# Cryptography and Complexity Theory

## 1 Games

We can encode the game rock-paper-scissors as a "payoff" matrix:

| | rock | paper | scissors |
|---|---|---|---|
| rock | $(0,0)$ | $(-1,1)$ | $(1,-1)$ |
| paper | $(1,-1)$ | $(0,0)$ | $(-1,1)$ |
| scissors | $(-1,1)$ | $(1,-1)$ | $(0,0)$ |

The entries of the matrix describe the "point value" of a pair of actions to each player.

More generally, we can make the following definition, where $P$ denotes the number of players in the game, and $[A] = \{1, \dots, A\}$ index the actions available to each player (for simplicity, we assume all players have the same number of actions available to them).

**Definition 1** (Game)**.** *Let $P, A \in \mathbb{N}$. A game*[1] *consists of a function $\mathsf{Payoff}_p : [A]^P \to \mathbb{R}$ for each player $p \in [P]$.*

In the field of Game Theory, one seeks to understand what strategies "self-interested" and "rational" actors ought to pursue in a given game. This theory has had enormous influence, especially in economics but also in other areas. Perhaps its central theorem is due to Nash.

**Theorem 2** (Informal [Nas50])**.** *Every game has an "optimal" strategy.*

If you have never seen this before, it is worth pausing to consider this. The definition of a game is incredibly general. What might "an optimal" strategy mean? A priori, it seems difficult to give a definition that is both meaningful and that exists for every game.

Let's investigate the game rock-paper-scissors. It is easy to see no single action is best. However, there is an intuitively optimal strategy for rock-paper-scissors: select a uniformly random element of {rock, paper, scissor}. This motivates the following definition.

**Definition 3** (Mixed Strategy)**.** *A mixed strategy $S$ is a distribution on $[A]$. For mixed strategies $S_1, \dots, S_P$, let*

$$\mathsf{Payoff}_p(S_1, \dots, S_P) = \mathop{\mathbb{E}}_{a_1 \leftarrow S_1, \dots, a_P \leftarrow S_P} [\mathsf{Payoff}_p(a_1, \dots, a_P)].$$

Can we get some mathematical handle on what mixed strategies self-interested rational actors will use in an arbitrary game? Nash had the following brilliant idea.

**Definition 4** (Nash Equilibrium)**.** *Strategies $S_1, S_2, \dots, S_P$ are a Nash equilibrium if for every player $p \in [P]$ and every mixed strategy $S'$, we have*

$$\mathsf{Payoff}_p(S_1, \dots, S_P) \geq \mathsf{Payoff}_p(S_1, \dots, S_{i-1}, S', S_{i+1}, \dots S_P).$$

Intuitively, this means that no individual player has an incentive to change their strategy, which is why it is called an equilibrium. In rock-paper-scissors (indeed, any two-player zero-sum game), there is a unique Nash equilibrium, which also corresponds to (for example) the "maxi-min" strategy (where one maximizes your worst-case success probability over any enemy mixed strategy).

We can now state Nash's theorem formally.

[1] Technically, this is refered to as a normal form game.

**Theorem 5** ([Nas50]). *Every game has a Nash equilibrium.*

We note that Nash equilibria are not necessarily unique, and are not necessarily the "best" strategy. Consider,

| | stag | hare |
|---|---|---|
| stag | $(3,3)$ | $(0,1)$ |
| hare | $(1,0)$ | $(1,1)$ |

Stag-stag and hare-hare are both Nash equilibria, but stag-stag is clearly the best strategy.

We will sketch the proof of Nash's theorem, assuming Brouwer's fixed point theorem.

**Theorem 6** (Brouwer's Fixed Point Theorem). *Let $f : [0,1]^d \to [0,1]^d$ be a continuous function. Then there exists an $x \in [0,1]^d$ such that $f(x) = x$.*

*Proof in case where $d = 1$.* Apply the intermediate value theorem to $x - f(x)$. At zero, it is at most 0 and at one it is at least zero, so somewhere in the interval it must be zero. □

In fact, we will need a further generalization (which we will not prove here).

**Theorem 7** (Brouwer's Fixed Point Theorem). *Let $X \subseteq \mathbb{R}^d$ be compact (closed and bounded) and convex. Let $f : X \to X$ be a continuous function. Then there exists an $x \in X$ such that $f(x) = x$.*

Each of the requirements is necessary to some degree:

- Closure: $f(x) = \frac{x}{2} + 1/2$ has no fixed point on $(0,1)$.
- Bounded: $f(x) = x + 1$ on $\mathbb{R}$
- Convex: $f(x) = 2x$ on $[-2,-1] \cup [1,2]$

Now we will sketch the proof of Nash's theorem.

*Proof Sketch.* Let $\Delta$ denote the set of distributions on $[A]$. Note that we can view $\Delta = \{(p_1, \dots, p_A) \in [0,1]^A : 1 = p_1 + \cdots + p_A\}$. It can be shown that $\Delta^P$ is compact and convex.

Now we will construct a function $Improve : \Delta^P \to \Delta^P$ with the following two properties:

- **continuous**: $Improve$ is continuous
- **improving**: Suppose $Improve(S_1, \dots, S_P) = (S'_1, \dots, S'_P)$. Then we have that for all $p \in P$ that if there exists an $S^\star_p$ such that

$$\mathsf{Payoff}_p(S_1, \dots, S_{p-1}, S^\star_p, S_{p+1}, \dots S_P) > \mathsf{Payoff}_p(S_1, \dots, S_P),$$

  then

$$\mathsf{Payoff}_p(S_1, \dots, S_{p-1}, S'_p, S_{p+1}, \dots S_P) > \mathsf{Payoff}_p(S_1, \dots, S_P).$$

We leave the construction of $Improve$ as an exercise (the idea is to continuously nudge up the probabilities of actions that improve the payoff for the $p$'th player).

Then by Brouwer's fixed point theorem, we have that there exists $S_1, \dots, S_P$ such that $f(S_1, \dots, S_P) = (S_1, \dots, S_P)$, which implies that it is a Nash equilibrium (by the improving property). □

## 2 The Hardness of Computing Nash Equilibrium

In economics, it is common to analyze a system by modeling it as a game and assuming players act according to a Nash equilibrium. This seems like quite a reasonable assumption, but what if it is hard to compute Nash Equilibria? Then one might question the assumption, as real-world players may never reach a Nash equilibrium. For decades, people tried to find a polynomial-time algorithm for computing a Nash equilibrium (for example, one can $\epsilon$-approximate two-player Nash equilibrium in time $n^{O(\epsilon^{-2}\log n)}$ [LMM03]).

Celebrated work [DGP09, CDT09] shows that computing even approximate two-player Nash Equilibria is complete for the complexity class PPAD (which we will not define here). This class contains, for example, the Sink-of-Line Problem.

**Definition 8** (Sink-Of-Line Problem)**.** *Sink-Of-Line is the following task:*

- ***Given:*** $x_s \in \{0,1\}^n$ *and* $T \in [2^n]$ *and*
  - *a "successor" circuit* $S : \{0,1\}^n \to \{0,1\}^n$ *and*
  - *a "verifier" circuit* $V : \{0,1\}^n \times [2^n]$ *satisfying* $V(x,i) = \mathbb{1}[x = S^{i-1}(x_s))]$
- ***Output:*** $w \in \{0,1\}^n$ *such that* $V(w,T) = 1$.

Using cryptography, we can show that the sink-of-line problem is hard and hence so is computing a Nash equilibrium.

**Theorem 9** ([BPR15])**.** *Assume subexponentially secure iO and subexponentially secure injective PRGs exist. Then Sink-Of-Line is hard (and hence, so is computing approximate Nash Equilibria).*

*Proof.* We set up some notation:

- Let $\lambda$ be a sufficiently large polynomial in $T$.
- Let $PRF_K : \{0,1\}^\lambda \to \{0,1\}^\lambda$ be a subexponentially secure puncturable PRF. $PRF_{K\{x\}}$ denotes the PRF punctured at $x$.
- $G$ be the injective pseudorandom generator that outputs $2\lambda$ bits. We will run $G$ on inputs of length $\log T$, in which case it has security $T^{\Omega(1)}$ or inputs of length $\lambda$ in which case it has security $2^{\lambda^{\Omega(1)}}$.

Now consider the following distributions on circuits.

Sample $K \leftarrow \{0,1\}^\lambda$ and $F = PRF_K$. Output

$$iO\left((i \in [T], \sigma \in \{0,1\}^\lambda) \mapsto \begin{cases} \bot, & \text{if } \sigma \neq F(i) \\ \text{SOLVED}, & \text{if } i = T \\ (i+1, F(i+1)), & \text{otherwise.} \end{cases}\right)$$

We will show that this distribution on circuits is $2^{-T^{\Omega(1)}}$ indistinguishable from a distribution on circuits that never output SOLVED. This implies that Sink-of-Line cannot be solved in time $2^{-T^{\Omega(1)}}$. We now give the hybrid argument. Below $\approx$ means $2^{-\lambda^{\Omega(1)}}$ indistinguishable except the blue $\approx$ which means $T^{-\Omega(1)}$ indistinguishable (we only use this in one hybrid, which is important because we will have $T$ hybrids).

$$iO\left(\begin{cases}\bot, & \text{if } \sigma \neq F(i)\\ \text{SOLVED}, & \text{if } i = T\\ (i+1, F(i+1)), & \text{otherwise.}\end{cases}\right)$$

where $K \leftarrow \{0,1\}^\lambda$ and $F = PRF_K$

$\approx$

$$iO\left(\begin{cases}\bot, & \text{if } G(i) = r\\ \bot, & \text{if } \sigma \neq F(i)\\ \text{SOLVED}, & \text{if } i = T\\ (i+1, F(i+1)), & \text{otherwise.}\end{cases}\right)$$

where $K \leftarrow \{0,1\}^\lambda$, $F = PRF_K$, and $r \leftarrow \{0,1\}^{2\lambda}$

$\approx$

$$iO\left(\begin{cases}\bot, & \text{if } G(i) = r\\ \bot, & \text{if } \sigma \neq F(i)\\ \text{SOLVED}, & \text{if } i = T\\ (i+1, F(i+1)), & \text{otherwise.}\end{cases}\right)$$

where $K \leftarrow \{0,1\}^\lambda$, $F = PRF_K$, and $s \leftarrow [T], r = G(s)$

$\approx$

$$iO\left(\begin{cases}\bot, & \text{if } i = s\\ \bot, & \text{if } \sigma \neq F(i)\\ \text{SOLVED}, & \text{if } i = T\\ (i+1, F(i+1)), & \text{otherwise.}\end{cases}\right)$$

where $K \leftarrow \{0,1\}^\lambda$, $F = PRF_K$, $s \leftarrow [T]$

$\approx$

$$iO\left(\begin{cases}\bot, & \text{if } i = s+1 \text{ and } \sigma \neq \sigma_{s+1}\\ \bot, & \text{if } i = s\\ \bot, & \text{if } i \neq s+1 \text{ and } \sigma \neq F(i)\\ \text{SOLVED}, & \text{if } i = T\\ (i+1, F(i+1)), & \text{otherwise.}\end{cases}\right)$$

where $K \leftarrow \{0,1\}^\lambda$, $s \leftarrow [T]$, $F = PRF_{K\{s+1\}}$, $\sigma_{s+1} = PRF_K(s+1)$

$\approx$

$$iO\left(\begin{cases}\bot, & \text{if } i = s+1 \text{ and } \sigma \neq \sigma_{s+1}\\ \bot, & \text{if } i = s\\ \bot, & \text{if } i \neq s+1 \text{ and } \sigma \neq F(i)\\ \text{SOLVED}, & \text{if } i = T\\ (i+1, F(i+1)), & \text{otherwise.}\end{cases}\right)$$

where $K \leftarrow \{0,1\}^\lambda$, $s \leftarrow [T]$, $F = PRF_{K\{s+1\}}$, $\sigma_{s+1} \leftarrow \{0,1\}^\lambda$

$\approx$

$$iO\left(\begin{cases}\bot, & \text{if } i = s+1 \text{ and } G(\sigma) \neq v_{s+1}\\ \bot, & \text{if } i = s\\ \bot, & \text{if } i \neq s+1 \text{ and } \sigma \neq F(i)\\ \text{SOLVED}, & \text{if } i = T\\ (i+1, F(i+1)), & \text{otherwise.}\end{cases}\right)$$

where $K \leftarrow \{0,1\}^\lambda$, $s \leftarrow [T]$, $F = PRF_{K\{s+1\}}$, $\sigma_{s+1} \leftarrow \{0,1\}^\lambda$, $v_{s+1} = G(\sigma_{s+1})$

$\approx$

$$iO\left(\begin{cases}\bot, & \text{if } i = s+1\\ \bot, & \text{if } i = s\\ \bot, & \text{if } i \neq s+1 \text{ and } \sigma \neq F(i)\\ \text{SOLVED}, & \text{if } i = T\\ (i+1, F(i+1)), & \text{otherwise.}\end{cases}\right)$$

where $K \leftarrow \{0,1\}^\lambda$, $s \leftarrow [T]$, $F = PRF_{K\{s+1\}}$

$\approx$

$$iO\left(\begin{cases}\bot, & \text{if } i \in \{s, s+1\}\\ \bot, & \text{if } i \neq s+1 \text{ and } \sigma \neq F(i)\\ \text{SOLVED}, & \text{if } i = T\\ (i+1, F(i+1)), & \text{otherwise.}\end{cases}\right)$$

where $K \leftarrow \{0,1\}^\lambda$, $s \leftarrow [T]$, $F = PRF_K$

$\approx$

$$iO\left(\begin{cases}\bot, & \text{if } i \neq s+1 \text{ and } \sigma \neq F(i)\\ \bot, & \text{if } i \in \{s, \ldots T\}\\ (i+1, F(i+1)), & \text{otherwise.}\end{cases}\right)$$

where $K \leftarrow \{0,1\}^\lambda$, $s \leftarrow [T]$, $F = PRF_K$

$\square$

## 3 Branching Programs

Branching programs are a model of computation used to study non-uniform space-bounded algorithms.

**Definition 10** (Branching Program). *An $L$-layer $W$-width branching program with $n$-inputs consists of*

- *nodes $v = (\ell, w) \in [L] \times [W]$ (we say $v$ is in layer $\ell$)*
- *a special "start" node in the first layer and special "accept" and "reject" nodes in the last layer*
- *an input $i_v \in [n]$ associated with each node*
- *From each node (except for those in the last layer), two directed edges $e_0$ and $e_1$ which go to nodes in the next layer (if there is one).*

*On input $x$, the output of a branching program is obtained by beginning at the start node, reading the input bit associated with it, following the corresponding edge, and repeating until one reaches either accept or reject.*

For example, there is a simple width-two branching program for the OR function, the AND function, and addition modulo two. Can width-two branching programs compute everything? One can show that they cannot compute majority (no matter how large they are). On the other hand, width-three branching programs of exponential size can compute all functions (consider a DNF).

It is natural to conjecture that width-three branching programs are quite weak.

**Conjecture 11** ([BDFP83]). *Any constant width branching program for Majority requires exponential-size.*

A significant amount of work (including by cryptographers Yao and Ajtai) makes partial progress toward this conjecture. But it turns out this conjecture is false! Our exposition closely follows Viola and Zhou's.[2]

**Theorem 12** ([Bar86]). *Every $n$-input $d$-depth circuit can be computed by a branching program of width 5 and length $4^d$.*

Amazingly, there is some sense in which this theorem is true for the same reason one, famously, cannot solve a quintic equation using "nice algebraic operations."

Recall, the group $S_5 = \{\text{permutations } \pi : [5] \to [5]\}$, where multiplication denotes function composition. A permutation $\pi$ is a *cycle* if $\pi^i(1)$ is distinct for all $i \in [5]$. We write such a $\pi$ in cycle notation as $(1\ \pi(1)\ \ldots\ \pi^4(1))$. Let $e$ denote the identity permutation.

It will be useful to consider a special type of branching program that (in some sense) corresponds to computing over $S_5$.

**Definition 13** ($S_5$-Program). *An $\ell$-length $n$-input $S_5$-program consists of three tuples:*

- *group elements $(g_1^0, \ldots, g_\ell^0) \in S_5^\ell$,*
- *group elements $(g_1^1, \ldots, g_\ell^1) \in S_5^\ell$, and*
- *indices $(k_1, \ldots, k_\ell) \in [n]^\ell$.*

[2] https://www.khoury.northeastern.edu/home/viola/classes/gems-08/lectures/le11.pdf

*The output of the program $P$ on input $x$ is*

$$P(x) = \prod_{i=1}^{\ell} g_i^{x[k_i]}.$$

*We say $P$ $\alpha$-computes $f : \{0,1\}^n \to \{0,1\}$ if*

$$P(x) = \begin{cases} \alpha, & \text{if } f(x) = 0 \\ e, & \text{if } f(x) = 1. \end{cases}$$

Any $S_5$-program can be converted into an equal length width five branching program.

**Proposition 14.** *If an $\ell$-length $S_5$ program $\alpha$-computes $f$, then an $\ell$-length width-5 branching program computes $f$.*

*Proof.* Consider the branching program where the $(j,i)$ goes to $(\pi(j), i+1)$ where $\pi = g_i^{x[k_i]}$. Since $\alpha$ cannot be the identity permutation, there exists some $j^\star$ such that $\alpha(j^\star) \neq j^\star$. Make the start node $(j^\star, 1)$, the accept node $(\alpha(j^\star), \ell)$ and the reject node $(j^\star, \ell)$. □

Now we prove some lemmas about $S_5$ and $S_5$-programs.

**Lemma 15** (Cycles are Conjugate)**.** *Let $\alpha, \beta \in S_5$ be cycles. Then there is a $\rho \in S_5$ such that $\beta = \rho^{-1}\alpha\rho$.*

*Proof.* Write $\alpha = (\alpha_1 \ \dots \ \alpha_5)$ and $\beta = (\beta_1 \ \dots \ \beta_5)$. Then set set $\rho(\beta_i) = \alpha_i$. □

**Lemma 16** (Choice of Cycle is Irrelevant)**.** *Let $\alpha, \beta \in S_5$ be cycles. If there is a length-$\ell$ $S_5$-program that $\alpha$-computes $f$, then there is an $\ell$-length $S_5$-program that $\beta$-computes $f$.*

*Proof.* By the previous lemma, pick $\rho$ such that $\beta = \rho^{-1}\alpha\rho$. If an $S_5$ program with $(g_1^0, \dots, g_\ell^0)$ and $(g_1^1, \dots, g_\ell^1)$ $\alpha$-computes $f$, then the same program but with $(\rho^{-1}g_1^0, \dots, g_\ell^0\rho)$ and $(\rho^{-1}g_1^1, \dots, g_\ell^1\rho)$ $\beta$-computes $f$. This is because $\rho^{-1}e\rho = e$ and $\rho^{-1}\beta\rho = \alpha$ □

**Lemma 17** (Computing Negation)**.** *If an $\ell$-length $S_5$-program $\alpha$-computes $f$, then a program of the same length computes $\neg f$.*

*Proof.* By previous lemma, we can $\alpha^{-1}$-compute $f$ in the same length. Multiplying $g_\ell^0$ and $g_\ell^1$ by $\alpha$ then computes $\neg f$. □

**Lemma 18** (Computing AND)**.** *If $f$ is $\ell$-length $\alpha$-computable and $g$ is $\ell$-length $\beta$-computable, then $f \wedge g$ is $4\ell$-length $\alpha\beta\alpha^{-1}\beta^{-1}$-computable.*

*Proof.* Consider the $4\ell$-length program that concatenates the following programs:

1. $\alpha$-compute $f$
2. $\beta$-compute $g$
3. $\alpha^{-1}$-compute $f$
4. $\beta^{-1}$-compute $g$.

If $f(x) \wedge g(x) = 1$, then the output of this program is $\alpha\beta\alpha^{-1}\beta^{-1}$. But if $f(x) = 0$, then the output is $e\beta e\beta^{-1} = e$, and similarly if $g(x) = 0$. □

**Lemma 19** (Cycles Conjugate to their Commutator)**.** *There are cycles $\alpha$ and $\beta$ such that $\alpha\beta\alpha^{-1}\beta^{-1}$ is a cycle.*

*Proof.* For example, consider $\alpha = (12345)$ and $\beta = (13542)$. □

By induction, we then have the following theorem, which implies Barrington's theorem.

**Theorem 20.** *Let $f$ be a function computable by a $d$-depth circuit. For every cycle $\alpha$, $f$ is $4^d$-length $\alpha$-computable.*

# 4 Perfect Randomized Encodings and Their Applications

**Definition 21** ([AIK04])**.** *A perfect local randomized encoding for a function $f : \{0,1\}^n \to \{0,1\}$ consists of two algorithms $Enc$ and $Dec$ with the following two properties:*

- ***Functionality:*** *$Dec(Enc(x)) = f(x)$ for all $x$*
- ***Locality:*** *$Enc(x)$ is computable by an $\mathsf{NC}^0$ circuit*
- ***Secrecy:*** *There is a simulator $Sim$ such that $Sim(f(x))$ is* identical *to the distribution $Enc(x)$.*
- ***Balanced:*** *$Sim$ run on a uniformly random bit is distributed uniformly at random.*
- ***Output Length****: If $Enc$ takes $q$ bits of randomness, then it outputs at least $q+1$ bits.*

Perhaps surprisingly, perfect local randomized encodings exist *unconditionally.*

**Theorem 22** ([AIK04])**.** *For every $d$-depth circuit $C : \{0,1\}^n \to \{0,1\}$, there is a perfect local randomized encoding where $Enc, Dec$ and $Sim$ run in* $\mathrm{poly}(n, 2^d)$ *time.*

*Proof.* We won't quite prove the theorem (mostly because 5! is not a power of two) but we will prove something close.

We know that we can $\alpha$-compute $C$ with a group program over $S_5$

$$C(x) = \prod_{i=1}^{\ell} g_i^{x[k_i]},$$

where $\ell = 4^d$.

The output of $Enc(x; r)$ will be

$$(g_1^{x[k_1]} r_1^{-1}, r_1 g_2^{x[k_2]} r_2^{-1}, \ldots, r_{\ell-1} g_\ell^{x[k_\ell]})$$

where we sample $r_1, \ldots, r_{\ell-1} \leftarrow S_5$ using the randomness $r$ (note: there is an annoying problem here with sampling these because 5! is not a power of two.). It is easy to see that this is computable in constant depth. If 5! were a power of two, then $Enc$ would take $(\ell-1)\log_2(120)$ bits of randomness and output a string of length $\ell \log_2(120)$.

We then decode by evaluating the group program. Finally, to simulate given output $b$ we just output uniformly random group elements $(g_1, \ldots, g_\ell)$ with the constraint that they decode to $b$. Note this not balanced because the product of the group elements will always either be $e$ or $\alpha$. □

As a corollary, we get that $O(\log n)$-depth PRGs (for which there are many candidates) imply *constant* depth PRGs (with small stretch).

**Corollary 23** ([AIK04]). *Suppose there is a PRG $G : \{0,1\}^n \to \{0,1\}^m$ computable in $d$-depth and $s$-size. Then there is a PRG $G' : \{0,1\}^{n+qm} \to \{0,1\}^{(q+1)m}$ computable with a constant-depth circuit of size* $\mathrm{poly}(s, 4^d)$ *for some $q$.*

*Proof.* Let $Enc$ be a perfect local randomized encoding for $G$ (since $G$ has multiple outputs, we concatenate in the natural way). Let $G'(x) = Enc(x)$. Any distinguisher $D'$ implies a distinguisher $D$ for $G$. In particular, $D(y) = D'(Sim(y))$. In particular, letting $U$ denote uniformly random inputs, we have that

$$D(U) \equiv D'(Sim(U)) \equiv D'(U)$$

and that

$$D(G(U)) \equiv D'(Sim(G(U))) \equiv D'(G'(U)),$$

so $D$ has exactly the same distinguishing advantage that $D'$ has. □

Recall, to construct iO, we needed to assume the existence of local PRGs with large (polynomial) stretch.

## References


[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in $\mathrm{nc}^0$. In *FOCS*, pages 166–175. IEEE Computer Society, 2004.

[Bar86] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in $\mathrm{nc}^1$. In *STOC*, pages 1–5. ACM, 1986.

[BDFP83] Allan Borodin, Danny Dolev, Faith E. Fich, and Wolfgang J. Paul. Bounds for width two branching programs. In *STOC*, pages 87–93. ACM, 1983.

[BPR15] Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a nash equilibrium. In *FOCS*, pages 1480–1498. IEEE Computer Society, 2015.

[CDT09] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player nash equilibria. *J. ACM*, 56(3):14:1–14:57, 2009.

[DGP09] Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a nash equilibrium. *Commun. ACM*, 52(2):89–97, 2009.

[LMM03] Richard J. Lipton, Evangelos Markakis, and Aranyak Mehta. Playing large games using simple strategies. In *EC*, pages 36–41. ACM, 2003.

[Nas50] John F. Nash. Equilibrium points in ¡i¿n¡/i¿-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, 1950.