to itself has at least one fixed point, meaning a point $x \in C$ with $f(x) = x$.[10]

Consider the special case where C is a simplex in $\mathbb{R}^2$. Let $f : C \to C$ be continuous. Subdivide C into small triangles as in Figure 20.4. Color a triangle corner x 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, representing a triangle whose corners are pulled in different directions by f. 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 subsequence that

while the graph G defined in the proof of Theorem 20.4 is undirected. There is, however, a canonical way to direct the edges of the graph G. Also, the canonical source vertex of a $\mathcal{PPAD}$ problem has out-degree 1, while the source of the graph G has degree $2k-1$ for some positive integer k. This can be rectified by splitting the source vertex of G into k vertices, a source vertex with out-degree 1 and $k-1$ vertices with in- and out-degree 1.

[10] So after stirring a cup of coffee, there is some point in the coffee that ends up exactly where it started!

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.[11]

To sketch how Nash's theorem (Theorem 20.5) reduces to Brouwer's fixed-point theorem, consider a k-agent game with strategy sets $S_1, \ldots, S_k$ and payoff functions $\pi_1, \ldots, \pi_k$. The relevant convex compact set is $C = \Delta_1 \times \cdots \times \Delta_k$, where Δ_i is the simplex representing the mixed strategies over S_i. We want to define a continuous function $f : C \to C$, from mixed strategy profiles to mixed strategy profiles, such that the fixed points of f are the MNE of this game. We define f separately for each component $f_i : C \to \Delta_i$. A natural idea is to set f_i to be a best response of agent i to the mixed strategy profiles of the other agents. This does not lead to a continuous, or even well defined, function. We instead use a "regularized" version of this idea, defining

$$f_i(x_i, \mathbf{x}_{-i}) = \operatorname*{argmax}_{x_i' \in \Delta_i} g_i(x_i', \mathbf{x}), \tag{20.3}$$

where

$$g_i(x_i', \mathbf{x}) = \underbrace{\mathop{\mathbf{E}}_{s_i \sim x_i', \mathbf{s}_{-i} \sim \mathbf{x}_{-i}} [\pi_i(\mathbf{s})]}_{\text{linear in } x_i'} - \underbrace{\|x_i' - x_i\|_2^2}_{\text{strictly convex}}. \tag{20.4}$$

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 (Exercise 20.5). By definition, every MNE of the given game is a fixed point of f. For the converse, suppose that $\mathbf{x}$ is not a MNE, with agent i able to increase her expected payoff by deviating unilaterally from x_i to x_i'. A simple computation shows that, for sufficiently small $\epsilon > 0$,

[11] Here's the idea for extending Brouwer's fixed-point theorem to all convex compact subsets of $\mathbb{R}^n$. First, since Sperner's lemma extends to higher dimensions, the same argument shows that Brouwer's fixed-point theorem holds for simplices in any number of dimensions. Second, radial projection shows that every pair C_1, C_2 of convex compact subsets of equal dimension are homeomorphic, meaning there is a bijection $h : C_1 \to C_2$ with h and h^{-1} continuous. Homeomorphisms preserve fixed-point theorems (Exercise 20.4).

$g_i((1-\epsilon)x_i + \epsilon x_i', \mathbf{x}) > g_i(x_i, \mathbf{x})$, and hence $\mathbf{x}$ is not a fixed point of f (Exercise 20.6).

This proof of Nash's theorem translates the path-following algorithm for computing a panchromatic subsimplex of a legally colored subdivided simplex to a path-following algorithm for computing an approximate MNE of a finite game. This establishes membership of this problem in $\mathcal{PPAD}$.

20.5.2 The Lemke-Howson Algorithm

There is also a second way to prove that computing a MNE of a bimatrix game is a $\mathcal{PPAD}$ problem, via the Lemke-Howson algorithm. Describing this algorithm is outside the scope of this book, but the essential point is that 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. These convergence proofs show that the problem of computing a MNE of a bimatrix game lies in $\mathcal{PPAD}$.

20.5.3 Final Remarks

The two connections between $\mathcal{PPAD}$ and computing a MNE are incomparable. The Lemke-Howson algorithm applies only to games with two players, but it shows that the problem of computing an *exact* MNE of a bimatrix game belongs to $\mathcal{PPAD}$. The path-following algorithm derived from Sperner's lemma applies to games with any fixed finite number of players, but only shows that the problem of computing an *approximate* MNE is in $\mathcal{PPAD}$.[12]

In any case, we conclude that our motivating problem of computing a MNE of a bimatrix game is a $\mathcal{PPAD}$ problem. Theorem 20.3 shows the complementary result that the problem is as hard as every

[12] In fact, with 3 or more players, the problem of computing an exact MNE of a game appears to be strictly harder than any problem in $\mathcal{PPAD}$ (see the Notes).

other problem that can be solved by a generic path-following procedure in a directed graph with a canonical source vertex and all in- and out-degrees at most 1. This shows that $\mathcal{PPAD}$ is, at last, the right complexity class for building evidence that the problem is computationally intractable.

For example, Theorem 20.3 implies that the problem of computing an approximate fixed point of a finitely represented continuous function, a $\mathcal{PPAD}$ problem, reduces to the problem of computing a MNE. This effectively reverses the reduction outlined in Section 20.5.1.

20.6 Discussion

One interpretation of Theorem 20.3, which is somewhat controversial, 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.

Intractability is not necessarily first on the list of the Nash equilibrium's issues. For example, its non-uniqueness already limits its predictive power in many settings. But the novel computational intractability critique in Theorem 20.3 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 20.3 suggests shining a brighter spotlight on computationally tractable classes of games and equilibrium concepts. For example, our convergence guarantees for no-regret dynamics motivate identifying properties that hold for all correlated or coarse correlated equilibria of a game.

How hard are $\mathcal{PPAD}$ problems, anyways? This basic question is not well understood. In the absence of an unconditional proof about whether or not $\mathcal{PPAD}$ problems are polynomial-time solvable, it is important to relate the assumption that $\mathcal{PPAD} \not\subseteq \mathcal{FP}$ to other complexity assumptions stronger than $\mathcal{P} \neq \mathcal{NP}$. For example, can we base the computational intractability of $\mathcal{PPAD}$ problems on cryptographic assumptions like the existence of one-way functions?

The Upshot

- ☆ There is no known polynomial-time algorithm for computing a MNE of a bimatrix game, despite significant effort by many experts.
- ☆ $\mathcal{TFNP}$ is the subclass of $\mathcal{NP}$ search problems ($\mathcal{FNP}$) for which the existence of a witness is guaranteed. Examples include all $\mathcal{PLS}$ problems and computing a MNE of a bimatrix game.
- ☆ No $\mathcal{TFNP}$ problem can be $\mathcal{FNP}$-complete, unless $\mathcal{NP} = co\mathcal{NP}$.
- ☆ $\mathcal{TFNP}$ does not seem to contain complete problems.
- ☆ $\mathcal{PPAD}$ is the subclass of $\mathcal{TFNP}$ where a witness can always be computed by a generic path-following procedure in a directed graph with a source vertex and all in- and out-degrees at most 1.
- ☆ The problem of computing a MNE of a bimatrix game is $\mathcal{PPAD}$-complete.

Notes

The definition of $\mathcal{TFNP}$ and Theorem 20.1 are due to Megiddo and Papadimitriou (1991). Theorem 20.2 is from Johnson et al. (1988). The complexity class $\mathcal{PPAD}$, as well as several other syntactically defined subclasses of $\mathcal{TFNP}$, is defined by Papadimitriou (1994). Daskalakis et al. (2009a) develop most of the machinery in the proof of Theorem 20.3 and prove that computing an approximate MNE of a three-player game is $\mathcal{PPAD}$-complete. The result stated for bimatrix games (Theorem 20.3) is due to Chen and Deng; see Chen et al. (2009). For overviews of this proof in order of increasing levels of detail, see Roughgarden (2010b),

Papadimitriou (2007), and Daskalakis et al. (2009b). Sperner's lemma (Theorem 20.4) is from Sperner (1928). Our proof of Nash's theorem (Theorem 20.5) follows Geanakoplos (2003) and is a variant of the one in Nash (1951). The Lemke-Howson algorithm is from Lemke and Howson (1964); see von Stengel (2002) for a thorough exposition. Etessami and Yannakakis (2010) show that the problem of exact MNE computation in games with more than two players is complete for a complexity class $\mathcal{FIXP}$ that appears to be strictly larger than $\mathcal{PPAD}$. Bitansky et al. (2015) and Rosen et al. (2016) discuss the prospects of basing the intractability of $\mathcal{PPAD}$-complete problems on cryptographic assumptions.

Problem 20.2, which shows that an approximate MNE of a bimatrix game can be computed in quasipolynomial time, is due to Lipton et al. (2003).[13] Such an approximate MNE need not be close to any exact MNE. Rubinstein (2016) proves that, under plausible complexity assumptions, there is no significantly faster algorithm for computing an approximate MNE. Problem 20.3 is derived from Brown and von Neumann (1950) and Gale et al. (1950).

Exercises

Exercise 20.1 *(H)* Assume for this exercise that every system of linear equations $\mathbf{Cx} = \mathbf{d}$ that has a solution has at least one solution with description length (in bits) polynomial in that of $\mathbf{C}$ and $\mathbf{d}$.

Prove that every bimatrix game $(\mathbf{A}, \mathbf{B})$ has a MNE with description length polynomial in that of the payoff matrices $\mathbf{A}$ and $\mathbf{B}$.

Exercise 20.2 *(H)* Consider the following problem: given descriptions of a nondeterministic algorithm $\mathcal{A}_1$ (for accepting "yes" instances) and a co-nondeterministic algorithm $\mathcal{A}_2$ (for accepting "no" instances), a polynomial time bound, and an encoding of an instance, find a witness that causes one of $\mathcal{A}_1, \mathcal{A}_2$ to accept the given instance in the prescribed number of time steps.

Why isn't this a generic complete problem for $\mathcal{NP} \cap co\mathcal{NP}$, in the sense of Section 20.2.3?

[13] The approximation in this result is additive. Daskalakis (2013) proves that for multiplicative approximation, as in Definitions 14.5 and 16.2, the problem of computing an approximate MNE of a bimatrix game is $\mathcal{PPAD}$-complete, and hence unlikely to admit a quasipolynomial-time algorithm.

Exercise 20.3 This exercise demonstrates the necessity of all of the hypotheses of Brouwer's fixed-point theorem, even in a single dimension.

(a) Exhibit a (discontinuous) function f from a compact interval to itself that has no fixed point.

(b) Exhibit a continuous function f from the union of two compact intervals to itself that has no fixed point.

(c) Exhibit a continuous function f from a bounded open interval to itself that has no fixed point.

Exercise 20.4 Suppose C_1 and C_2 are subsets of $\mathbb{R}^n$ that are *homeomorphic*, meaning that there is a bijection $h : C_1 \to C_2$ such that both h and h^{-1} are continuous. Prove that if every continuous function from C_1 to itself has a fixed point, then every continuous function from C_2 to itself has a fixed point.

Exercise 20.5 Prove that the function defined in (20.3)–(20.4) is continuous.

Exercise 20.6 In the mixed strategy profile $\mathbf{x}$, suppose that agent i can increase her expected payoff by deviating unilaterally from x_i to x_i'. Prove that $g_i((1-\epsilon)x_i + \epsilon x_i', \mathbf{x}) > g_i(x_i, \mathbf{x})$ for all sufficiently small $\epsilon > 0$, where g_i is the function defined in (20.4).

Problems

Problem 20.1 *(H)* Assume for this problem that there is a polynomial-time algorithm that determines whether or not a system of linear equations has a solution, and computes a solution if one exists. Use such an algorithm as a subroutine to compute a MNE of a bimatrix game in time bounded above by $2^n \cdot p(n)$ for some polynomial p, where n is the combined number of rows and columns.

Problem 20.2 Let $(\mathbf{A}, \mathbf{B})$ be a bimatrix game in which each player has at most n strategies and all payoffs lie in $[0, 1]$. An *ϵ-approximate*

mixed Nash equilibrium (ϵ-MNE) is a pair $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ of mixed strategies over the rows and columns such that

$$\hat{\mathbf{x}}^\top \mathbf{A}\hat{\mathbf{y}} \geq \mathbf{x}^\top \mathbf{A}\hat{\mathbf{y}} - \epsilon$$

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

$$\hat{\mathbf{x}}^\top \mathbf{B}\hat{\mathbf{y}} \geq \hat{\mathbf{x}}^\top \mathbf{B}\mathbf{y} - \epsilon$$

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

(a) *(H)* Fix $\epsilon \in (0, 1)$, let $(\mathbf{x}^*, \mathbf{y}^*)$ be a MNE of $(\mathbf{A}, \mathbf{B})$, and define $K = (b \ln n)/\epsilon^2$, where $b > 0$ is a sufficiently large constant, independent of n and ϵ. Let $(r_1, c_1), \ldots, (r_K, c_K)$ denote K outcomes sampled independently from the product distribution defined by $(\mathbf{x}^*, \mathbf{y}^*)$. Let $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ denote the corresponding marginal empirical distributions. For example, the component $\hat{x}_i$ is defined as the fraction of outcomes (r_ℓ, c_ℓ) for which r_ℓ is the row i. Prove that, with high probability over the choice of $(r_1, c_1), \ldots, (r_K, c_K)$, $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ is an ϵ-MNE.

(b) *(H)* Give an algorithm for computing an ϵ-MNE of a bimatrix game that runs in time at most $n^{2K} \cdot p(n)$ for some polynomial p, where K is defined as in (a).[14]

(c) Extend your algorithm and analysis to compute an ϵ-MNE with expected total payoff close to that of the maximum expected payoff achieved by any exact MNE. Your running time bound should remain the same.

Problem 20.3 A bimatrix game $(\mathbf{A}, \mathbf{B})$ is *symmetric* if both players have the same strategy set and $\mathbf{B} = \mathbf{A}^\top$. Examples include Rock-Paper-Scissors (Section 18.3.1) and the traffic light game (Section 13.1.4). Nash's theorem and the Lemke-Howson algorithm can both be adapted to show that every symmetric game has a *symmetric* MNE $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$, in which both players employ the same mixed strategy (i.e., $\hat{\mathbf{x}} = \hat{\mathbf{y}}$).

[14] A running time bound of the form $n^{b \ln^a n}$ for constants a and b is called *quasipolynomial*. Such a bound is bigger than every polynomial but smaller than every exponential function.

(a) *(H)* Give a reduction, in the sense of Section 19.2.3, from the problem of computing a MNE of a general bimatrix game to that of computing a symmetric MNE of a symmetric bimatrix game.

(b) Give a reduction from the problem of computing a MNE of a general bimatrix game to that of computing *any* MNE of a symmetric bimatrix game.

(c) *(H)* Give a reduction from the problem of computing a MNE of a general bimatrix game to that of the problem of computing an *asymmetric* MNE of a symmetric bimatrix game, or reporting "no solution" if there are no asymmetric MNE.