# Computing Equilibria: A Computational Complexity Perspective

Tim Roughgarden*

February 20, 2009

**Abstract**

Computational complexity is the subfield of computer science that rigorously studies the intrinsic difficulty of computational problems. This survey explains how complexity theory defines "hard problems"; applies these concepts to several equilibrium computation problems; and discusses implications for computation, games, and behavior. We assume minimal prior background in computer science.

## 1 Introduction

### 1.1 "Easy" and "Hard" Computational Problems

Consider two problems about grouping individuals together in a harmonious way.

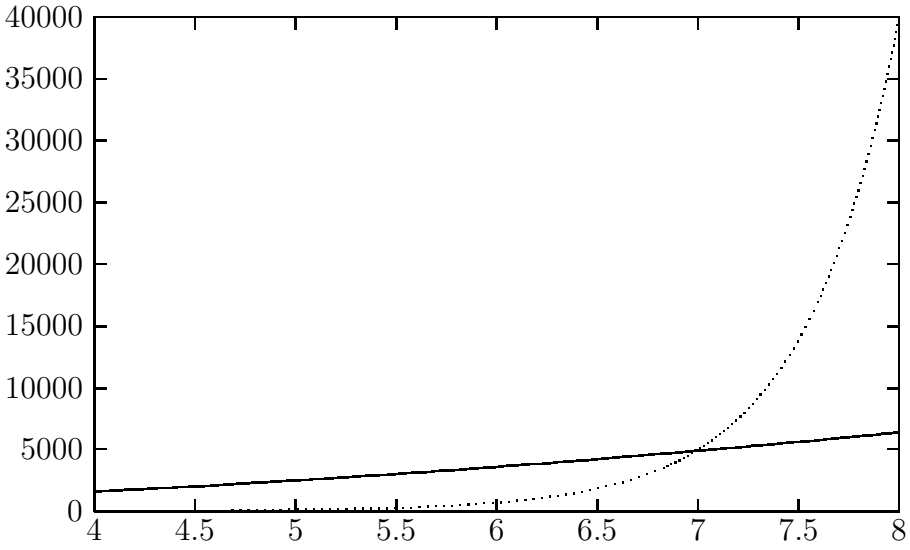
1. You have a database with the rankings of 1000 medical school graduates over 1000 hospitals, and of these hospitals over these new doctors. Your task is to find a matching of doctors to hospitals that is *stable*: there should be no doctor-hospital pair who would rather be matched to each other than to their assigned partners.

2. You work for a company with 150 employees, many of whom do not get along with each other. Your task is to form a committee of fifteen people, no two of whom are incompatible, or to show that this is impossible.

Is one of these problems "computationally more difficult" than the other? If you had to solve only one of them — by the day's end, say — which would you choose?

The first problem is, of course, the *stable marriage* problem, and can be solved automatically via the Gale-Shapley deferred acceptance algorithm [46]. A byproduct of this algorithm is a constructive proof of a fundamental existence result: no matter what the preferences of the agents are, there is at least one stable marriage (with every agent in some pairing). The algorithm is also *computationally efficient*; indeed, a variant of it had been deployed in practice several years prior to the analysis of Gale and Shapley (see Roth [110] and Stalnaker [126]). Despite the astronomical number of possibilities — $n!$ distinct matchings among two groups of size $n$, which is more than the estimated

*Department of Computer Science, Stanford University, 462 Gates Building, 353 Serra Mall, Stanford, CA 94305. Supported in part by NSF CAREER Award CCF-0448664, an ONR Young Investigator Award, an AFOSR MURI grant, and an Alfred P. Sloan Fellowship. Email: tim@cs.stanford.edu.




(a) Growth of $100n^2$ vs. $n!$

| $n$ | 2 | 4 | 6 | 7 | 8 | 10 | 25 | 70 |
|---|---|---|---|---|---|---|---|---|
| $100n^2$ | 400 | 1,600 | 3,600 | 4,900 | 6,400 | 10,000 | 62,500 | 490,000 |
| $n!$ | 2 | 24 | 720 | 5,040 | 40,320 | 3,628,800 | $> 10^{25}$ | $> 10^{100}$ |

(b) Growth of $100n^2$ vs. $n!$

Figure 1: The factorial function $n!$ grows exponentially faster with $n$ than the function $100n^2$. In (a), the solid curve is the function $f(x) = 100x^2$; the dotted curve is Stirling's approximation of $n!$, extended to the positive reals $x$.

number of atoms in the known universe when $n = 1000$ — the amount of work performed by the algorithm scales only as a constant times $n^2$, enabling the quick and automated solution of very large problems.[1] Figure 1 shows just how differently these two functions grow with $n$.

What about the second problem? It cannot be solved efficiently by enumerating all of the possibilities — there are $\binom{150}{15} > 10^{15}$ of them, too many to check in a reasonable amount of time on today's computers. Is there a more clever general-purpose solution to this seemingly innocuous problem? The answer is believed to be "no". This belief is rigorously translated into mathematics by proving that the problem is "$NP$-complete", a concept that we define and interpret in Section 2.

Similarly, some equilibrium computation problems are apparently harder than others. Problems like computing a stable marriage, and computing a Nash equilibrium of a two-player zero-sum game, admit computationally efficient general-purpose solutions (the latter via linear programming [29]). No such solutions have been discovered for many other equilibrium computation problems, such as computing a Nash equilibrium in a general game. Could there be something intrinsically difficult, in a precise sense, about these problems? *Computational complexity* is the subfield of computer science concerned with such questions; its basic definitions and its consequences for equilibrium computation are the subjects of this survey.

What do we learn by proving that an equilibrium computation problem is "difficult" in a complexity-theoretic sense? First, assuming widely believed mathematical conjectures, it implies

[1]For implementation details, see Kleinberg and Tardos [75, §2.3].

that there will never be a fast, general-purpose computational method for solving the problem. Second, it rules out many structural results, such as convexity or duality, that are often used to understand and justify economic models. Finally, as we explain in the next section, a complexity-theoretic hardness result can diminish the predictive interpretation of an equilibrium concept and suggests more tractable alternatives.

### 1.2 Who Cares About Computing Equilibria?

Many interpretations of an equilibrium concept involve someone — the participants or a designer — determining an equilibrium. For example, the idea that markets implicitly compute a solution to a significant computational problem goes back at least to Adam Smith's notion of the invisible hand [125]. If all parties are boundedly rational, then an equilibrium can be interpreted as a credible prediction only if it can be computed with reasonable effort. Rigorous intractability results thus cast doubt on the predictive power of equilibrium concepts.[2] In a practical design context, it is obvious that a mechanism that is actually implemented had better be computationally tractable to run, like the deferred acceptance algorithm, and also easy to play, in the sense that participants should not need to perform difficult computations.[3]

Equilibrium computation also has a number of well-known applications in economic analysis: predicting the outcome and estimating suitable parameter values of a model, comparing experimental results with model predictions, testing the design of a mechanism, automatically generating conjectures and counterexamples, and so on. These topics are discussed in detail elsewhere [13, 36, 38, 65, 84].

Computer scientists also care about computing equilibria. The original motivating application was the design of automated game-playing programs, such as for chess; a sample of modern applications is reviewed by Koller and Pfeffer [78]. For example, automated agents — anything from a physical robot to an automated bidding strategy in an Internet auction — are often programmed to optimize some payoff function while participating in a competitive or cooperative environment (see e.g. Shoham and Leyton-Brown [120]). Computationally efficient algorithms for computing equilibria are valuable tools for automating the decision-making of such agents.

Finally, complexity theorists are interested equilibrium computation problems because they include several of the most natural problems believed to be of "intermediate difficulty", between "easy" (solvable by a polynomial-time algorithm, see Section 2.1) and "hard" ($NP$-hard, see Section 2.2). Their study has led to some recent advances in complexity theory.

### 1.3 Overview

Section 2 formally defines the standard paradigm for proving that a problem is "hard" and develops the theory of "$NP$-hardness". The key tool is a *reduction*, a precise way of deeming one problem at least as hard as another. A problem is then "hard for some problem class" if it is at least as hard as every problem in the class. The problem class $NP$ denotes the (very large) set of problems for which purported solutions can be verified quickly for correctness. Informally, we interpret an $NP$-hard problem as one that admits no general-purpose computational solution that is significantly faster than brute-force search (i.e., systematically enumerating all possible solutions). As an example, we prove that determining whether or not a given two-player normal-form game has a Nash equilibrium

[2] This type of critique dates back at least to Rabin [106].

[3] See Milgrom [89] for a recent example of these principles.

in which both players obtain an expected payoff of at least a given amount is an $NP$-hard problem. The section also includes a careful discussion of the assumptions and implications of $NP$-hardness results.

Different equilibrium computation problems require different notions of "hardness", and Section 3 explains why. This section focuses on congestion games, where a potential function argument guarantees the convergence of better-response dynamics to a pure-strategy Nash equilibrium. We show that the problem of computing a pure-strategy Nash equilibrium of a congestion game is $PLS$-hard, where $PLS$ is a general class of local search problems. A consequence of this result is that better-response dynamics can require an exorbitant number of iterations to converge in congestion games. The section concludes by explaining the behavioral implications of these negative results.

Section 4 studies another, more fundamental equilibrium computation problem where the existence of a solution is guaranteed: computing a (mixed-strategy) Nash equilibrium of a bimatrix game. Again, $NP$-hardness is not the right concept of difficulty for this problem; we instead introduce the class $PPAD$, a relative of $PLS$ that is motivated by the Lemke-Howson algorithm, for this purpose. We briefly discuss very recent and deep research showing that computing a Nash equilibrium of a bimatrix game is a $PPAD$-hard problem, and provide economic interpretations. The computational complexity perspective echoes a number of previous results suggesting that, for predictive purposes, the correlated equilibrium can be a more appropriate solution concept than the Nash equilibrium.

### 1.4 Omitted Topics

Each of the next three sections concentrates on one problem concerning Nash equilibrium computation, to illustrate a particular type of intractability result. Dozens of other equilibrium computation problems have been studied from a complexity-theoretic viewpoint. The most prominent topics omitted from this survey include:

- equilibrium computation in extensive-form games (e.g. [9, 24, 54, 77, 137]), normal-form games with three or more players [41], compactly represented games (e.g. [31, 72, 102, 118]), stochastic games (surveyed by Johnson [63, §2] and Yannakakis [140, §2]), and cooperative games (e.g. [35, 68]);
- computing equilibrium refinements (e.g. [40, 90, 135]);
- the computational complexity of playing a repeated game, from the perspective of a boundedly rational player (surveyed by Kalai [67], see also Section 2.6) or of a designer or mediator [14, 83];
- equilibrium computation in markets, initiated by Scarf [116] and recently surveyed by Vazirani [132] and Codenotti and Varadarajan [25];
- and complexity results in mechanism design, such as the winner determination problem in combinatorial auctions (surveyed by Blumrosen and Nisan [11] and Lehmann, Müller, and Sandholm [79]).

We also omit discussion of non-trivial algorithms for computing equilibria, with the exception of the Lemke-Howson algorithm (Section 4.2). As the other papers in this special issue attest, there

are a number of such algorithms that are practically useful for problems that are sufficiently small or well-structured.

# 2 Algorithms, Reductions, and Hardness

This section formalizes the notions of "good algorithms" (Section 2.1) and "hard problems" (Sections 2.2 and 2.3). Sections 2.4 and 2.5 illustrate the theory of $NP$-hardness by showing that determining whether or not a bimatrix game has a high-payoff Nash equilibrium is an $NP$-hard problem. Section 2.6 offers interpretations of $NP$-hardness results, addresses the frequently voiced objections to worst-case analysis and the definition of polynomial time, and discusses alternatives to time complexity analysis, such as space-bounded models of bounded rationality. Our treatment is largely self-contained, though perhaps too brisk for the reader with absolutely no prior exposure to the material. Several good textbooks cover these topics in depth [53, 62, 75, 100, 123].

## 2.1 Computationally Efficient Algorithms

A compelling definition of a "hard problem" is one that admits no good algorithm. But what is a "good algorithm", or even just an "algorithm"? Numerous essentially equivalent definitions of an algorithm are possible, all of which describe a procedure for transforming an input to an output according to a predetermined and finite sequence of "basic instructions"; in addition, arbitrary internal state can be maintained throughout the procedure's execution.[4] Conceptually, an algorithm is a single program (in any programming language) that runs on a modern computer with unbounded memory. The most common formalism is a Turing machine [130], but all of the most common models of time-efficient computation are equivalent for our purposes.

Before suggesting what makes an algorithm "good", we must be precise about what it means to "solve a problem". A *problem* is a (generally infinite) collection of *instances*, each representing a possible input; and, for each instance, a collection of *solutions*. Instances and solutions should both be encoded in some canonical way. Consider, for example, a problem concerning a *graph* $G$, with vertices $V$ and edges $E$ (Figure 2). A graph can be specified by two integers (for $|V|$ and $|E|$), followed by a list of $|E|$ pairs of distinct integers in $\{1, 2, \ldots, |V|\}$, which are interpreted as unordered pairs denoting the endpoints of the edges. Representing each of these numbers in binary (say), we can canonically encode every graph as a sequence of zeroes and ones. An algorithm then *solves a problem* if, for every instance, it either terminates with a correct solution or correctly determines that no solution exists. A concrete example is to compute a large *clique* of a graph — a subset of mutually connected vertices — if one exists.[5] This is, of course, the same as the second problem in Section 1.1, with vertices corresponding to employees and edges to pairs of compatible employees.

[4] For simplicity, we disallow "randomized algorithms" — algorithms that can have different executions on the same input (depending, for example, on the value of a "random seed" derived from a computer's internal clock). Complexity theory has studied such algorithms deeply, however, and all evidence suggests that they cannot overcome the negative results we discuss.

[5] Complexity theory often focuses on *decision problems*, such as "correctly determine whether or not a given graph has a large clique", where only a "yes/no" answer is required. Equilibrium computation problems are more naturally phrased as *search problems*, in which a bona fide solution should be computed, if possible. We focus on search problems in this survey.

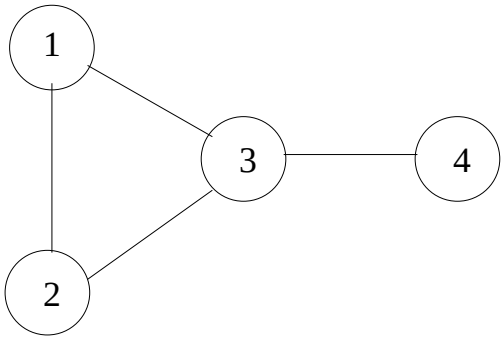



Figure 2: A graph with vertices $V = \{1, 2, 3, 4\}$ and edges $E = \{(1, 2), (1, 3), (2, 3), (3, 4)\}$. The graph contains a 3-clique but no 4-clique.

**Problem 2.1 (Maximum Clique Problem)** Given (canonical encodings of) a graph $G = (V, E)$ and an integer target $k$:

(a) if there is a set $K \subseteq V$ with $|K| = k$ and with $(i, j) \in E$ for every distinct $i, j \in K$, then output such a set;

(b) otherwise, indicate that $G$ has no such "$k$-clique".

Formally, an algorithm that solves Problem 2.1 should encode its output in a canonical way (e.g., by responding "0" in case (b)). If the algorithm is given an input that is not a valid encoding of a Maximum Clique instance, then the algorithm should recognize this and proceed as in case (b).

There is certainly an algorithm that solves the Maximum Clique problem, namely *brute-force search*. The brute-force search algorithm systematically enumerates each subset of $k$ vertices, and checks whether or not the subset forms a $k$-clique. Should this algorithm be deemed "good"?

Intuitively, no. The primary reason is it scales poorly with the problem size. To see this, consider a sequence of graphs in which the $n$th graph has $n$ vertices and the target $k$ is set to $n/2$. The lengths of the natural encodings of these Maximum Clique instances grow polynomially in $n$, since there are at most $\binom{n}{2} \approx n^2/2$ edges in a graph with $n$ vertices, and a number between 1 and $n$ can be encoded using $\approx \log_2 n$ zeroes and ones. The running time of brute-force search grows exponentially faster, almost as fast as the function $2^n$; qualitatively, the contrast shown in Figure 1(a) also applies to these two different functions. This brute-force search algorithm would be computationally feasible (would halt within a week, say) using today's technology for values of $n$ up to around 30, when $2^n$ is roughly a billion.

Some interesting problems are small enough to be solvable via exponential-time brute-force search. But many are not, and as the last 50 years of computational work across science and engineering has shown, our appetite for solving larger and larger problems is insatiable — as computers become more powerful, our ambitions grow. We should therefore judge a general-purpose algorithm not only by its performance on today's problems, but also by how well it scales to problems of moderate and large sizes — to *tomorrow's* problems. Thinking ahead to equilibrium computation problems, who wouldn't be interested in an exact equilibrium analysis of chess, or the poker game Texas Hold 'Em, or numerous less recreational large-scale applications, within our lifetimes?

Perhaps the paucity of insight in brute-force search can be overcome by tomorrow's lightning-fast processors? Moore's law, asserting that computing speeds double every 1-2 years [92], actually argues the opposite point: brute-force search will *never* be useful for large problems. Every time computing speeds double, an exponential-time algorithm will be able to handle problem sizes that are only *one more* than before, and thus will never be able to solve problems of even moderate size. This motivates an informal definition of a "good algorithm":

> *an algorithm is "good" if the input sizes that it can solve in a fixed amount of time scales multiplicatively with increasing computational power.*

This is equivalent to insisting that the running time of the algorithm scales *polynomially* with the problem size; for example, with a quadratic-time algorithm, a doubling in computing power yields a $\sqrt{2}$ factor growth in the solvable problem size. Here "running time" counts the number of "basic operations" — conceptually, one line of a program in some fixed programming language — that the algorithm executes before halting. This is the usual formal definition of a "good algorithm" in complexity theory.

**Definition 2.2 (Polynomial-Time Algorithm)** An algorithm is *polynomial time* if there are constants $c, k > 0$ such that, for every $n \geq 1$ and every input of length $n$, the algorithm halts after executing at most $cn^k$ basic operations.

For example, the Gale-Shapley deferred acceptance algorithm is polynomial time; the brute-force search algorithm for the Maximum Clique problem is not.

Definition 2.2 requires several comments. First, the definition is robust in the sense that, for any two reasonable formal models of algorithms (e.g., different choices of "basic operations"), a problem is solvable by a polynomial-time algorithm in one model if and only if it is similarly solvable in the other, possibly with a different choice of the constants $c$ and $k$ [131].[6] Second, when designing an algorithm to solve a problem, minimizing the constants $c$ and $k$ in Definition 2.2 is important. For example, a linear-time algorithm ($k = 1$) is typically far preferable to a quadratic-time one ($k = 2$). Since this survey emphasizes negative results — evidence that for some equilibrium computation problems, there is no algorithm that solves it polynomial time, for any choice of $c$ and $k$ — we will not worry unduly about precise values of $c$ and $k$. Finally, Definition 2.2 concerns "worst-case" performance, in that a polynomial-time algorithm is guaranteed to solve *every* instance quickly. Section 2.6 discusses the reasons for this assumption as well as potential alternatives.

There is also an intellectual reason why we should classify brute-force search as a "bad algorithm" and a polynomial-time algorithm as a "good algorithm": for a problem like stable marriage and a polynomial-time solution like the Gale-Shapley algorithm, the algorithm necessarily solves the problem while examining a vanishingly small fraction of the exponentially large solution space. In contrast to brute-force search, a polynomial-time algorithm necessarily exploits the structure of the problem — and implicitly provides a "proof" that such structure exists.

## 2.2 Verifiable Solutions and $NP$

We have already identified a candidate definition of a "hard problem":

[6] An exception is the Blum-Shub-Smale model of computation over the real numbers [10]. This model appears to be more powerful than the classical ones (see [4, 140]) but its connection to practical computation remains under debate (e.g. [16]).

*a problem is "hard" if there is no "good" (i.e., polynomial-time) algorithm that solves it.*

In other words, no significant improvement over brute-force search is possible. Determining whether or not the problems studied in this survey are hard in this sense is a deep, open question; see Section 2.6.

In lieu of establishing problem hardness in the above strong sense, we settle for the following:

(*) *a problem is "hard" if it is as difficult as every problem in a broad class.*

Put differently, if any problem in the class cannot be solved by a polynomial-time algorithm, then the "hard" problems cannot either. This and the next section provide the two definitions needed to formalize condition (*) — one for the appropriate "broad problem class", and one to anoint one problem "at least as hard as" another.

Suppose we wanted to apply the paradigm (*) to a problem like Maximum Clique, and prove that it is as hard as every problem in some class $\mathcal{C}$. The strength of such a result depends on how big the class $\mathcal{C}$ is. Why not choose $\mathcal{C}$ to be the class of *all* problems? This is too ambitious — a simple counting argument shows that many computational problems cannot be solved correctly by *any* algorithm, polynomial-time or otherwise.[7] We know that the Maximum Clique problem can be solved by an algorithm, namely brute-force search. We can therefore maximize our ambitions by, intuitively, choosing $\mathcal{C}$ to be the set of all problems that are solvable by an analogous brute-force search algorithm. This motivates the class $NP$, which we define next.[8]

Informally, a problem is in $NP$ if a purported solution to an instance can be verified for correctness in polynomial time. For example, in an instance of the Maximum Clique problem with graph $G$ and target $k$, a purported solution is a subset $K$ of $k$ vertices, and it is easy to check whether or not $K$ is a $k$-clique of $G$ in time polynomial in the input size. Formally, recall that a problem $\Pi$ consists of a collection of (canonically encoded) instances and, for each instance $x \in \Pi$, a set $S(x)$ of solutions.

**Definition 2.3 ($NP$)** A problem $\Pi$ is in $NP$ if there is an algorithm $V$ that, given an instance $x \in \Pi$ and a purported solution $y$ as input, correctly determines whether or not $y$ is in $S(x)$ in time polynomial in the size of $x$.

We often call the algorithm $V$ the *verifier*. Since the verifier can only read one input bit per basic operation, we can assume that the length of a purported solution $y$ is at most the running time of the verifier, and hence polynomial in the length of the instance $x$.

Every problem in $NP$ can solved in exponential time by brute-force search. To see this, consider an $NP$ problem $\Pi$; by definition, it has a polynomial-time verifier $V$. Let the running time of $V$ be bounded by $cn^k$ whenever its first input $x \in \Pi$ has length $n$. There are "only" $2^{cn^k}$ possibilities for a string $y$ that could convince $V$ to accept the input $(x, y)$. Each of these possibilities can be checked by $V$ in time polynomial in the size of $x$. Checking all possibilities and outputting a

[7] An algorithm is specified by a finite list of instructions, so there are only countably many. A problem is a *set* of finite sequences, so there are uncountably many. Remarkably, there are also concrete and important problems unsolvable by any algorithm — *undecidable* is the technical term — such as the "Halting Problem", which asks whether or not a given program halts on a given input [130].

[8] The class $NP$ is traditionally defined in terms of decision problems; we work with the search problem analog, which is sometimes called $FNP$, or "functional $NP$". The letters $NP$ stand for "nondeterministic polynomial", reflecting the class's origin story via nondeterministic Turing machines. See Knuth [76] for an amusing terminological discussion.

solution (if one exists) solves the problem $\Pi$ in exponential time. A polynomial-time algorithm for an $NP$ problem represents an exponential improvement over this brute-force search algorithm.

## 2.3 Reductions and Hard Problems

Relative difficulty between two different problems is rigorously established via a *reduction*, possibly the most fundamental notion in all of complexity theory. The definition is engineered so that if one $NP$ problem $\Pi_1$ reduces to another $\Pi_2$, then a polynomial-time algorithm solving $\Pi_2$ immediately yields one solving $\Pi_1$.

**Definition 2.4 (Reduction)** A *reduction* from an $NP$ problem $\Pi_1$ to an $NP$ problem $\Pi_2$ is a pair $R_1, R_2$ of polynomial-time algorithms such that:

(a) given an instance $x \in \Pi_1$ as input, algorithm $R_1$ produces an instance $R_1(x) \in \Pi_2$ such that $R_1(x)$ has a solution if and only if $x$ does;

(b) given an instance $x \in \Pi_1$ and a solution $y \in S(R_1(x))$ to the corresponding instance of $\Pi_2$ as input, algorithm $R_2$ produces a solution $R_2(y) \in S(x)$ to the original problem.

In words, a reduction is one computationally efficient procedure that maps instances of problem $\Pi_1$ to instances of problem $\Pi_2$, and a complementary such procedure for translating solutions to the latter instances to solutions to the former ones. Moreover, even though the first procedure is unaware of whether or not the given instance has any solutions, precisely the instances of $\Pi_1$ with at least one solution are mapped to instances of $\Pi_2$ with at least one solution.[9] Reductions are a natural mathematical concept and many "non-computational" proofs implicitly contain one. Early game-theoretic examples include the reductions of Brown and von Neumann [17] and of Gale, Kuhn, and Tucker [45], from computing a Nash equilibrium of a general bimatrix game to computing a symmetric Nash equilibrium of a two-player symmetric game.

The definition of a reduction ensures that the next proposition holds.

**Proposition 2.5** *Suppose the problem $\Pi_1$ reduces to the problem $\Pi_2$. Then $\Pi_2$ is solvable by a polynomial-time algorithm only if $\Pi_1$ is.*

*Proof:* The proof is by composition and is illustrated in Figure 3. Let $(R_1, R_2)$ denote a reduction from $\Pi_1$ to $\Pi_2$ and $A$ a polynomial-time algorithm solving $\Pi_2$. Suppose the running times of $R_1$, $R_2$, and $A$ (as a function of the input size $n$) are bounded by $c_1 n^{k_1}$, $c_2 n^{k_2}$, and $c_3 n^{k_3}$, respectively. Let $B$ denote the algorithm that, given an instance $x$ of $\Pi_1$, invokes the reduction $R_1$ to obtain an instance $R_1(x)$ of $\Pi_2$; uses the algorithm $A$ to compute a solution $y \in S(R_1(x))$, or correctly determine that none exist; and uses the algorithm $R_2$ to obtain from $y$ a solution $R_2(y) \in S(x)$, or reports "no solution", if no such $y$ exists. By Definition 2.4, the algorithm $B$ solves the problem $\Pi_1$.

We complete the proof by bounding the running time of algorithm $B$. If it begins with an instance $x$ of length $n$, it first spends at most $c_1 n^{k_1}$ time producing $R_1(x)$; the length of $R_1(x)$ is necessarily at most $c_1 n^{k_1}$. Using algorithm $A$ as a subroutine to produce $y \in S(R_1(x))$, or determine that no such $y$ exists, requires at most $c_3 c_1^{k_3} n^{k_1 + k_3}$ time. This expression also bounds the size of $y$. Invoking algorithm $R_2$, if necessary, to recover a solution to $x$ from $y$ then requires

[9] What we are calling a reduction is often called a *polynomial-time reduction* or a *Karp reduction* (for search problems), to distinguish it from other types of reductions in complexity theory.

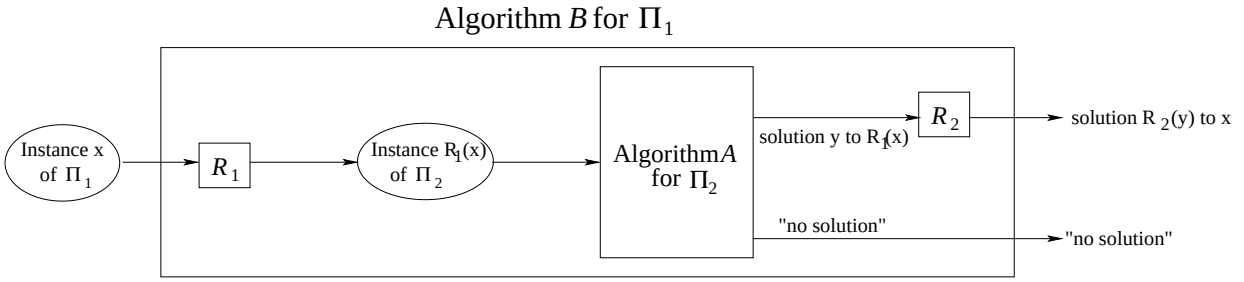



Figure 3: Proof of Proposition 2.5. If there is a reduction $(R_1, R_2)$ from the problem $\Pi_1$ to the problem $\Pi_2$, then a polynomial-time algorithm $A$ that solves $\Pi_2$ induces a polynomial-time algorithm $B$ that solves $\Pi_1$.

at most $c_2 c_3^{k_2} c_1^{k_2+k_3} n^{k_1+k_2+k_3}$ time. The overall running time of algorithm $B$ is thus polynomial in the original input size $n$, as desired. ■

Proposition 2.5 implies that if one problem reduces to another, then the latter is as hard as the former with respect to polynomial-time solvability. The next definition broadens the scope of this idea by considering classes of problems.

**Definition 2.6 ($\mathcal{C}$-Hard and $\mathcal{C}$-Complete Problems)** Let $\Pi$ be an $NP$ problem and $\mathcal{C}$ a set of $NP$ problems.

(a) The problem $\Pi$ is *$\mathcal{C}$-hard* if for every problem $\Pi' \in \mathcal{C}$, there is a reduction from $\Pi'$ to $\Pi$.

(b) The problem $\Pi$ is *$\mathcal{C}$-complete* if it is $\mathcal{C}$-hard and is also a member of $\mathcal{C}$.

Proposition 2.5 implies that a $\mathcal{C}$-hard problem is solvable by a polynomial-time algorithm only if the same is true for *every* problem in $\mathcal{C}$.

## 2.4 Existence of $NP$-Complete Problems

Proving that a problem is $NP$-hard in the sense of Definitions 2.3 and 2.6 is intimidating evidence of its intrinsic difficulty. The reason is that the class $NP$ is big: it contains every problem for which we can recognize, in a computationally efficient way, a correct solution. It comprises literally tens of thousands of important problems spanning dozens of application areas, from mathematics to biology, operations research to computer science, chemistry to economics. By Proposition 2.5, $NP$-hard problems are solvable by a polynomial-time algorithm only if all problems in $NP$ are. Put differently, if $P \neq NP$ — if there are any problems in $NP$ not solvable in polynomial time — then $NP$-hard problems admit no polynomial-time algorithm. We do not know that $P \neq NP$ — resolving this statement is one of the deepest open research problems in all of mathematics — but it is a widely adopted working hypothesis; see also Section 2.6.

But do $NP$-hard problems even exist? The answer is yes, in abundance — the Maximum Clique problem is one example, and the next section shows that computing high-payoff Nash equilibria in bimatrix games is another.

Complexity-theoretic hardness results for important problems typically require several steps. The first step is to identify a hard, possibly "unnatural" problem. Reductions are then composed to identify successively more natural hard problems, culminating with the target problem.

Toward this end, we begin with a "generic" problem $\Pi_0$ that will be $NP$-complete — both $NP$-hard and in $NP$ — almost by definition. Instances $x \in \Pi_0$ are described by four binary sequences: an encoding of a polynomial-time verifier $V$;[10] two positive integers $c$ and $k$; and a sequence $z$. By definition, the solutions $S(x)$ to such an instance $x$ are the proofs $y$ for which the verifier $V$ accepts the input $(z, y)$ within $cn^k$ elementary operations, where $n$ is the length of $z$. Thus, an instance of $\Pi_0$ poses the following task: find a solution to the instance $z$ of the $NP$ problem implicitly defined by the verifier $V$.

The problem $\Pi_0$ is in $NP$, as proved by the following polynomial-time verifier $V_0$. Given an instance of $\Pi_0$ as above and a proposed solution $y$, the algorithm $V_0$ simulates the verifier $V$ on the input $(z, y)$; this is tedious but easy to do in polynomial time given an encoding of $V$. The verifier $V_0$ accepts the purported proof $y$ (for the instance specified by $V, z, c, k$) if and only if $V$ accepts $y$ (for the instance specified by $z$) within the allotted time bound of $cn^k$ operations, where $n$ is the length of $z$.

The problem $\Pi_0$ is also $NP$-hard. Let $\Pi$ be an $NP$ problem with verifier $V$ with running time bounded by $cn^k$ on inputs of length $n$. The first algorithm $R_1$ of the reduction from $\Pi$ to $\Pi_0$ is trivial: each instance $z$ of $\Pi$ is mapped to an instance $x$ of $\Pi_0$ by prepending to $z$ encodings of $V$, $c$, and $k$. The second algorithm $R_2$ is simply the identity map. All of the requirements of Definition 2.4 are easily checked.

The generic problem $\Pi_0$ is remarkable as a proof of concept: there really is a "universal" problem that simultaneously encodes countless important and seemingly disparate problems. The problem is obviously artificial, but it serves as a launching pad for proving that thousands of more natural problems are also $NP$-complete. Specifically, *the composition of two reductions is again a reduction.* (The formal proof is similar to that of Proposition 2.5.) Thus, if an $NP$-hard problem reduces to the problem $\Pi$, then $\Pi$ is also $NP$-hard. We can therefore prove that a "natural" problem is $NP$-hard by reducing a known $NP$-hard problem, such as the generic problem $\Pi_0$, to it. Then, as the arsenal of $NP$-hard problems to reduce from grows, it becomes ever-easier to prove that additional problems are $NP$-hard. The earliest result of this type is the *Cook-Levin Theorem* [28, 81], which shows that conceptually simple and practically relevant problems in Boolean logic — for example, computing a truth assignment for the variables of a given Boolean formula so that it evaluates to "true" — are $NP$-complete.[11] Karp [71] quickly built on the Cook-Levin Theorem to show that numerous fundamental combinatorial problems, including the Maximum Clique problem, are $NP$-complete; and this in turn opened the floodgates for thousands of further reductions.[12]

[10] As with a graph, an algorithm — meaning its code in some programming language — can be canonically encoded as a sequence of zeroes and ones.

[11] The proof is via reduction from the generic problem $\Pi_0$. Given an instance of $\Pi_0$ — an encoding of a verifier $V$, constants $c$ and $k$, and an instance $z$ of length $n$ — the basic idea is to use $cn^k$ Boolean variables to represent the bits of a potential solution $y$ for $z$ (as judged by $V$) and roughly $c^2n^{2k}$ auxiliary variables to represent the internal state of $V$ throughout its execution on the input $(z, y)$. Logical constraints between different variables can be added to enforce the accurate simulation of the evolution of $V$'s internal state and that $V$ accepts the purported proof $y$ within $cn^k$ elementary steps. Solutions $y$ of $S(z)$ are then in bijective correspondence with variable assignments that meet all of these logical constraints. Details are in any of the complexity theory textbooks cited at the beginning of this section.

[12] See Garey and Johnson [47] for an outdated but terrifically useful list.

**Theorem 2.7 ([71])** *The Maximum Clique problem is $NP$-complete.*

The proof of Theorem 2.7 is not overly long (see e.g. Dasgupta, Papadimitriou, and Vazirani [30, pp. 249–250] or Kleinberg and Tardos [75, pp. 460–462]), but we skip it and instead leverage Theorem 2.7 to prove that a natural equilibrium computation problem is $NP$-complete.

## 2.5 Application: Computing a High-Payoff Nash Equilibrium is $NP$-Complete

This section proves that a natural equilibrium computation problem is $NP$-hard, by reducing the Maximum Clique problem to it. Recall that a bimatrix (i.e., two-player normal-form) game is specified by two $m \times n$ payoff matrices, with the rows and columns indicating the players' respective strategy sets. For bimatrix games with rational payoffs, a natural encoding specifies $m$, $n$, and the $2mn$ payoffs, listed in a canonical order. (A rational number can be specified by encoding its numerator, denominator, and sign separately.) Recall that a *Nash equilibrium* is a probability distribution for each player — *mixed strategies* — so that each player is maximizing its expected payoff, given the mixed strategy employed by the other player.

**Problem 2.8 (Computing High-Payoff Nash Equilibria of Bimatrix Games)**
Given (canonical encodings of) a bimatrix game $G$ with rational payoffs and a rational number $r$:

(a) if $G$ has a Nash equilibrium in which both players have expected payoff at least $r$, compute one;

(b) otherwise, indicate that $G$ has no such Nash equilibrium.

As a technical aside, we note that Problem 2.8 is an $NP$ problem. This is not immediately obvious, since there appear to be an infinite number of candidate solutions (all mixed strategy profiles). In Section 4 we outline an argument implying that, if an instance of Problem 2.8 has a solution, then it has a solution consisting of mixed-strategy profiles $x$ and $y$ for the two players made up of rational numbers whose binary encodings have length polynomial in the input size. Given such an $x$ and $y$, the Nash equilibrium and payoff requirements can be verified in time polynomial in the input size, by explicitly computing expected payoffs and checking each pure-strategy deviation by each player. This proves membership of Problem 2.8 in $NP$, and implies that it can be solved in exponential time via brute-force search. The next result gives evidence that no substantially faster algorithm exists.

**Theorem 2.9 ([51])** *Problem 2.8 is $NP$-complete.*

*Proof:* We give a reduction from the Maximum Clique problem to Problem 2.8. The first algorithm $R_1$ takes as input an encoding of a graph $H = (V, E)$ and a target $k$, and constructs an instance of Problem 2.8 as follows (see also Table 1). Each player has the same set of $2n$ strategies, where $n = |V|$. Each strategy consists of picking a vertex, and of choosing whether to be "nice" or "mean". We emphasize that the game has only one stage, and players make both of these decisions simultaneously. For strategy profiles in which both players play nice, the payoffs to both players are:

- $1 + \epsilon$ if the players choose the same vertex, where $\epsilon = 1/2k$;
- 1 if the players choose distinct vertices that are adjacent in $H$ (i.e., connected by an edge);

| | 1N | 2N | 3N | 4N | 1M | 2M | 3M | 4M |
|---|---|---|---|---|---|---|---|---|
| 1N | $1+\epsilon, 1+\epsilon$ | 1, 1 | 1, 1 | 0, 0 | $-k, k$ | 0, 0 | 0, 0 | 0, 0 |
| 2N | 1, 1 | $1+\epsilon, 1+\epsilon$ | 1, 1 | 0, 0 | 0, 0 | $-k, k$ | 0, 0 | 0, 0 |
| 3N | 1, 1 | 1, 1 | $1+\epsilon, 1+\epsilon$ | 1, 1 | 0, 0 | 0, 0 | $-k, k$ | 0, 0 |
| 4N | 0, 0 | 0, 0 | 1, 1 | $1+\epsilon, 1+\epsilon$ | 0, 0 | 0, 0 | 0, 0 | $-k, k$ |
| 1M | $k, -k$ | 0, 0 | 0, 0 | 0, 0 | 0, 0 | 0, 0 | 0, 0 | 0, 0 |
| 2M | 0, 0 | $k, -k$ | 0, 0 | 0, 0 | 0, 0 | 0, 0 | 0, 0 | 0, 0 |
| 3M | 0, 0 | 0, 0 | $k, -k$ | 0, 0 | 0, 0 | 0, 0 | 0, 0 | 0, 0 |
| 4M | 0, 0 | 0, 0 | 0, 0 | $k, -k$ | 0, 0 | 0, 0 | 0, 0 | 0, 0 |

Table 1: Proof of Theorem 2.9: the bimatrix game produced by the algorithm $R_1$ given the graph in Figure 2 as input. Each strategy is annotated with the vertex it corresponds to (1, 2, 3, or 4) and whether the player plays "nice" (N) or "mean" (M).

- 0 if the players choose distinct vertices that are not adjacent in $H$.

If both players choose the same vertex and exactly one of them plays nice, then the mean player obtains payoff $k$ and the nice player obtains payoff $-k$. In all other cases, both players receive zero payoff. This defines the game $G$; the target payoff $r$ is set to $1 + \frac{\epsilon}{k}$. This algorithm $R_1$ runs in time polynomial in the size of the given Maximum Clique instance.

Call a (symmetric) mixed strategy profile of $G$ *good* if both players always play nice and randomize uniformly over the vertices of the same $k$-clique of $H$. Our key claim is that the Nash equilibria of $G$ in which both players obtain expected payoff at least $r = 1 + \frac{\epsilon}{k}$ are precisely the good strategy profiles (if any). Given this claim, we can easily complete the reduction: the second algorithm $R_2$ takes a solution to the constructed instance of Problem 2.8, which we now know is a good strategy profile, and returns the corresponding $k$-clique of $H$. Assuming the claim is true, the algorithms $R_1$ and $R_2$ constitute a reduction in the sense of Definition 2.4.

One direction of the claim is easy. Consider a good strategy profile, with both players playing nice and randomizing uniformly over the vertices of a $k$-clique $K$. The expected payoff of each player is $\frac{1}{k} \cdot (1 + \epsilon) + \frac{k-1}{k} \cdot 1 = r$. To check that this is a Nash equilibrium, we only need to check the $2n$ pure-strategy deviations of one of the players. Unilaterally deviating and playing nice yields expected payoff at most $r$. Deviating and playing mean yields expected payoff either $\frac{1}{k} \cdot k = 1 < r$ (if a vertex of $K$ is played) or 0 (otherwise). Thus every good strategy profile is a Nash equilibrium with the desired payoffs.

To prove the converse, we establish four facts about the game $G$.

(1) For every Nash equilibrium, for each player $p$ and vertex $i \in V$, the probability that player $p$ plays nice and chooses strategy $i$ is at most $(1 + \epsilon)/k$.

(2) For every mixed-strategy profile in which both players obtain expected payoff at least $r$, each player plays nice with probability at least $1 - \epsilon$.

(3) For every Nash equilibrium in which both players obtain expected payoff at least $r$, each player randomizes over at least $k$ different vertices when playing nice.

(4) For every Nash equilibrium in which both players obtain expected payoff at least $r$, for every vertex $i$ that is played with positive probability by a player $p$ when playing nice, the other player $q$ plays nice and chooses vertex $i$ with probability at least $1/k$.

The non-trivial direction of the key claim follows easily from facts (3) and (4). By (3), in a Nash equilibrium with the desired payoffs, there are $k$ distinct vertices $K$ that the row player sometimes selects when playing nice; fact (4) then implies that the column player must always play nice and randomize uniformly over the vertices of $K$; invoking (4) again, the row player plays exactly the same way, yielding a good mixed-strategy profile.

To show the contrapositive of (1), assume that $p$ plays nice and selects strategy $i$ with probability more than $(1+\epsilon)/k$. If the other player $q$ deterministically plays mean and chooses strategy $i$, its expected payoff is more than $1+\epsilon$. Every best response by $q$ thus randomizes only over mean strategies. But then every best response by $p$ would also randomize only over mean strategies, so we did not start with a Nash equilibrium. For (2), note that whenever at least one player is mean, the sum of the payoffs to the players is 0; when both are nice, the sum of the payoffs is at most $2(1+\epsilon)$. Thus, in a mixed-strategy profile in which both players obtain expected payoff at least $r$, both players are (and hence each player is) nice with probability at least $2r/2(1+\epsilon) > 1/(1+\epsilon) > 1-\epsilon$. Assertion (3) follows immediately from (1) and (2) and our choice of $k$: under the stated hypothesis, each player must be nice with probability at least $1-\epsilon$ but can allocate at most $(1+\epsilon)/k$ probability to each of the $\ell$ nice strategies that it employs; since $\epsilon = 1/2k$, rearranging terms yields $\ell > k-1$. To establish (4) and complete the proof of the theorem, the hypothesis implies that the expected payoff of $p$ when deterministically playing nice and selecting vertex $i$ is at least $r = 1 + \frac{\epsilon}{k}$, and this is true only when the other player $q$ plays nice and selects vertex $i$ with probability at least $1/k$. ■

Theorem 2.9 is due to Gilboa and Zemel [51]. See McLennan and Tourky [85] and Hazan and Krauthgamer [60] for variants of the above reduction, and Conitzer and Sandholm [27] for an alternative proof of Theorem 2.9.

## 2.6 Discussion

Theorem 2.9 implies that, unless $P = NP$, there is no polynomial-time algorithm for deciding whether or not a bimatrix game has a high-payoff Nash equilibrium. We next discuss the most common questions about and objections to negative complexity results such as this. This section focuses on general issues common to all $NP$-hardness results. Sections 3.5 and 4.5 address points particular to equilibrium computation applications, including the economic and behavioral significance of computational hardness results such as Theorem 2.9. We also touch on some alternatives to time complexity analysis, such as space complexity.

**Who Cares About Worst-Case Analysis?** Perhaps the most common objection to the theory of $NP$-completeness is that the definition of a "good algorithm", as an algorithm that runs in polynomial time *for every possible input*, is too strong. If a problem is $NP$-complete and $P \neq NP$, then it has no computationally efficient general purpose solution (unlike, say, linear programming) — but surely the running time of an algorithm only matters on *inputs that we actually care about*? This objection is legitimate; how can we formalize it?

One option is to consider random inputs under some probability distribution; see Bárány, Vempala, and Vetta [7] and Porter, Nudelman, and Shoham [104] for results along these lines in equilibrium computation problems.[13] Modeling random inputs in a meaningful yet analytically tractable

[13] There is also a complexity theory of "hard-on-average problems", although it is less mature than that of worst-case hardness. See Bogdanov and Trevisan [12] for a recent survey.

way is difficult, however. For example, for the most popular definitions of random games and numerous well-studied classes of "structured" (and presumably "interesting") games, a random game is structured *with probability zero* [97]. Thus positive results for random games are not always meaningful for the types of games that we typically want to study.

A second approach — and usually a better one, in our opinion — is to formalize the structural constraints inherent in a class of applications. The "congestion games" of the next section are one case study of this idea. Hard problems can become easy when additional structure is imposed. For example, Problem 2.8 is polynomial-time solvable using linear programming if the bimatrix game is zero-sum. We believe that this approach should be pursued rigorously and systematically, using complexity-theoretic tools to map the tractability frontier between solvable and unsolvable special cases of a problem. We should strive for polynomial-time algorithms for "relevant" special cases, and for reductions that reach beyond contrived examples and apply to such special cases.

**Who Cares About Polynomial Time?** Even after accepting worst-case running time as a useful performance statistic, identifying "computationally efficient" with "polynomial running time" can initially seem rather arbitrary. Surely we should differentiate between higher- and lower-order polynomials? And why is an algorithm with running time $n^{100}$ "fast" and an algorithm with running time $n^{\log_2 n}$ "slow"?

Initially, the definition of polynomial time was motivated primarily by its mathematical elegance and robustness (e.g. in Edmonds [39]): it ensures that the class $P$ of computationally tractable problems is independent of the details of the computational model, that polynomial-time reductions are closed under composition (cf., the proof of Proposition 2.5), and so on. With decades of hindsight, we can now say that polynomial time has served as a remarkably effective and accurate classification tool. Problems in $P$ almost always admit general-purpose algorithmic solutions that are fast in practice. Often a slow, polynomial-time algorithm comes first, and further work produces practical algorithms with very good worst-case running times. Problems that seem to lie outside $P$, such an $NP$-hard problems, are not necessarily "unsolvable" in practice, but in almost all cases solving even moderate problem sizes requires significant human and computational effort. A case in point is the $NP$-hard Traveling Salesman Problem [5].

There is also remarkable correlation between the class $P$ and problems that have "nice mathematical structure", such as a succinct characterization of feasibility or optimality. For example, classes of mathematical programs that have a strong duality theory (such as linear and convex programming) are almost always polynomial-time solvable, while those that do not (such as general integer or nonlinear programming) are typically $NP$-hard. There are also examples in game theory, such as the close correspondence between uniqueness and tractable computational complexity for the sequential elimination of strategies under different dominance relations [50]. We revisit this point when comparing Nash and correlated equilibria of bimatrix games in Section 4.5.

**What if $P = NP$?** Most computer scientists believe that $P \neq NP$, but that the $P$ vs. $NP$ question will not be resolved soon; see Gasarch [48] for a straw poll and Aaronson [1] and Sipser [122] for much more on the issue. This is, however, only a conjecture. Indeed, the evidence for believing that $P \neq NP$ is largely psychological rather than mathematical: tens of thousands of important problems, across numerous scientific fields, are now known to be $NP$-complete. For decades, many have searched far and wide for a computationally efficient solution to at least one of these. And crucially, if even one of these problems admits a polynomial-time algorithm, *then all of them do.*

Most computer scientists do not believe that $P = NP$ because, frankly, it seems too remarkable to be true. And if one day $P = NP$ is proved, equilibrium computation will not be the problem to dominate the mainstream news headlines — that would be the $NP$ problem of factoring large integers, the presumed difficulty of which underlies the RSA cryptosystem [108] and most of modern electronic commerce.

Philosophically, $P = NP$ would suggest that *all creativity can be automated* [138]. For example, for every true proposition with an efficiently verifiable proof, such as Fermat's Last Theorem, there would be an automated way of quickly generating such a proof. As Wigderson [138] writes, "we would similarly revolt against the possibility of a monkey spewing out a perfect Shakespeare tragedy." At the very least, $P \neq NP$ seems to be a safe working hypothesis for the foreseeable future.

**Bounded Rationality and Other Complexity Notions.** We often adopt polynomial-time algorithms as surrogates for boundedly rational participants and designers (Sections 1.2, 3.5, and 4.5). *Space complexity*, which counts the maximum amount of memory required by an algorithm, is an alternative complexity measure that has informed much economic research on the bounded rationality program of Simon [121] in repeated games. The idea is to limit the strategies in a repeated game — each a function from a history-so-far to an action — to those implementable by a space-bounded algorithm. Restricting the complexity of strategies yields a modified game, which generally has a different set of equilibria than the original repeated game. For example, cooperative equilibria emerge in the repeated Prisoner's Dilemma; see Neyman [96] and Papadimitriou and Yannakakis [103] for details.

Space complexity was developed for automata [92, 107] — a highly restrictive type of algorithm — prior to the study of general time and space complexity [58, 59].[14] Automata are far easier to work with than general algorithms, as exemplified by the Myhill-Nerode Theorem [95] that characterizes the minimum space required to implement a particular strategy, and they form the basis of most works on this topic.[15] See Kalai [66] for a survey of this line of research and Rubinstein [112] for a broader discussion of bounded rationality.

These two notions of bounded rationality — polynomial-time algorithms and automata with polynomially many states — are not directly comparable. Automata appear relevant only in repeated games, while time complexity analysis is useful whenever the game can be described in a succinct way (as in both single-shot games, as studied here, and repeated games [14, 83]). Also, participants in a repeated game face two types of complexity: that of *formulating* a strategy, and that of *executing* a strategy. Permitting only strategies representable as polynomial-size automata limits the latter type of complexity without addressing the former problem, which can be $NP$-hard in some natural contexts [49, 98]. Modeling a participant as a polynomial-time algorithm, with the game description provided as input, constrains simultaneously the effort it can invest in both planning and playing the game.

There are additional useful complexity measures. To mention one more, *communication complexity* counts only the amount of information exchanged between participants and a designer, and

[14] The relationship between the time and space complexity of many problems is not completely understood, but there are some obvious connections: since an algorithm can only write one thing to memory each time step, the space it requires is no more than its running time; and since an algorithm that does not go into an infinite loop cannot return to a previously visited state (i.e., the same position in the program code with the same contents in its memory), its running time is at most exponential in its space requirement.

[15] An exception is Megiddo and Wigderson [87].

ignores computation completely. This relatively simple measure is useful for establishing negative results, in the form of lower bounds on the number of bits of communication needed to solve a problem. Game-theoretic applications include equilibrium computation in games with a large number $n$ of players, where the worst-case amount of payoff information that must be exchanged between players to reach a Nash equilibrium is exponential in $n$ [56]; and combinatorial auctions, where a mechanism for obtaining an approximately welfare-maximizing allocation must elicit an amount of preference information that is exponential in the number of goods [119].

# 3 Pure Nash Equilibria in Congestion Games

This section studies the problem of computing pure-strategy Nash equilibria in *congestion games*, where such equilibria are guaranteed to exist [109]. After reviewing the definition and properties of congestion games (Section 3.1), Section 3.2 defines a notion of hardness, weaker than $NP$-hardness, suitable for computational problems in which a solution is guaranteed to exist. Section 3.3 shows that computing a pure Nash equilibrium of a congestion game is hard in this sense, and Section 3.4 derives negative consequences for the convergence time of best-response dynamics in such games. Section 3.5 discusses economic and behavioral interpretations of these intractability results.

## 3.1 Congestion Games

A *congestion game* is an $n$-player finite game with a particularly nice payoff structure (Figure 4). There is a ground set $R$ of *resources*, and each resource $r \in R$ has a *cost function* $c_r : \{1, 2, \ldots, n\} \to \mathbb{R}$. To minimize technical details, we assume that all costs are integers. Each player $i$ has a strategy set $\mathcal{S}_i$, and each strategy $S_i \in \mathcal{S}_i$ is a non-empty subset of resources. In a strategy profile $(S_1, \ldots, S_n)$, the *cost* (negative payoff) to player $i$ is defined by

$$C_i(S_1, \ldots, S_n) = \sum_{r \in S_i} c_r(n_r),$$

where $n_r = |\{j : r \in S_j\}|$ denotes the corresponding number of players that select a strategy containing the resource $r$.

Congestion games are remarkable in that they model diverse applications — including traffic routing (e.g. [111]), network formation (e.g. [127]), firms competing for production processes [109], and the migration of animals between different habitats [88, 105] — yet always possess pure Nash equilibria. We recall the proof.

**Theorem 3.1 (Rosenthal's Theorem [109])** *Every congestion game has at least one pure Nash equilibrium.*

*Proof:* Given a congestion game, define a *potential function* $\Phi$ on its strategy profiles by

$$\Phi(S_1, \ldots, S_n) = \sum_{r \in R} \sum_{i=1}^{n_r} c_r(i). \tag{1}$$

For every strategy profile $(S_1, \ldots, S_n)$ and deviation $\tilde{S}_i$ by a player $i$, the definition of $\Phi$ ensures that

$$C_i(\tilde{S}_i, S_{-i}) - C_i(S_i, S_{-i}) = \sum_{r \in \tilde{S}_i \setminus S_i} c_r(n_r + 1) - \sum_{r \in S_i \setminus \tilde{S}_i} c_r(n_r) = \Phi(\tilde{S}_i, S_{-i}) - \Phi(S_i, S_{-i}). \tag{2}$$

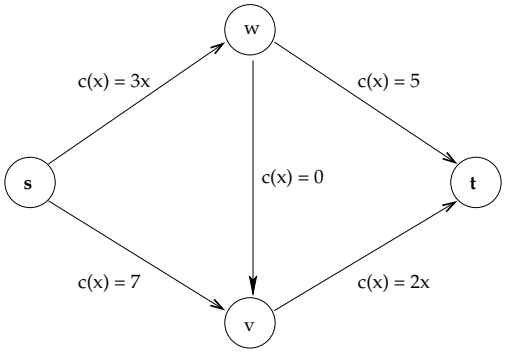



Figure 4: Example of a congestion game. Two players choose from three strategies, the directed paths from $s$ to $t$. Each edge is labeled with its cost function. In the unique pure Nash equilibrium, both players select the path $s \to w \to v \to t$ and incur cost 10.

At a global minimum $(S_1, \ldots, S_n)$ of $\Phi$ — which exists, since the game is finite — the right-hand side of (2) is non-negative for every player $i$ and deviation $\tilde{S}_i$. This profile is therefore a pure Nash equilibrium. ■

Our goal is to understand the computational complexity of the following basic problem.

**Problem 3.2 (Computing Pure Nash Equilibria of Congestion Games)**
Given a congestion game with integer costs, compute a pure Nash equilibrium.[16]

In Problem 3.2, we assume that a congestion game is described as a list of strategies for each player (and each strategy as a list of resources), followed by a binary encoding of the $n|R|$ integers that define the cost structure. The number of strategy profiles of a congestion game — and hence the size of its normal-form representation — can be exponentially larger than the size of this description.[17] Even congestion games with an extremely large number of players — the common case in network applications — can be represented practically in this way.

Given a congestion game and a purported solution to Problem 3.2 — a strategy profile — its correctness can be checked in time polynomial in the game's (natural) description, simply by examining each of the $|S_i| - 1$ possible deviations of each player $i$ in turn. Problem 3.2 is therefore an $NP$ problem. Unlike the search for high-payoff equilibria in bimatrix games (Problem 2.8), where an algorithm was required to recognize instances without a solution, Rosenthal's Theorem guarantees that every instance of Problem 3.2 has at least one solution.[18]

[16] Finding *all* pure Nash equilibria of a congestion game is only harder than finding one. Negative results are more compelling for the easier problem of computing a single equilibrium.

[17] Consider $n$ players and $n+1$ resources, where player $i$ can use either resource 0 or resource $i$. Resource 0 always has cost 0, while resource $i$ has cost $i$ if used. This game has $2^n$ (non-isomorphic) strategy profiles yet its natural description has size near-linear in $n$.

[18] The subclass of $NP$ problems with this "guaranteed existence of solutions" property is called $TFNP$, for "total functional $NP$" [86].

### 3.2 Better-Response Dynamics and $PLS$

As an $NP$ problem, Problem 3.2 can be solved via brute-force search: given a congestion game, systematically enumerate its finitely many strategy profiles, and return the first one that is a pure Nash equilibrium. There is another obvious algorithm for Problem 3.2, a form of *guided search*: *better-response dynamics*. Recall that in better-response dynamics, as long as the current strategy profile is not a pure Nash equilibrium, we allow an arbitrary player to deviate to an arbitrary strategy that strictly decreases its cost. As emphasized by Monderer and Shapley [91], equation (2) implies that every iteration of better-response dynamics in a congestion game strictly decreases Rosenthal's potential function. Better-response dynamics is thus guaranteed to terminate, necessarily at a pure Nash equilibrium of the game. Does better-response dynamics provide a polynomial-time algorithm for Problem 3.2? A byproduct of our complexity-theoretic study of this problem is a negative answer to this question.[19]

Empirically, almost all $NP$ problems are either polynomial-time solvable or $NP$-complete. So which is Problem 3.2? Our next goal is to give evidence that the answer is *neither*: Problem 3.2 belongs to the rare species of problems that seem to possess "intermediate complexity".[20]

There is a simple but subtle reason why complexity theorists do not expect problems like computing a pure Nash equilibrium of a congestion game to be $NP$-hard. Recall the Maximum Clique problem (Problem 2.1), and the reason for its membership in $NP$: a purported solution (a potential $k$-clique) can be succinctly described and quickly checked for correctness. But what if you wanted to convince someone that a graph did *not* have a $k$-clique? Could you write down a polynomial-size proof of this fact that could be verified in polynomial time?[21] Intuitively, it seems impossible to be sure that a graph has no $k$-clique without checking all or at least most of the exponentially many potential solutions. In complexity theory parlance, this is equivalent to the "$NP \neq coNP$" conjecture; it is mathematically stronger than the $P \neq NP$ conjecture, but it is believed with equal fervor.

Consider an attempt to prove that Problem 3.2 is $NP$-hard in the following sense: if it admits a polynomial-time algorithm, then all $NP$ problems do. The obvious proof approach is illustrated in Figure 5: use a reduction $R_1$ to map instances of an $NP$-hard problem like Maximum Clique to congestion games; employ a hypothetical polynomial-time algorithm $A$ for Problem 3.2 to compute a pure Nash equilibrium; and use a polynomial-time algorithm $R_2$ to infer a solution (or the non-existence thereof) to the original instance of Maximum Clique. If the reduction $(R_1, R_2)$ exists, then we obtain the desired conclusion: the polynomial-time algorithm $A$ for computing pure Nash equilibria in congestion games exists only if $P = NP$.

We claim that the existence of the reduction $(R_1, R_2)$ would resolve, negatively, the "$NP \neq coNP$" conjecture! To see this, assume that $(R_1, R_2)$ exists and consider an encoding $x$ of a graph with no $k$-clique. A pure Nash equilibrium $S$ of the congestion game $R_1(x)$ has size polynomial in $x$, and we claim that it constitutes a polynomial-time verifiable proof that $x$ has no solution; crucially, such an equilibrium exists by Rosenthal's theorem. Given $S$, we simply execute $R_1$ on $x$ to obtain a congestion game $R_1(x)$ and confirm that $S$ is a pure Nash equilibrium for it, and execute $R_2$

[19] Better-response dynamics *is* a polynomial-time algorithm in the special case of congestion games in which the potential function $\Phi$ takes on only polynomially many distinct values. This occurs when, for example, the cost of every resource is always a nonnegative integer that is bounded above by a polynomial in the number of players.

[20] The two best-known such problems are factoring large integers and determining whether or not two unlabeled graphs are isomorphic.

[21] For example, Farkas's Lemma provides such certificates for infeasible systems of linear inequalities (e.g. Vohra [134]).

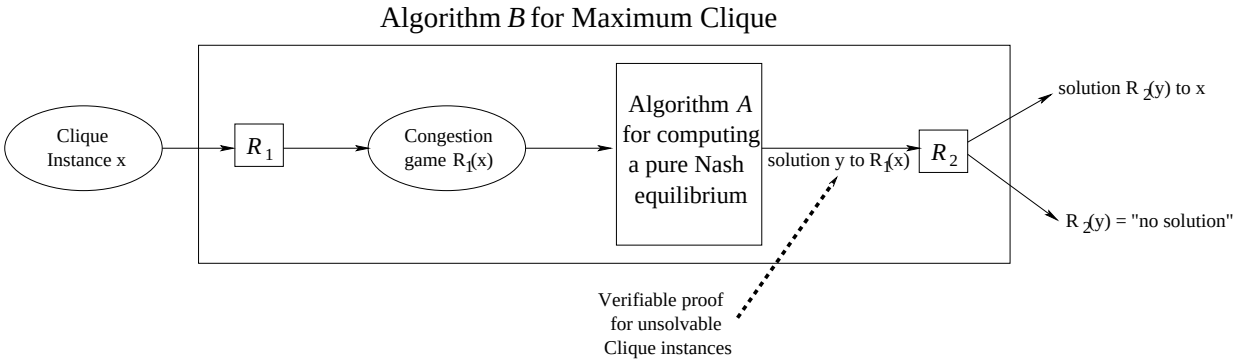



Figure 5: A reduction from the Maximum Clique problem to that of computing pure Nash equilibria in congestion games would yield short and efficiently verifiable proofs that graphs contain no large cliques, refuting the "$NP \neq coNP$" conjecture.

on $R_1(x)$ and $S$ to confirm that $R_2$'s output is "no solution". All of this can be accomplished in polynomial time, and the correctness of $R_1$ and $R_2$ imply irrefutably that the graph encoded by $x$ has no $k$-clique.

**Theorem 3.3 ([64, 86])** *Problem 3.2 is $NP$-hard only if the "$NP \neq coNP$" conjecture is false.*

The moral of Theorem 3.3 is that, if we aim to show that computing pure Nash equilibria of congestion games is $\mathcal{C}$-hard for some class $\mathcal{C}$, then $\mathcal{C}$ needs to be a subclass of $NP$. Recall that the intent of the definition of $NP$ was to capture all problems solvable by brute-force search. Problem 3.2 is apparently not among the most difficult problems in $NP$ because it also admits a guided-search algorithm, namely better-response dynamics. Could it be as hard as any problem solvable by an analogous guided search algorithm?

To answer this question formally, we define the problem class $PLS$, for "polynomial local search".[22] Recall that the definition of an $NP$ problem — a verifier for candidate solutions and an explicit polynomial bound on its worst-case running time — was motivated by the minimal information required to run brute-force search. What are the minimal ingredients for executing a local search procedure analogous to better-response dynamics? We define a $PLS$ problem in terms of encodings of *three* algorithms $A, B, C$ and explicit polynomial bounds on their running times.[23]

- Algorithm $A$ accepts an instance (e.g., an encoding of a congestion game) and outputs an initial candidate solution (e.g., the profile in which each player selects its first strategy).
- Algorithm $B$ accepts an instance and a candidate solution for it and returns the objective function value of the candidate solution according to some objective (e.g., the Rosenthal potential value).

[22] This class was originally defined to study local search problems, not equilibrium computation problems [64]. The connection between local search algorithms and better-response dynamics was made explicit by Fabrikant, Papadimitriou, and Talwar [42].

[23] We focus on minimization problems, though maximization problems can analogously be members of $PLS$ (see Yannakakis [139]).

- Algorithm $C$ accepts an instance and a candidate solution for it and either outputs a candidate solution with strictly smaller objective function value or reports that the provided solution is *locally optimal* (e.g., executes a better response for some player starting from the input strategy profile, if possible).

As in Section 2, if one of the algorithms fails to halt within the specified time bound or otherwise misbehaves, we interpret its output in some canonical way. For example, we can interpret the output of $B$ as $+\infty$ in this case. Similarly, if algorithm $C$ produces a solution no better than the one it was provided (which can be checked using algorithm $B$), we can interpret its output as a declaration of local optimality. We say that an algorithm *solves a PLS problem* if, for every instance, it returns a locally optimal solution.

By the definitions and the proof of Rosenthal's theorem, Problem 3.2 is a $PLS$ problem. A different $PLS$ problem is linear programming with a given simplex pivot rule: an instance is a linear program, candidate solutions are basic feasible solutions, and the algorithm $C$ executes an improving pivot step, if possible. Every problem in $PLS$ admits a guided search algorithm for computing a locally optimal solution: given an instance, use algorithm $A$ to obtain a start state, and iteratively apply algorithm $C$ until a locally optimal solution is reached. Since the objective function values of the candidate solutions strictly decrease until a locally optimal solution is found, and since there are only finitely many distinct candidate solutions, this procedure eventually terminates. As the number of candidate solutions can be exponential in their encoding length — just as the number of strategy profiles can be exponential in the description length of a congestion game — this guided search procedure need not run in polynomial time.

We emphasize that solving a $PLS$ problem simply means computing, by whatever means, some locally optimal solution. An algorithm is *not* constrained to follow the guided search paradigm. For example, one can optimally solve a linear programming problem — yielding a global optimum, which is locally optimal with respect to every pivot rule — in polynomial time using the ellipsoid method [73] or interior-point methods [70]. An algorithm is not even required to return the same locally optimal solution that would be computed by guided search. In Problem 3.2, *every* pure Nash equilibrium of the given congestion game, whether or not it is reached by better-response dynamics from a canonical initial state, is a legitimate output.

### 3.3 Example: Computing a Pure Nash Equilibrium of a Congestion Game is $PLS$-Complete

This section shows that computing a pure Nash equilibrium of a congestion game (Problem 3.2) is $PLS$-hard (and hence $PLS$-complete).

**Theorem 3.4 ([42])** *Problem 3.2 is PLS-complete.*

Thus, Problem 3.2 is solvable by a polynomial-time algorithm if and only if all problems in $PLS$ are.

Our proof that Problem 2.8 — computing high-payoff Nash equilibria in bimatrix games — is $NP$-hard (Theorem 2.9) implicitly required several steps. We established a "generic" $NP$-hard problem; used the Cook-Levin Theorem [28, 81] to reduce it to a problem in Boolean logic (proving the latter $NP$-hard); relied on Karp's reduction from the Cook-Levin problem to the Maximum Clique problem [71]; and finally reduced the Maximum Clique problem to Problem 2.8. We outline a similar approach to proving Theorem 3.4.

$PLS$ also has a "generic" complete problem; the argument is similar to but slightly more technical than that in Section 2.4, and is left to the reader. The analog of the Cook-Levin Theorem, proved by Johnson, Papadimitriou, and Yannakakis [64], states that a particular local search problem for Boolean circuits called "Circuit Flip" is $PLS$-hard. Circuit Flip has been reduced to a number of other local search problems, proving them $PLS$-hard as well.

We make use of the following *Minimum Agreement problem*. An instance of Minimum Agreement consists of $n$ *variables* $x_1, x_2, \ldots, x_n$ and $m$ unordered *triples*, each consisting of three distinct variables. Each triple also has a positive integral *cost*. An *assignment* gives each variable the value 0 or 1. The *cost* of an assignment is the sum of the costs of the triples in which all three variables are assigned the same value. For example, an assignment has cost 0 if and only if each triple has at least one variable assigned to each of "0" and "1". The local optima of a Minimum Agreement instance are defined as the assignments whose cost cannot be decreased by changing the value of a single variable. In $PLS$ terms, the first algorithm $A$ outputs the all-zero assignment (say) of a given Minimum Agreement instance; the second $B$ returns the cost of the proposed assignment; and the third $C$ flips the value of exactly one variable to obtain a new assignment with smaller cost. If the third algorithm finds no such improved assignment, it declares local optimality; if there are several, it chooses one arbitrarily. Schäffer and Yannakakis [117] proved, by a sophisticated sequence of reductions, that the Minimum Agreement problem is $PLS$-hard.[24]

*Proof of Theorem 3.4:* We reduce the Minimum Agreement problem to Problem 3.2. The first algorithm $R_1$ of the reduction accepts a Minimum Agreement instance, with variables $x_1, \ldots, x_n$, triples $A_1, \ldots, A_m$, and costs $a_1, \ldots, a_m$, and constructs the following congestion game. There are $2m$ resources, with two resources $s_j, t_j$ per triple $A_j$. There are $n$ players, and each player $i$ has two strategies: $S_i = \{s_j \,|\, x_i \in A_j\}$ and $T_i = \{t_j \,|\, x_i \in A_j\}$. Finally, the cost functions $c_j$ of the resources $s_j$ and $t_j$ are both defined as $c_j(1) = c_j(2) = 0$ and $c_j(3) = a_j$. (Each resource belongs to the strategies of only three different players.) This construction can be implemented in polynomial time.

The second algorithm $R_2$ of the reduction should accept a pure Nash equilibrium of the congestion game constructed by $R_1$ and recover, in polynomial time, a locally optimal solution of the original Minimum Agreement instance. There is a natural bijection between assignments for a Minimum Agreement instance and strategy profiles in the corresponding congestion game, where we identify strategies $S_i$ and $T_i$ with the assignments $x_i = 1$ and $x_i = 0$, respectively. The Rosenthal potential function value of a strategy profile equals the cost of the assignment it corresponds to. The mapping thus induces a bijection between improving unilateral deviations in the congestion game and cost-decreasing variable flips in the Minimum Agreement instance. The algorithm $R_2$ can easily recover a locally optimal assignment from a pure Nash equilibrium via this mapping. ■

Section 2.6 discussed the value of extending the reach of a hardness result to important special cases. In that spirit, we consider Problem 3.2 restricted to *symmetric* congestion games, in which all players have the same strategy set.

**Problem 3.5 (Computing Pure Nash Equilibria of Symmetric Congestion Games)** Given a symmetric congestion game with integer costs, compute a pure Nash equilibrium.

A second reduction shows that even this special case is a $PLS$-hard problem.

[24] The Minimum Agreement problem is an equivalent minimization version of the "POS NAE 3SAT" problem studied in Schäffer and Yannakakis [117].

**Theorem 3.6 ([42])** *Problem 3.5 is PLS-complete.*

*Proof Sketch:* Via a reduction from the more general Problem 3.2. Given a congestion game with $n$ players and arbitrary strategy sets $\mathcal{S}_1, \ldots, \mathcal{S}_n$, the reduction constructs the following "symmetrized version". The player set remains the same. The old resource set is augmented by $n$ additional resources $r_1, \ldots, r_n$. The cost function of each of these is defined to be zero if used by only one player, and extremely large if used by two or more. Each strategy of $\mathcal{S}_i$ is supplemented by the resource $r_i$, and any player can adopt any one of these augmented strategies. The key insight is that at a pure Nash equilibrium of the constructed symmetric game, each player will adopt the identity of exactly one player from the original (asymmetric) game. This follows from the large penalty incurred by two players that choose strategies that share one of the new resources. Such a pure Nash equilibrium is then easily mapped to one of the original asymmetric game. ■

## 3.4 Consequences for Better-Response Dynamics

If $PLS \not\subseteq P$, then no polynomial-time algorithm is guaranteed to compute a pure Nash equilibrium of a congestion game. In particular, this would imply that better-response dynamics cannot always converge in polynomial time.

Something much stronger is true: hardness results for $PLS$ problems imply strong lower bounds on the worst-case running time of local search algorithms. For example, Schäffer and Yannakakis [117] proved that the standard guided search algorithm can require exponential time in a Minimum Agreement instance, visiting nearly all possible assignments. Precisely, they showed that there is a constant $c > 0$ such that, for infinitely many $n$, there is an instance of Minimum Agreement with $n$ variables such that guided search requires at least $2^{cn}$ iterations to reach a locally optimal assignment. This lower bound applies to every method of breaking ties between competing improving variable flips.

The proof of Theorem 3.4 constructs a bijection between the possible trajectories of better-response dynamics in certain congestion games and guided search in instances of Minimum Agreement. This yields an immediate lower bound on the convergence time of better-response dynamics in congestion games.

**Corollary 3.7** *There is a constant $c > 0$ such that for arbitrarily large $n$, there is an $n$-player congestion game with binary strategy sets and an initial strategy profile from which better-response dynamics requires $2^{cn}$ iterations to converge to a pure Nash equilibrium, no matter how players are selected in each step.*

We emphasize that Corollary 3.7 holds even in the (arguably unlikely) event that $PLS \subseteq P$. In exploring the limits of arbitrary polynomial-time algorithms, complexity theory thus provided a concrete answer to a natural game-theoretic question: whether or not better-response dynamics is guaranteed to find a pure Nash equilibrium faster than unguided enumeration in congestion games. For further results on computing pure Nash equilibria and the convergence of better-response dynamics in congestion games, Vöcking [133] and Dunkel and Schulz [37].

## 3.5 Discussion

This section considers the implications of Theorems 3.4 and 3.6 and Corollary 3.7. Naturally, the discussion in Section 2.6 about worst-case analysis and polynomial-time algorithms is equally

relevant here. We next focus on possible behavioral interpretations of these results and the common objections to such interpretations.

**Behavioral Implications.** Rosenthal's Theorem — the rare existence proof that comes equipped with a convergence result — suggests that pure Nash equilibria could form a credible prediction of the outcome of rational behavior in congestion games. Theorem 3.4 and Corollary 3.7 cast doubt on this interpretation. Certainly, under the strong assumption that players act according to better-response dynamics, Corollary 3.7 implies that they need not reach an equilibrium in their lifetimes, assuming at least a moderate number of players. More generally, no matter what kind of dynamics are adopted, Theorem 3.4 implies that reaching a pure Nash equilibrium requires solving a $PLS$-hard problem, and evidence suggests that this cannot always be done in polynomial time.

So is the pure Nash equilibrium a "bad" solution concept for congestion games? In the author's experience, economists can be reticent to accept complexity-theoretic critiques of equilibrium concepts. Many (but not all) of the reasons for this skepticism are well founded. The first rebuttal of such a critique is that "interesting" congestion games possess structure beyond those produced in the proofs of Theorems 3.4 and 3.6. As we argued in Section 2.6, this is often a legitimate objection and deserves to be made formal by defining interesting special cases and rigorously proving positive complexity results — polynomial-time convergence of better-response dynamics, or at least a polynomial-time algorithm for computing pure Nash equilibria. For example, Ackermann, Röglin, and Vöcking [3] show that better-response dynamics converges quickly provided players' strategy sets are "well-structured" in a precise sense. The second potential escape is to argue that players can quickly reach outcomes that are "as good as an equilibrium", and exponential convergence is required only for reaching an exact pure Nash equilibrium. A third tack would be to question the assumption that $PLS \not\subseteq P$. These two reasonable objections are addressed below. We also discuss another common complaint that elicits little sympathy from theoretical computer scientists: the possibility that the collective behavior of players in a game cannot be efficiently simulated by a computer.

**Alternative Dynamics and Approximate Equilibria.** Players in "real-life games" may play only an approximation of an equilibrium. Could "approximate pure Nash equilibria" be easier to compute than exact equilibria in congestion games? There has been a lively back and forth on this question in the recent theoretical computer science literature, which we briefly survey next.

For concreteness, we restrict attention to congestion games with nonnegative, nondecreasing cost functions. We define an *$\epsilon$-pure Nash equilibrium* to be a strategy profile such that no player can decrease its cost by more than a multiplicative factor of $1+\epsilon$ via a unilateral deviation. In *$\epsilon$-better-response dynamics*, as long as the current strategy profile is not an $\epsilon$-pure Nash equilibrium, an arbitrary player deviates to a strategy that reduces its cost by at least a factor of $1+\epsilon$. On the positive side, Chien and Sinclair [23] proved that in every symmetric congestion game (cf., Problem 3.5) with cost functions that satisfy a weak Lipschitz condition, $\epsilon$-better-response dynamics converges to an $\epsilon$-pure Nash equilibrium in time polynomial in the size of the description of the game and in $1/\epsilon$. This result makes a compelling argument for the approximate pure Nash equilibrium as a credible solution concept in symmetric congestion games.[25] On the negative side, Skopalik and Vöcking [124] prove that computing even an $\epsilon$-pure Nash equilibrium is $PLS$-hard in asymmetric

[25] Even stronger convergence results, which assume little about the precise form of the dynamics, are possible with a continuum of players; see Sandholm [113] for details.

congestion games, and that $\epsilon$-better-response dynamics can require an exponential number of steps to converge in such games. This $PLS$-hardness result implies that *any* dynamics for reaching an approximate equilibrium that requires only polynomial-time computation per iteration does not always converge in a polynomial number of iterations, unless $PLS \subseteq P$. All of these positive and negative results are robust and hold for several notions of approximate pure Nash equilibria and variations on $\epsilon$-better-response dynamics.

**What if $PLS \subseteq P$?** The consequences of $P = NP$ would be revolutionary; those of $PLS \subseteq P$ only unexpected and very surprising. In addition, since less effort has been expended on solving $PLS$-hard problems than $NP$-hard ones, it is more conceivable that $P \supseteq PLS$ than $P = NP$. That said, a proof that $PLS \subseteq P$ would be remarkable: it would essentially imply that, given only an abstract description of a $PLS$ problem in terms of three algorithms (recall Section 3.2), there is a generic, problem-independent way of finding a "shortcut" to a locally optimal solution, exponentially faster than rote traversal of the path suggested by the guided search algorithm. In particular, it would give a fundamentally new polynomial-time algorithm for solving linear programs. Even if $PLS \subseteq P$, the scientific community seems far from a proof of this; and such a proof would have to be novel in a specific technical sense.[26]

**Simulating "The Invisible Hand".** Perhaps humans are qualitatively better at computing equilibria than polynomial-time algorithms? There is no empirical evidence of this, and a possible explanation is provided by the following simulation argument. First consider a single individual and their (computationally efficient) decision to adopt a certain strategy in a game. We may not understand this process scientifically (yet), but there is no reason to suspect that it cannot in principle be efficiently simulated, or at least approximated for the purposes of equilibrium computation, by a computer program. Given this assumption, a much smaller leap of faith implies that the collective decisions of a group of individuals can also be efficiently simulated. Group behavior amounts to a sequence of interactions between and local decisions by the participants. If the decision-making process of each individual can be simulated in polynomial time, then so can every group interaction of polynomial length between a polynomial number of individuals (with a larger polynomial bound).

# 4 Mixed Nash Equilibria in Bimatrix Games

Is there a computationally efficient version of Nash's Theorem? This section surveys recent hardness results that suggest a negative answer, even in the special case of two-player games. After casting equilibrium computation in bimatrix games as a finite problem (Section 4.1), we briefly review the structure of the Lemke-Howson algorithm in Section 4.2. This form of "guided search" motivates the complexity class $PPAD$, which we define in Section 4.3. Section 4.4 touches on the recent proofs that computing a Nash equilibrium of a bimatrix game is a $PPAD$-complete problem. Section 4.5 provides interpretations, discusses positive and negative results for special cases, and contrasts the intractability of computing Nash equilibria with the tractability results known for computing approximate Nash and correlated equilibria.

[26] The formal term is *non-relativizing*; see Beame et al. [8] and Fortnow [43] for details.

## 4.1 Preliminaries

We consider bimatrix games described by two $m \times n$ matrices $A, B$ with rational payoffs, as in Section 2.5. Nash's Theorem [93] guarantees that every such game admits a *(mixed-strategy) Nash equilibrium* — a pair $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$ of probability distributions that each maximize the expected payoff of the respective player given the strategy adopted by the other player. Mathematically, $x^T Ay \geq \hat{x}^T Ay$ and $x^T By \geq x^T B\hat{y}$ for all alternative mixed strategies $\hat{x} \in \mathbb{R}^m, \hat{y} \in \mathbb{R}^n$.

We study the following fundamental problem.

**Problem 4.1 (Computing Mixed Nash Equilibria of Bimatrix Games)**
Given a bimatrix game with rational payoffs, compute a Nash equilibrium.

We obtain the most interesting and compelling results by asking for only one of the possibly many Nash equilibria of a game; the general problem of finding all equilibria is only harder [27, 51]. The problem also appears to be significantly harder in finite games with three or more players [41].

Despite its seemingly infinite solution space, Problem 4.1 admits a finite brute-force search algorithm. This fact is a consequence of the following claim: given the supports of a Nash equilibrium — the rows $R$ and columns $C$ that are played with non-zero probability — an equilibrium can be recovered in polynomial time.

To prove this claim, consider the problem of computing a mixed strategy for the column player, mixing only over the given set $C$ of columns, so that all of the rows of $R$ are best responses by the row player. This amounts to finding non-negative real numbers $\{p_j\}_{j \in C}$, summing to 1, and a real number $\lambda$ such that the expected payoff (with respect to the $p_j$'s) of playing each row of $R$ equals $\lambda$, and also the expected payoff of playing each row outside $R$ is at most $\lambda$. This system of equations and inequalities is linear in the $p_j$'s and $\lambda$, and can be solved in polynomial time — via linear programming or more direct methods [136] — and the computed solution (if any) necessarily has size polynomial in that of the system. An analogous linear system can be used to efficiently compute a mixed strategy for the row player, mixing only over the rows of $R$, such that all of the columns of $C$ are best responses of the column player. If $R$ and $C$ are the supports of a Nash equilibrium, then the two corresponding mixed strategies provide feasible solutions to the two linear systems (in a Nash equilibrium, players mix only over pure best responses). Conversely, if both linear systems have feasible solutions, then the constraints ensure that the two solutions form a Nash equilibrium of the game, as every mixture of pure best responses is also a best response.

This claim implies that Problem 4.1 is an $NP$ problem and admits a brute-force search algorithm: systematically enumerate all (exponentially many) pairs $(R, C)$ of possible supports until the two corresponding linear systems both have feasible solutions, and return this pair of solutions. Nash's Theorem [93] implies that at least one choice of supports is that of a Nash equilibrium, and at this point the algorithm will recover a Nash equilibrium and terminate.[27]

## 4.2 Guided Search via the Lemke-Howson Algorithm

Since instances of Problem 4.1 are guaranteed to possess a solution that can be verified in polynomial time, the proof of Theorem 3.3 shows that this problem is $NP$-hard only if the "$NP \neq coNP$" conjecture is false. To show that computing a Nash equilibrium of a bimatrix game is a hard problem

[27] A similar argument, which supplements each of the two linear systems above with an extra constraint lower bounding the expected payoff $\lambda$, fulfills the promise in Section 2.5 that computing high-payoff Nash equilibria in bimatrix games (Problem 2.8) is an $NP$ problem.

in some sense, we need to identify a suitable problem class that is a subset of $NP$. We addressed this same issue for computing pure Nash equilibria in congestion games by noting that Rosenthal's potential yields a guided search algorithm, and by adopting the class $PLS$ of all problems solvable by a similar algorithm. But the existence of solutions to instances of Problem 4.1 is guaranteed by a fixed-point theorem, not a potential argument — where is the guided search algorithm?

An alternative proof of Nash's Theorem in the special case of bimatrix games is evidently a form of guided search: *the Lemke-Howson algorithm* [80].[28] We next briefly recall the essential combinatorial structure of the Lemke-Howson algorithm; details are in, for example, von Stengel [135]. Consider a bimatrix game described by $m \times n$ payoff matrices $A$ and $B$; we assume that all payoffs are positive, shifting them if necessary. We use the names $\{1, 2, \ldots, m\}$ and $\{m+1, m+2, \ldots, m+n\}$ for the strategies of the row and column player, respectively. The Lemke-Howson algorithm boils down to traversing a path in a graph $G$. To define the graph we make use of two polytopes, $P = \{x \in \mathbb{R}^m : x \geq \mathbf{0}, B^T x \leq \mathbf{1}\}$ and $Q = \{y \in \mathbb{R}^n : Ay \leq \mathbf{1}, y \geq \mathbf{0}\}$, where $\mathbf{0}$ and $\mathbf{1}$ denote the all-zero and all-ones vectors (of the appropriate dimensions). These polytopes are non-empty and, since all payoffs are positive, they are bounded. *Vertices* will always refer to "corners" of these polytopes; the graph $G$ will consist of *nodes* (and edges).

Each node of $G$ corresponds to a certain type of pair of vertices of $P$ and $Q$. Each of $P$ and $Q$ is defined by $m+n$ inequalities, which are naturally associated with the strategies of both players. Associate each vertex of $P$ and $Q$ with the inequalities that are binding at that point (a subset of $\{1, 2, \ldots, m+n\}$, called *labels*). For example, the vertex $\mathbf{0}$ of $P$ receives the labels $\{1, 2, \ldots, m\}$, the vertex $\mathbf{0}$ of $Q$ the labels $\{m+1, m+2, \ldots, m+n\}$. Now consider vertices $x \in P$ and $y \in Q$ that are *completely labeled* in the sense that the union of their label sets is $\{1, 2, \ldots, m+n\}$. This occurs if $x = \mathbf{0}$ and $y = \mathbf{0}$, but suppose that at least one (and hence both) are nonzero. We can scale $x$ and $y$ by $\lambda, \mu > 0$ such that $\lambda x, \mu y$ are mixed strategies for the row and column player, respectively. Consider, for example, pure strategy $i$ of player 1. If $y$ received the label $i$, then strategy $i$ is a best response to $\mu y$ with expected payoff $\mu$. If $x$ received the label $i$, then $x_i = 0$ and strategy $i$ is not employed by the mixed strategy $\lambda x$. More generally, since $x, y$ are completely labeled, every strategy that is played with non-zero probability in $(\lambda x, \mu y)$ is a best response to the other player's strategy. The converse argument also holds, so *a non-zero pair $(x, y)$ of vertices is a Nash equilibrium if and only if it is completely labeled.*

The nodes of $G$ are the pairs $(x, y)$ of vertices of $P \times Q$ that are either completely labeled or *almost completely labeled* in the sense that every label but the first is present at $x$ or $y$. There is an edge between two nodes $(x_1, y_1)$ and $(x_2, y_2)$ if and only if they are the same in one component and differ by a "pivot step", in the sense of the Simplex algorithm [29], in the other. Traversing an edge in $G$ can be thought of as traversing an edge (a one-dimensional face) in either $P$ or $Q$, or as exchanging one binding inequality (i.e., one label) for another. An almost completely labeled pair of vertices must have a duplicate label (since $x$ and $y$ are vertices, they have at least $m+n$ labels between them), and a duplicate label suggests a corresponding pivot step to another almost or completely labeled pair of vertices. If $A$ and $B$ are "nondegenerate" in a precise sense, then completely labeled nodes of $G$ have one incident edge (corresponding to the first label) while almost completely labeled nodes have two incident edges (corresponding to a unique duplicate label).[29]

[28] For the problem of computing an *approximate* Nash equilibrium, as defined in Section 4.5, guided search algorithms follow from the path-following proofs of Sperner's Lemma (as by Cohen [26] or Scarf [115]) that can be used to prove Brouwer's fixed-point theorem. This approach also extends to the problem of computing an approximate Nash equilibrium in a game with more than two players [32, 99].

[29] Precisely, each vertex of $P$ and $Q$ should have no more than $m$ and $n$ labels, respectively. This condition is

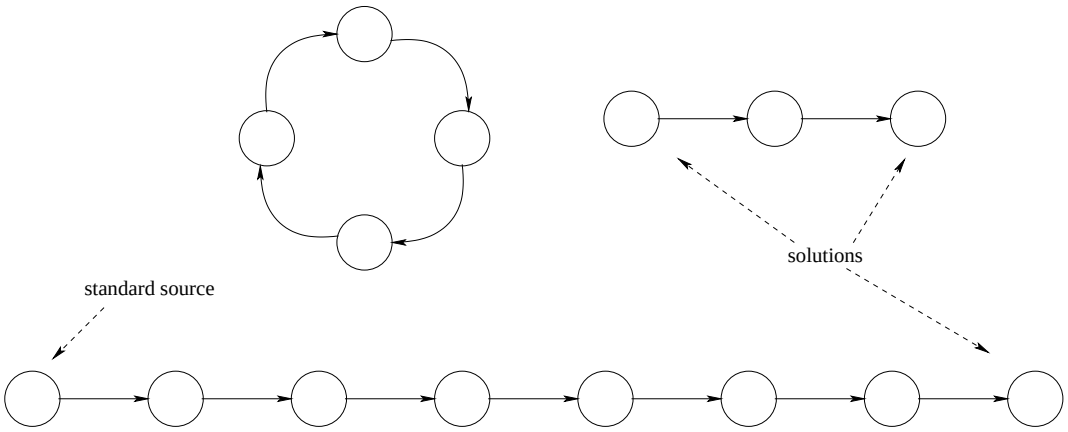



Figure 6: The structure of the Lemke-Howson graph, and more generally of the guided search paths and cycles in a $PPAD$ problem.

The graph $G$ is thus a collection of (finite) paths and cycles (Figure 6). The completely labeled node $(\mathbf{0}, \mathbf{0})$ is at one end of such a path, and the Lemke-Howson algorithm simply traverses this path to reach the Nash equilibrium given by the completely labeled node at the other end.

### 4.3 The Class $PPAD$

We are finally in a position to define the appropriate class of problems to characterize the computational complexity of computing a Nash equilibrium of a bimatrix game (Problem 4.1). To review, the class $NP$ of problems with polynomial-time verifiable solutions seems too big, because the Lemke-Howson algorithm shows that every instance of Problem 4.1 has at least one solution (cf., Theorem 3.3). The class $P$ of polynomial-time solvable problems seems too small, because all of the known algorithms for Problem 4.1, including the Lemke-Howson algorithm, require exponential time in the worst case [114]. The class $PLS$ does not appear relevant for Problem 4.1 because the form of guided search in the Lemke-Howson algorithm differs from better-response dynamics in congestion games (Theorem 3.1) — it is not guided by an objective function akin to Rosenthal's potential, and might cycle if initialized at a non-standard starting point. We therefore need to introduce a problem class called $PPAD$; the letters stand for "polynomial parity argument, directed version" [99]. See Figure 7 for a review of the complexity classes discussed in this survey, of which $PPAD$ is the final one.

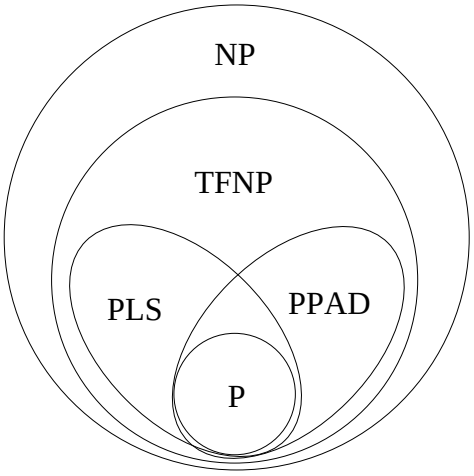
Analogous to our definition of $PLS$ in Section 3.2, the point of the class $PPAD$ is to formalize the class of all problems solvable by a path-following algorithm like the Lemke-Howson algorithm. The "directed" qualifier in $PPAD$ is motivated by a non-obvious fact: the graph $G$ searched by the Lemke-Howson algorithm is naturally directed. This is intuitively clear for the path traversed by the Lemke-Howson algorithm (which is directed away from $(\mathbf{0}, \mathbf{0})$); using the signs of subdeterminants of the payoff matrices as a guide, the other paths and cycles of the graph can also be oriented in a consistent fashion (see Todd [128]). In other words, even an amnesiac variant of the Lemke-Howson algorithm, which cannot recall the nodes already seen, always has a unique "next node" to visit.[30]

enforceable by suitable tie-breaking in determining the "leaving variable" in a pivot step.

[30] There are also problems — the possibly larger class called $PPA$ [99] — that are solvable by path-following

| Class | Informal Definition | Complete Problem |
|---|---|---|
| $P$ | Polynomial-time solvable | Not discussed in this survey |
| $NP$ | Solutions are polynomial-time verifiable | High-Payoff NE in Bimatrix Games |
| $TFNP$ | Solutions are guaranteed to exist | None known to exist |
| $PLS$ | Solvable by local search | Pure NE in Congestion Games |
| $PPAD$ | Solvable by directed path-following | Mixed NE in Bimatrix Games |

(a) Recap of Complexity Classes




(b) Suspected Relationships

Figure 7: Complexity classes studied in this survey. In (a), NE stands for "Nash equilibria". In (b), the inclusions $PLS \cup PPAD \subseteq TFNP \subseteq NP$ follow from the definitions. The inclusion $P \subseteq PLS \cap PPAD$ can be proved by representing a problem in $P$ as a degenerate type of $PLS$ or $PPAD$ problem, and by interpreting the claim "no solution" as an (efficiently verifiable!) solution to the problem. No other relationships between the five classes are known.

What are the minimal ingredients to execute a Lemke-Howson-like path-following procedure? We define a $PPAD$ problem via encodings of three algorithms $A, B, C$ and explicit polynomial bounds on their running times.

- Algorithm $A$ accepts an instance (e.g., an encoding of a bimatrix game) and outputs an initial candidate solution, called the *standard source* (e.g., the $(\mathbf{0}, \mathbf{0})$ node of the Lemke-Howson graph).
- Algorithms $B$ and $C$ each accept a candidate solution and return another, the *predecessor* and *successor* solutions, respectively (e.g., the nodes of the Lemke-Howson graph obtainable from the given one via a pivot step, oriented appropriately).

Such a trio of algorithms defines a directed graph for every possible instance. The nodes are all candidate solutions, and the directed edge $(u, v)$ is present if and only if $u$ claims $v$ as its successor

algorithms but do not seem to possess this additional orientation property. In general, our current understanding of the computational complexity of problems with guaranteed solutions, including several equilibrium computation problems, lacks the clarity of the "$P$ vs. $NP$-hard" dichotomy that classifies almost all natural problems in many application domains. See Johnson [63] and Papadimitriou [99] for further discussion.

($C(u) = v$) *and* $v$ claims $u$ as its predecessor ($B(v) = u$). We ignore the output of $B$ on the standard source $s$ to ensure that it has no incoming arc, and insist that $B(C(s)) = s$, so that $s$ has an outgoing arc. This graph is then a collection of paths and cycles, with at least one path (with the standard source at one end); see Figure 6. In the spirit of the Lemke-Howson graph, the *solutions* of an instance of a $PPAD$ problem are all of the sinks of this graph (nodes without outgoing arcs), as well as all of the sources (nodes without incoming arcs) other than the standard one. We say that an algorithm *solves a $PPAD$ problem* if, for every instance of the problem, it computes a solution. Every $PPAD$ problem can be solved, in particular, by the corresponding guided search procedure: given an instance, use the algorithm $A$ to obtain the standard source; traverse the path it initiates using algorithms $B$ and $C$, halting when a sink vertex is reached. By definition, computing Nash equilibria in bimatrix games (Problem 4.1) is a $PPAD$ problem — the three algorithms $A, B, C$ are instantiated as outlined above. The corresponding guided search algorithm in this case is precisely the Lemke-Howson algorithm. The length of the path traversed by guided search can be exponential in the size of the instance, even in the specific case of the Lemke-Howson algorithm [114], so the guided search procedure that solves a $PPAD$ problem is not generally a polynomial-time algorithm.

## 4.4 Example: Computing a Nash Equilibrium of a Bimatrix Game is $PPAD$-Complete

Since a Nash equilibrium of a bimatrix game can be computed by a directed path-following algorithm (the Lemke-Howson algorithm), it is a $PPAD$ problem. A sequence of recent papers proved that Problem 4.1 is in fact a *universal* such problem, in that every $PPAD$ problem reduces to it.

**Theorem 4.2 ([22, 33])** *Problem 4.1 is $PPAD$-complete.*

Thus, Problem 4.1 is solvable by a polynomial-time algorithm only if all problems in $PPAD$ are. But is $PPAD$ an interesting problem class? After all, its definition might appear so stylized as to contain Problem 4.1 and nothing else. Papadimitriou [99] showed that it also contains several problems related to combinatorial topology, including the problem of computing approximate fixed points of Brouwer functions, where membership in $PPAD$ follows from path-following arguments as in Cohen [26] and Scarf [115]. Computing approximate fixed points is a hard problem in a particular precise sense (Section 4.5), and a $PPAD$-hard problem inherits this evidence of intractability.

The overall proof approach to Theorem 4.2, outlined below, was developed by Daskalakis, Goldberg, and Papadimitriou [32, 52] and used to prove that the problem of computing approximate Nash equilibria in games with four or more players is $PPAD$-hard. This approach was quickly refined [18, 34], culminating in the proof of Chen and Deng [19] that Problem 4.1 is $PPAD$-hard (and hence, $PPAD$-complete). See Johnson [63] and Papadimitriou [101] for more detailed accounts of this development.

Theorem 4.2 effectively encodes a completely abstract path-following problem, where the (possibly exponential-length) path is only implicitly described via three supplied algorithms, as Nash equilibrium computation in bimatrix games. Any such encoding argument is necessarily technical. By contrast, our proof of Theorem 2.9 stood on the shoulders of the Cook-Levin Theorem and a reduction of Karp, which replaced the generic $NP$-complete problem with the conceptually much simpler Maximum Clique problem. Similarly, Theorem 3.4 did not need to work directly with a generic $PLS$-complete problem, because the heavy lifting was done by Schäffer and Yan-

nakakis [117], who proved that the far more concrete Minimum Agreement problem is also $PLS$-complete. Problem 4.1 is the first known $PPAD$-complete problem that does not explicitly supply a description of an arbitrary polynomial-time algorithm as part of the input; thus Theorem 4.2 is also remarkable from a purely complexity-theoretic perspective.

It is not clear how to encode directly a generic path-following problem as Nash equilibrium computation in bimatrix games, so the proof of Theorem 4.2 relies on a long sequence of reductions. The proof begins with a "generic $PPAD$-complete problem" (cf., Section 2.4): given a description of an abstract $PPAD$ problem, in terms of encodings of the three defining algorithms and bounds on their running times, and an instance of this problem, compute a solution — a sink or non-standard source in the directed graph implicitly defined by the three algorithms for the supplied instance.

We now outline, at a very high level, the sequence of reductions; details are in Daskalakis, Goldberg, and Papadimitriou [33] and Chen, Deng, and Teng [22]. The first step is a standard (e.g. [100, Theorem 8.1]), if tedious, transformation of the polynomial-time algorithms described in an instance of the generic problem into polynomial-size *Boolean circuits*.[31] This transformation reduces the generic $PPAD$-complete problem to a more convenient version called the "End-of-the-Line" problem [33]: given encodings of three circuits $X, Y, Z$, which play exactly the same roles as the three algorithms $A, B, C$ of a $PPAD$ problem, and a problem instance, compute a sink or non-standard source of the corresponding implicitly described directed graph.[32]

The second reduction involves a restricted type of fixed-point computation problem called Cubical Brouwer. An instance of this problem is defined by a single Boolean circuit $W$, with $3n$ 0-1 inputs and two 0-1 outputs. Such a circuit can be interpreted as a function from the unit cube to itself in the following way. The inputs specify a point $x \in \{0, 1/(2^n-1), 2/(2^n-1), \ldots, 1\}^3$, and we interpret $f(x)$ as $x$ plus one of four possible displacements, as encoded by the two corresponding outputs: $\delta_0 = (-\epsilon, -\epsilon, -\epsilon)$, $\delta_1 = (\epsilon, 0, 0)$, $\delta_2 = (0, \epsilon, 0)$, or $\delta_3 = (0, 0, \epsilon)$. Here $\epsilon$ is strictly smaller than $1/(2^n-1)$, and we assume that displacements are defined on boundary points so that $f$ maps all points back to the unit cube. (We can canonically interpret circuit outputs to enforce this condition.) We can interpret $f$ as a function defined on the entire unit cube by linearly extending its explicitly defined values within each "cubelet". Then, a cubelet contains a fixed point of $f$ if and only if all four possible displacements are represented amongst its eight corners. The Cubical Brouwer problem is, given a description of a circuit that implicitly defines such a function, compute the coordinates of a fixed-point-containing cubelet; one exists, by Brouwer's fixed-point theorem.

Daskalakis, Goldberg, and Papadimitriou [32] modified an earlier construction of Papadimitriou [99, Theorem 14] to reduce the End-of-the-Line problem to the Cubical Brouwer problem. While technical, this reduction is conceptually quite direct. It starts from an arbitrary End-of-the-Line instance: three Boolean circuits and a corresponding instance, which induce a directed graph $G$. It constructs, in polynomial time, an instance of Cubical Brouwer (a new Boolean circuit $W$) that has a "Sperner graph" — very roughly, with edges connecting neighboring cubelets

[31] Formally, a Boolean circuit is a directed acyclic graph in which every node is either an "input" with no incoming arc; a NOT node with one incoming arc; or an AND or OR node with two incoming arcs. Interpreting 0 and 1 as "false" and "true", respectively, every assignment of the inputs to $\{0, 1\}$ extends uniquely to the rest of the nodes via their respective logical operations. The induced labeling of the terminal nodes — nodes without outgoing arcs — can be regarded as the "output" of the circuit.

[32] The End-of-the-Line problem can be simplified further by partially instantiating the circuits $X, Y, Z$ on the supplied problem instance, effectively "hard-wiring" it into the circuits. An End-of-the-Line instance can then be specified solely via these three circuits [33].

at which the displacements $\delta_1, \delta_2, \delta_3$ are all present — that is isomorphic to $G$.[33] This construction effectively "embeds" $G$ into the unit cube in a non-crossing way — hence the need for three dimensions — and appropriately surrounds the edges of $G$ with the displacements $\delta_1, \delta_2, \delta_3$ to ensure that all four displacements are simultaneously present only at the cubelets where the paths of $G$ end (excluding the standard source of $G$, which is mapped to the corner of the unit cube at which $\delta_0$ is absent). In particular, this construction puts the solutions to the End-of-the-Line instance in bijective correspondence with those of the constructed Cubical Brouwer instance, and one of the former can be recovered from one of the latter in polynomial time.

The last reduction is from the Cubical Brouwer problem to the problem of computing a Nash equilibrium of a bimatrix game — in a sense, reversing the reduction in the standard proof of Nash's Theorem [94] — and it is by far the most elaborate one in the sequence. Papadimitriou [101, §2.6] gives an accessible high-level overview of this reduction, both in its original form [33] and with the subsequent refinements [22], so we furnish only a few sentences here. Given an instance of Cubical Brouwer in the form of a Boolean circuit $W$, the first step is to encode the fixed points of the corresponding function as the Nash equilibria of a finite game with a very large number of players. A Nash equilibrium of the constructed game must, in particular, simulate the computation performed by $W$. This simulation is done node-by-node. For example, a node of $W$ that performs a logical AND operation — or, more generally, multiplication — on its two inputs can represented as a game with four players, each of which has two strategies.[34] Other operations can be similarly simulated, and the games corresponding to all of the nodes of $W$ can be glued together into a single game (with many players) that implicitly performs, at a Nash equilibrium, the entire computation of $W$. Augmenting this construction to discard the Nash equilibria that do not correspond to fixed points of the function defined by $W$ requires several additional ideas [33]. The payoff of each player in the resulting game turns out to depend on the strategies of only a few others, so the size of the game remains polynomial in the size of $W$. The final step in the proof of Theorem 4.2 is to reduce the number of players in this game from a polynomial number to two. The following naive idea, implemented with enough care [22, 33], turns out to work. The players of the game are partitioned into two appropriate sets, and each set is replaced by a "super-player" that simultaneously plays the roles of all players in the set. By adding matching pennies-type payoffs, super-players can be forced to adopt the role of each player in its set with positive probability in a Nash equilibrium. Moreover, because of the special structure of the game, there are no problems with inadvertent collusion among players that are grouped in the same set — the best responses of a super-player correspond to uncoordinated best responses by each of its constituent original players. A Nash equilibrium of the induced two-player game can thus be naturally and efficiently mapped to one of the game with many players, which in turn can be efficiently mapped to a fixed-point-containing cubelet of the function defined by $W$.

[33] This is the same type of graph in which path-following proves Sperner's Lemma [26].

[34] Consider players $a, b, c, d$, each with strategies 0 and 1. Ignore the payoffs of players $a$ and $b$. If player $c$ plays strategy 0 then it receives payoff 1 if both $a$ and $b$ play 1, and payoff 0 otherwise. If player $c$ plays strategy 1, then its payoff is independent of the strategies of players $a$ and $b$. Instead, it effectively plays a coordination game with player $d$, receiving payoff 0 (1) if player $d$ plays strategy 0 (1). Player $d$ effectively plays a game of matching pennies with player $c$, receiving payoff 0 (1) if its strategy is the same as (is different than) that of player $c$. At a Nash equilibrium in which players $a$ and $b$ play strategy 1 with probabilities $p_a$ and $p_b$, respectively, player $d$ must play its strategy 1 with probability precisely $p_a \cdot p_b$.

### 4.5 Discussion

**Behavioral Implications.** Theorem 4.2 implies that there is no polynomial-time algorithm for computing Nash equilibria in bimatrix games, unless all problems in $PPAD$ are solvable in polynomial time. This raises the strong possibility (equivalent to $PPAD \not\subseteq P$) that there is no computationally efficient, general-purpose algorithm for this problem. In turn, this casts doubt on any interpretation of the Nash equilibrium concept that requires its computation — for example, if $PPAD \not\subseteq P$ then there is no general and tractable procedure for learning Nash equilibria in a reasonable amount of time, even with only two players and publicly known payoff matrices. Of course, all of the caveats from Sections 2.6 and 3.5 also apply here.

**Nash vs. Correlated Equilibrium.** Does the fact the computing a Nash equilibrium is $PPAD$-hard indicate that we should discard the concept? Of course not. But it does suggest that more tractable solutions concepts are likely to be more appropriate in some contexts. Indeed, complexity results can argue both for and against the predictive power of an equilibrium concept. Gilboa and Zemel [51] were the first to cite computational complexity as a possible reason to prefer the correlated equilibrium [6] over the Nash equilibrium as a solution concept. They studied a number of computational problems about equilibria in bimatrix games (Problem 2.8 and others) and showed that all of them are $NP$-hard for Nash equilibria but polynomial-time solvable for correlated equilibria.

Subsequent results reinforced the dichotomy between Nash and correlated equilibria from a learning perspective. Foster and Vohra [44] and Hart and Mas-Colell [55, 57] showed senses in which correlated equilibria are "quickly learnable", even via "uncoupled dynamics" in which players are informed only of their own payoffs, and not of the game being played. In contrast, Hart and Mansour [56] proved that every type of uncoupled dynamics, no matter how complex, requires time exponential in the number of players to reach a Nash equilibrium. Theorem 4.2 echoes these developments from a computational complexity viewpoint. Finally, Papadimitriou and Roughgarden [102] showed that computing correlated equilibria remains computationally tractable even for most "succinctly described games", including the congestion games of Section 3.1.

**Approximate vs. Exact Nash Equilibrium.** The approximate Nash equilibrium appears to be another more computationally tractable solution concept than the exact Nash equilibrium, although the picture is currently much murkier than for correlated equilibria. Many notions of approximate Nash equilibria are possible. To fix the discussion, consider bimatrix games with payoffs that are rational numbers in $[0, 1]$, and define an *$\epsilon$-Nash equilibrium* as a pair of mixed strategies such that neither player can increase their expected payoff more than $\epsilon$ by switching strategies.

Computing an $\epsilon$-Nash equilibrium of a bimatrix game appears to be a strictly easier problem than computing an exact Nash equilibrium, at least when $\epsilon$ is a constant independent of the size of the game. For example, since every bimatrix game in which each player has at most $n$ strategies possesses an $\epsilon$-Nash equilibrium in which both players mix over only $\approx (\ln n)/\epsilon^2$ strategies [82], restricting the support enumeration algorithm of Section 4.1 to supports bounded by this size computes an $\epsilon$-Nash equilibrium in $n^{O(\epsilon^{-2} \log n)}$ time, which is sub-exponential (but super-polynomial) in the size of the game. No equally fast algorithm is known for computing an exact Nash equilibrium. There are also polynomial-time algorithms for computing an $\epsilon$-Nash equilibrium with relatively large $\epsilon$ (around 1/3) [15, 129]. On the negative side, Theorem 4.2 can be strengthened to rule out polynomial-time algorithms that compute an $\epsilon$-Nash equilibrium with $\epsilon$ going to zero inverse

polynomially (or faster) with the size of the game's description (assuming $PPAD \not\subseteq P$) [22]. A major open research question is whether or not there is an algorithm that, for arbitrarily small $\epsilon > 0$, computes an $\epsilon$-Nash equilibrium in time polynomial in the game's description (and possibly exponential in $1/\epsilon$). A related open question is whether or not $\epsilon$-Nash equilibria are "easy to learn", for example by uncoupled dynamics [56].

**Tractable and Intractable Special Cases.** Problem 4.1 can be solved efficiently in special cases such as constant-sum games (see e.g. Dantzig [29]) and certain generalizations [69, 82], and also in some models of random games [7]. On the other hand, Theorem 4.2 can be extended to games that have only 0-1 payoffs [2], including those with only a small number of non-zeros in each row and column [21]. That said, the games constructed via these reductions have an elaborate structure that seems unlikely to occur in "real" games. We look forward to a long-running mathematical debate over whether or not games encountered "in practice" are hard in a complexity-theoretic sense.

**What if $PPAD \subseteq P$?** An interesting and important research question is whether or not $PPAD \subseteq P$. There is reason to believe that $PPAD \not\subseteq P$, or at least that a proof of $PPAD \subseteq P$ is nowhere in sight: as with $PLS \subseteq P$ (Section 3.5), a proof that $PPAD \subseteq P$ would have to be "non-relativizing" [8, 43] and would yield a generic method of "shortcutting" the (exponential-time) guided search procedure in every $PPAD$ problem. A concrete negative result was given by Hirsch, Papadimitriou, and Vavasis [61] for computing approximate Brouwer fixed points, which applies even to the special case of the Cubical Brouwer problem (Section 4.4): every algorithm that computes an approximate fixed point and merely queries the supplied Brouwer function as a "black box", as opposed to examining the details of its description, requires exponential time in the worst case. However, such lower bounds do not guarantee that a problem like Cubical Brouwer cannot be solved in polynomial time. For example, the Simplex algorithm requires exponential time in the worst case [74], but other algorithmic methods show that linear programming is a polynomial-time solvable problem [70, 73].

# 5 Conclusions

Computation implicitly underlies many central concepts in economics and game theory — from a player determining a best response, to a market discovering equilibrium prices, to a designer striving for an efficient allocation of scarce resources. It has been widely noted that these concepts reflect reality only if the amount of computation required is "reasonable" in some sense. As we have seen, computational complexity theory offers a flexible paradigm for rigorously differentiating between computational problems that are "easy" — those requiring polynomial-time (or "reasonable") computation — and those that are "intrinsically hard". Originally inspired by engineering applications, this paradigm is increasingly used to inform basic research across the natural and social sciences.

For equilibrium computation problems, polynomial-time solvability correlates well with efficient learnability and has an appealing interpretation in terms of boundedly rational participants. While the exact definition of "hard" varies with the nature of the problem (cf., Theorems 2.9, 3.4, and 4.2), all such hardness results suggest that no fast and general algorithm will be discovered, motivating the study of heuristics, distributional analyses, domain-specific special cases, and alternative, more

tractable equilibrium concepts. We hope that the fundamental nature of these insights inspires the reader to supplement their analytic toolbox with complexity-theoretic techniques; there are surely ample unexplored opportunities for their application.

## Acknowledgments

Many colleagues offered good, and in some cases brilliant, suggestions that influenced this survey. I am grateful to all of them: Costis Daskalakis, Guilherme de Freitas, Matt Jackson, Ramesh Johari, Ehud Kalai, Ilan Kremer, John Ledyard, Yishay Mansour, Andy McLennan, Michael Ostrovsky, Christos Papadimitriou, Bill Sandholm, Rahul Savani, Alexander Skopalik, Adam Szeidl, Adrian Vetta, Ricky Vohra, Bernhard von Stengel, Bob Wilson, and an anonymous referee.

## References

[1] S. Aaronson. Is $P$ versus $NP$ formally independent? *Bulletin of the European Association for Theoretical Computer Science*, 81:109–136, 2003.

[2] T. G. Abbott, D. Kane, and P. Valiant. On the complexity of two-player win-lose games. In *Proceedings of the 46th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 113–122, 2005.

[3] H. Ackermann, H. Röglin, and B. Vöcking. On the impact of combinatorial structure on congestion games. *Journal of the ACM*, 55(6), 2008. Article 25.

[4] E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. B. Miltersen. On the complexity of numerical analysis. *SIAM Journal on Computing*, 2009. To appear.

[5] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton, 2007.

[6] R. J. Aumann. Subjectivity and correlation in randomized strategies. *Journal of Mathematical Economics*, 1(1):67–96, 1974.

[7] I. Bárány, S. Vempala, and A. Vetta. Nash equilibria in random games. *Random Structures and Algorithms*, 31(4):391–405, 2007.

[8] P. Beame, S. Cook, J. Edmonds, R. Impagliazzo, and T. Pitassi. The relative complexity of NP search problems. *Journal of Computer and System Sciences*, 57(1):3–19, 1998.

[9] J. R. S. Blair, D. Mutchler, and M. van Lent. Perfect recall and pruning in games with imperfect information. *Computational Intelligence*, 12:131–154, 1996.

[10] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer-Verlag, 1998.

[11] L. Blumrosen and N. Nisan. Combinatorial auctions. In N. Nisan, T. Roughgarden, É. Tardos, and V. V. Vazirani, editors, *Algorithmic Game Theory*, chapter 11, pages 267–299. Cambridge, 2007.

[12] A. Bogdanov and L. Trevisan. Average-case complexity. *Foundations and Trends in Theoretical Computer Science*, 2(1):1–106, 2006.

[13] J. L. Bona and M. S. Santos. On the role of computation in economic theory. *Journal of Economic Theory*, 72(2):241–281, 1997.

[14] C. Borgs, J. T. Chayes, N. Immorlica, A. T. Kalai, V. S. Mirrokni, and C. H. Papadimitriou. The myth of the folk theorem. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 365–372, 2008.

[15] H. Bosse, J. Byrka, and E. Markakis. New algorithms for approximate Nash equilibria in bimatrix games. In *Proceedings of the Third Annual International Workshop on Internet and Network Economics (WINE)*, volume 4858 of *Lecture Notes in Computer Science*, pages 17–29, 2007.

[16] M. Braverman and S. Cook. Computing over the reals: Foundations for scientific computing. *Notices of the American Mathematical Society*, 53(3):318–329, 2006.

[17] J. W. Brown and J. von Neumann. Solutions of games by differential equations. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory Games*, volume 1, pages 73–79. Princeton, 1950.

[18] X. Chen and X. Deng. 3-Nash is PPAD-complete. Technical Report TR05-134, ECCC, 2005.

[19] X. Chen and X. Deng. Settling the complexity of two-player Nash equilibrium. In *Proceedings of the 47th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 261–270, 2006.

[20] X. Chen, X. Deng, and S.-H. Teng. Computing Nash equilibria: Approximation and smoothed complexity. In *Proceedings of the 47th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 603–612, 2006.

[21] X. Chen, X. Deng, and S.-H. Teng. Sparse games are hard. In *Proceedings of the Second Annual International Workshop on Internet and Network Economics (WINE)*, volume 4286 of *Lecture Notes in Computer Science*, pages 262–273, 2006.

[22] X. Chen, X. Deng, and S.-H. Teng. Settling the complexity of two-player Nash equilibria. *Journal of the ACM*, 2009. To appear. Journal version of [18], [19], [20], and [21].

[23] S. Chien and A. Sinclair. Convergence to approximate Nash equilibria in congestion games. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 169–178, 2007.

[24] F. Chu and J. Halpern. On the NP-completeness of finding an optimal strategy in games with common payoffs. *International Journal of Game Theory*, 30(1):99–106, 2001.

[25] B. Codenotti and K. Varadarajan. Computation of market equilibria by convex programming. In N. Nisan, T. Roughgarden, É. Tardos, and V. V. Vazirani, editors, *Algorithmic Game Theory*, chapter 6, pages 135–158. Cambridge, 2007.

[26] D. I. A. Cohen. On the Sperner lemma. *Journal of Combinatorial Theory*, 2(4):585–587, 1967.

[27] V. Conitzer and T. Sandholm. Complexity results about Nash equilibria. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 765–771, 2003.

[28] S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing (STOC)*, pages 151–158, 1971.

[29] G. B. Dantzig. *Linear Programming and Extensions*. Princeton, 1963.

[30] S. Dasgupta, C. H. Papadimitriou, and U. Vazirani. *Algorithms*. McGraw-Hill, 2008.

[31] C. Daskalakis, A. Fabrikant, and C. H. Papadimitriou. The game world is flat: The complexity of Nash equilibria in succinct games. In *Proceedings of the 33rd Annual International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 4051 of *Lecture Notes in Computer Science*, pages 513–524, 2006.

[32] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou. The complexity of computing a Nash equilibrium. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 71–78, 2006.

[33] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 2009. To appear. Journal version of [32], [34], and [52].

[34] C. Daskalakis and C. H. Papadimitriou. Three-player games are hard. Technical Report TR05-139, ECCC, 2005.

[35] X. Deng and C. H. Papadimitriou. On the complexity of cooperative game solution concepts. *Mathematics of Operations Research*, 19(2):257–266, 1994.

[36] U. Doraszelski and A. Pakes. A framework for applied dynamic analysis in IO. In M. Armstrong and R. Porter, editors, *Handbook of Industrial Organization*, volume 3, chapter 30, pages 1887–1966. North-Holland, 2007.

[37] J. Dunkel and A. S. Schulz. On the complexity of pure-strategy Nash equilibria in congestion and local-effect games. *Mathematics of Operations Research*, 33(4):851–868, 2008.

[38] B. C. Eaves and K. Schmedders. General equilibrium models and homotopy methods. *Journal of Economic Dynamics and Control*, 23(9–10):1249–1279, 1999.

[39] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17(3):449–467, 1965.

[40] K. Etessami and A. Lochbihler. The computational complexity of evolutionarily stable strategies. *International Journal of Game Theory*, 37(1):93–113, 2008.

[41] K. Etessami and M. Yannakakis. On the complexity of Nash equilibria and other fixed points. In *Proceedings of the 48th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 113–123, 2007.

[42] A. Fabrikant, C. H. Papadimitriou, and K. Talwar. The complexity of pure Nash equilibria. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 604–612, 2004.

[43] L. Fortnow. The role of relativization in complexity theory. *Bulletin of the European Association for Theoretical Computer Science*, 52:229–244, 1994.

[44] D. Foster and R. Vohra. Calibrated learning and correlated equilibrium. *Games and Economic Behavior*, 21(1-2):40–55, 1997.

[45] D. Gale, H. W. Kuhn, and A. W. Tucker. On symmetric games. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory Games*, volume 1, pages 81–87. Princeton, 1950.

[46] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69(1):9–15, 1962.

[47] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[48] W. Gasarch. The $P =?NP$ poll. *SIGACT News*, 33(2):34–47, 2002.

[49] I. Gilboa. The complexity of computing best-response automata in repeated games. *Journal of Economic Theory*, 45(2):342–352, 1988.

[50] I. Gilboa, E. Kalai, and E. Zemel. The complexity of eliminating dominated strategies. *Mathematics of Operations Research*, 18(3):553–565, 1993.

[51] I. Gilboa and E. Zemel. Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*, 1(1):80–93, 1989.

[52] P. W. Goldberg and C. H. Papadimitriou. Reducibility among equilibrium problems. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 61–70, 2006.

[53] O. Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge, 2008.

[54] K. A. Hansen, P. B. Miltersen, and T. B. Sørensen. Finding equilibria in games of no chance. In *Proceedings of the 13th Annual Conference on Computing and Combinatorics (COCOON)*, volume 4598 of *Lecture Notes in Computer Science*, pages 274–284, 2007.

[55] S. Hart. Adaptive heuristics. *Econometrica*, 73(5):1401–1430, 2005.

[56] S. Hart and Y. Mansour. The communication complexity of uncoupled Nash equilibrium procedures. *Games and Economic Behavior*, 2009. To appear.

[57] S. Hart and A. Mas-Colell. A simple adaptive procedure leading to correlated equilibria. *Econometrica*, 68(5):1127–1150, 2000.

[58] J. Hartmanis, P. M. Lewis II, and R. E. Stearns. Hierarchies of memory limited computations. In *Proceedings of the 6th Annual IEEE Symposium on Switching Circuit Theory and Logical Design*, pages 179–190, 1965.

[59] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[60] E. Hazan and R. Krauthgamer. How hard is it to approximate the best Nash equilibrium? In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 720–727, 2009.

[61] M. D. Hirsch, C. H. Papadimitriou, and S. A. Vavasis. Exponential lower bounds for finding Brouwer fix points. *Journal of Complexity*, 5(4):379–416, 1989.

[62] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2001. Second edition.

[63] D. S. Johnson. The NP-completeness column: Finding needles in haystacks. *ACM Transactions on Algorithms*, 3(2), 2007. Article 24.

[64] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, 1988.

[65] K. L. Judd. Computational economics and economic theory: Substitutes or complements? *Journal of Economic Dynamics and Control*, 21(6):907–942, 1997.

[66] E. Kalai. Bounded rationality and strategic complexity in repeated games. In T. Ichiishi, A. Neyman, and Y. Tauman, editors, *Game Theory and Applications*, pages 131–157. Academic Press, 1990.

[67] E. Kalai. Games, computers, and O.R. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 468–473, 1996.

[68] E. Kalai and E. Zemel. Totally balanced games and games of flow. *Mathematics of Operations Research*, 7(3):476–478, 1982.

[69] R. Kannan and T. Theobald. Games of fixed rank: A hierarchy of bimatrix games. *Economic Theory*, 2009. This volume.

[70] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[71] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[72] M. Kearns. Graphical games. In N. Nisan, T. Roughgarden, É. Tardos, and V. V. Vazirani, editors, *Algorithmic Game Theory*, chapter 7, pages 159–180. Cambridge, 2007.

[73] L. G. Khachiyan. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20(1):191–194, 1979.

[74] V. Klee and G. J. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalities, III*, pages 159–175. Academic Press, 1972.

[75] J. Kleinberg and É. Tardos. *Algorithm Design*. Addison-Wesley, 2006.

[76] D. E. Knuth. A terminological proposal. *SIGACT News*, 6(1):12–18, 1974.

[77] D. Koller and N. Megiddo. The complexity of two-person zero-sum games in extensive form. *Games and Economic Behavior*, 4(4):528–552, 1992.

[78] D. Koller and A. Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94(1-2):167–215, 1997.

[79] D. Lehmann, R. Müller, and T. Sandholm. The winner determination problem. In P. Cramton, Y. Shoham, and R. Steinberg, editors, *Combinatorial Auctions*, chapter 12, pages 297–317. MIT Press, 2006.

[80] C. E. Lemke and J. T. Howson, Jr. Equilibrium points of bimatrix games. *SIAM Journal*, 12(2):413–423, 1964.

[81] L. Levin. Universal search problems (in Russian). *Problemy Peredachi Informatsii*, 9(3):115–116, 1973.

[82] 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.

[83] M. Littman and P. Stone. A polynomial-time Nash equilibrium algorithm for repeated games. *Decision Support Systems*, 39(1):55–66, 2005.

[84] R. D. McKelvey and A. McLennan. Computation of equilibria in finite games. In H. M. Amman, D. A. Kendrick, and J. Rust, editors, *Handbook of Computational Economics*, volume 1, pages 87–142. North-Holland, 1996.

[85] A. McLennan and R. Tourky. Simple complexity from imitation games. *Games and Economic Behavior*, 2009. To appear.

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

[87] N. Megiddo and A. Wigderson. On play by means of computing machines. In *Proceedings of the 1st Conference on Theoretical Aspects of Reasoning about Knowledge (TARK)*, pages 259–274, 1986.

[88] I. Milchtaich. Congestion models of competition. *American Naturalist*, 147(5):760–783, 1996.

[89] P. Milgrom. Assignment messages and exchanges. Working paper, December 2008.

[90] P. B. Miltersen and T. B. Sørensen. Computing a quasi-perfect equilibrium of a two-player game. *Economic Theory*, 2009. This volume.

[91] D. Monderer and L. S. Shapley. Potential games. *Games and Economic Behavior*, 14(1):124–143, 1996.

[92] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, 1965.

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

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

[95] A. Nerode. Linear automaton transformations. In *Proceedings of the American Mathematical Society*, volume 9, pages 541–544, 1958.

[96] A. Neyman. Finitely repeated games with finite automata. *Mathematics of Operations Research*, 23(3):513–552, 1998.

[97] E. Nudelman, J. Wortman, Y. Shoham, and K. Leyton-Brown. Run the GAMUT: A comprehensive approach to evaluating game-theoretic algorithms. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 880–887, 2004.

[98] C. H. Papadimitriou. On players with a bounded numbers of states. *Games and Economic Behavior*, 4(1):122–131, 1992.

[99] 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.

[100] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1995.

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

[102] C. H. Papadimitriou and T. Roughgarden. Computing correlated equilibria in multi-player games. *Journal of the ACM*, 55(3), 2008. Article 14.

[103] C. H. Papadimitriou and M. Yannakakis. On complexity as bounded rationality. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 726–733, 1994.

[104] R. Porter, E. Nudelman, and Y. Shoham. Simple search strategies for finding a Nash equilibrium. *Games and Economic Behavior*, 63(2):642–662, 2008.

[105] T. Quint and M. Shubik. A model of migration. Working paper #1088, Cowles Foundation, Yale University, 1994.

[106] M. O. Rabin. Effective computability of winning strategies. In M. Dresher, A. W. Tucker, and P. Wolfe, editors, *Contributions to the Theory Games*, volume 3, pages 147–157. Princeton, 1957.

[107] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):115–125, 1959.

[108] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[109] R. W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2(1):65–67, 1973.

[110] A. E. Roth. The evolution of the labor market for medical interns and residents: A case study in game theory. *Journal of Political Economy*, 92(6):991–1016, 1984.

[111] T. Roughgarden. *Selfish Routing and the Price of Anarchy*. MIT Press, 2005.

[112] A. Rubinstein. *Modeling Bounded Rationality*. MIT Press, 1998.

[113] W. H. Sandholm. Potential games with continuous player sets. *Journal of Economic Theory*, 97(1):81–108, 2001.

[114] R. Savani and B. von Stengel. Hard-to-solve bimatrix games. *Econometrica*, 74(2):397–429, 2006.

[115] H. E. Scarf. The approximation of fixed points of a continuous mapping. *SIAM Journal of Applied Mathematics*, 15(5):1328–1343, 1967.

[116] H. E. Scarf (with the collaboration of T. Hansen). *The Computation of Economic Equilibria*. Yale, 1973.

[117] A. A. Schäffer and M. Yannakakis. Simple local search problems that are hard to solve. *SIAM Journal on Computing*, 20(1):56–87, 1991.

[118] G. Schoenebeck and S. Vadhan. The computational complexity of Nash equilibria in concisely represented games. In *Proceedings of the 7th ACM Conference on Electronic Commerce (EC)*, pages 270–279, 2006.

[119] I. Segal. The communication requirements of combinatorial allocation problems. In P. Cramton, Y. Shoham, and R. Steinberg, editors, *Combinatorial Auctions*, chapter 11, pages 265–294. MIT Press, 2006.

[120] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic and Logical Foundations*. Cambridge, 2009.

[121] H. A. Simon. *Models of Bounded Rationality*. MIT Press, 1984.

[122] M. Sipser. The history and status of the P versus NP question. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC)*, pages 603–618, 1992.

[123] M. Sipser. *Introduction to the Theory of Computation*. Course Technology, 2005. Second edition.

[124] A. Skopalik and B. Vöcking. Inapproximability of pure Nash equilibria. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 355–364, 2008.

[125] A. Smith. *An Inquiry Into the Nature and Causes of the Wealth of Nations*. Methuen, 1776.

[126] J. M. Stalnaker. The matching program for intern placement: The second year of operation. *Journal of Medical Education*, 28(11):13–19, 1953.

[127] É. Tardos and T. Wexler. Network formation games and the potential function method. In N. Nisan, T. Roughgarden, É. Tardos, and V. V. Vazirani, editors, *Algorithmic Game Theory*, chapter 19, pages 487–516. Cambridge, 2007.

[128] M. J. Todd. Orientation in complementary pivot algorithms. *Mathematics of Operations Research*, 1(1):54–66, 1976.

[129] H. Tsaknakis and P. G. Spirakis. An optimization approach for approximate Nash equilibria. In *Proceedings of the Third Annual International Workshop on Internet and Network Economics (WINE)*, volume 4858 of *Lecture Notes in Computer Science*, pages 42–56, 2007.

[130] A. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, Series 2*, 42:230–265, 1936. Erratum: 43:544–546, 1937.

[131] P. van Emde Boas. Machine models and simulations. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 1, pages 1–66. MIT Press, 1990.

[132] V. V. Vazirani. Combinatorial algorithms for market equilibria. In N. Nisan, T. Roughgarden, É. Tardos, and V. V. Vazirani, editors, *Algorithmic Game Theory*, chapter 5, pages 103–134. Cambridge, 2007.

[133] B. Vöcking. Congestion games: Optimization in competition. In *Proceedings of the Second Workshop on Algorithms and Complexity in Durham (ACiD)*, pages 9–20, 2006.

[134] R. Vohra. *Advanced Mathematical Economics*. Routledge, 2005.

[135] B. von Stengel. Computing equilibria for two-person games. In R. J. Aumann and S. Hart, editors, *Handbook of Game Theory with Economic Applications*, volume 3, chapter 45, pages 1723–1759. North-Holland, 2002.

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

[137] B. von Stengel and F. Forges. Extensive form correlated equilibrium: Definition and computational complexity. *Mathematics of Operations Research*, 33(4):1002–1022, 2008.

[138] A. Wigderson. Knowledge, creativity, and $P$ versus $NP$. Manuscript, 2006.

[139] M. Yannakakis. Computational complexity. In E. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, chapter 2, pages 19–55. Wiley, 1997. Reprinted by Princeton University Press, 2003.

[140] M. Yannakakis. Equilibria, fixed points, and complexity classes. In *Proceedings of the 25th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 19–38, 2008.