# On Algorithms for Nash Equilibria

Tim Abbott, Daniel Kane, Paul Valiant

December 16, 2004

**Abstract**

We present a progress report on ongoing research in algorithms for finding sample Nash equilibria of two-player matrix games. We present a combination of background material, new results, and promising directions for further study. Our new results include a reduction from general games to $\{0,1\}$ games, a relation between the complexity of finding Nash equilibria and program obfuscation, and a fixed-parameter tractable algorithm for games with bounded treewidth and degree.

## Contents

## 1 Introduction

We study the problem of finding Nash equilibria in two-player matrix games, a problem that has found increasing applications in both economics and the internet. It has been shown that it is #P-hard to count all Nash equilibria of a two-player game, even if all the matrix entries are 0 or 1 [CS1]. However, the complexity of finding a Nash equilibrium is wide open, and has been proposed as one of the most important open problems in complexity theory today [Pap1].

We give a new polynomial reduction from finding Nash equilibria in general bimatrix games to finding Nash equilibria in games where all payoffs are either 0 or 1, resolving an open problem posed in [CS2]. We exposit the Lemke-Howson algorithm, which is the standard algorithm used in practice to find a single Nash equilibrium, and define the class PPAD that generalizes the problem. We then provide an argument that under standard assumptions, either PPAD is NP-hard or a particular class of random functions cannot be obfuscated. We then describe the state-of-the-art in theoretical algorithms for finding a Nash equilibrium and also the more general notion of an *approximate Nash equilibrium*. Finally, we give a new algorithm that computes an approximate Nash equilibrium which runs in polynomial time in the case where both treewidth and degree are constant.

## 2 Definitions and General Lemmas

**Definition 1 (Game)** *A* bimatrix game *is a two-player game defined by a pair* $(R, C)$ *where* $R$ *and* $C$ *are* $m \times n$ *matrices.* $R$ *and* $C$ *are the* payoff matrices *for the* row *and* column players, *respectively. When the game is played, the row player picks a row* $i$ *to play and the column player picks a column* $j$ *to play, and each player gets a* payoff *equal to the element* $(i, j)$ *of his payoff matrix.*

*The goal of the game is to maximize one's expected payoff.*

**Definition 2** *A* pure strategy *for the row or column player is a row or column index of the payoff matrix, respectively. A* (mixed) strategy *is a probability distribution over the pure strategies, denoted by a vector* $x$. *To be a probability distribution, each entry must be in* $[0, 1]$ *and their sum must be 1. The* support *of a strategy Supp* $(x)$ *is the subset of the pure strategies which the player sometimes plays.*

Note that the (expected) payoff for a player with payoff matrix $M$ if the row player is playing strategy $x$ and the column player is playing strategy $y$ is $x^T My$.

Given such a game, a natural question that arises is what a "rational" player should do. The notion of rationality that has become widely accepted and almost ubiquitous is that of a *best response*. The concept for it is that a player should play a strategy that maximizes his payoff, given what the other player is playing.

**Definition 3** *A strategy* $x$ *is a best response (for the row player) to a strategy* $y$ *if for every strategy* $x'$, $x'^T Ry \leq x^T Ry$.

Given such a game, one might ask what the outcome might reasonably be when both the row and column players are "rational". In this situation, both players will be playing the best response to the other's strategy. This is exactly the notion that the pair of row and column strategies are in a *Nash equilibrium*.

**Definition 4** *A* Nash equilibrium *is a pair of (mixed) strategies* $x^*$ *and* $y^*$ *such that each player's strategy is a best response to the other's, i.e. for any strategies* $x, y$,

$$x^T Ry^* \leq x^{*T} Ry^*, \text{ and } x^{*T} Cy \leq x^{*T} Cy^*$$

**Definition 5** *A* zero-sum game *is a game in which $R + C = 0$, where by 0 we mean the $m \times n$ zero matrix. More generally, a* constant-sum *game is one where $R + C$ is an $m \times n$ matrix all of whose entries are the same. We also will consider the notions of a $\{0, 1\}$* -game*, where all payoffs are either zero or one and a $[0, 1]$-game, where all the payoffs are in the interval $[0, 1]$.*

Note that constant-sum games are equivalent to zero-sum games, since subtracting the constant from all of the entries of one player's payoff matrix $M$ gives a zero-sum game. We apply the following lemma:

**Lemma 6** *The best responses for a player with payoff matrix $M$ are the same as those if we replace the matrix by $z(M - \mathtt{1}r)$, where $\mathtt{1}$ is the $m \times n$ matrix with all entries equal to one, for $r, z \in \mathbb{R}, z > 0$.*

**Proof:** We have that

$$x^T z(M - r\mathtt{1})y = z(x^T My - rx^T \mathtt{1}y) = z(x^T My - r)$$

so that the payoffs are simply shifted by a constant and then scaled by another constant, which by linearity does not affect the notion of a best response. ▌

**Corollary 7** *The problem of finding a Nash equilibrium in a general bimatrix game has a polynomial reduction to finding a Nash equilibrium in a $[0, 1]$-game.*

**Proof:** Apply the previous lemma for each of the matrices, with $r$ equal to the smallest payoff in the matrix and $z$ equal to the difference between the largest and the smallest, unless all the entries are equal in which case we set $z = 1$. ▌

**Lemma 8** *$x$ is a best response to $y$ if and only if for every pure strategy $e_i$,*

$$e_i{}^T Ry \le x^T Ry$$

**Proof:** Suppose every pure strategy $e_i$ satisfied

$$e_i{}^T Ry \le x^T Ry.$$

Then by linearity of matrix multiplication, for an arbitrary $x' = \sum_{i=1}^m x_i^{'} e_i$, we have

$$x'^T Ry = \left(\sum_{i=1}^m x_i^{'} e_i\right)^T Ry = \sum_{i=1}^m x_i^{'} \left(e_i{}^T Ry\right) \le \sum_{i=1}^m x_i^{'} \left(x^T Ry\right) = \left(\sum_{i=1}^m x_i^{'}\right) (x^T Ry) = x^T Ry$$

which implies that $x$ is a best response. The converse is obvious, since any pure strategy is a strategy. ▌

The following corollary simplifies the process of testing whether a pair of strategies is a Nash equilibrium.

**Corollary 9** *$(x^*, y^*)$ is a Nash equilibrium if and only if neither player has a pure strategy that gives that player a higher payoff.*

# 3 Complexity Classes and Reductions

There are several natural questions to ask about the Nash equilibria of a game $G$. Some standard questions are:

1. Find a Nash equilibrium of $G$.
2. Find the Nash equilibrium with highest total expected payoff.
3. Find all Nash equilibria of $G$.
4. Does there exist more than one Nash equilibrium?
5. Does there exist a Nash equilibrium where a specified player's expected payoff is above a specified constant?
6. Does there exist a Nash equilibrium with support that intersects/is disjoint from a specified set?

Unfortunately, all the above problems but the first are known to be NP hard [CS1]. The first problem, however, is wide open, and defines the complexity class NASH . We assume here that the elements of the matrices that define game $G$ are represented as rational numbers in the canonical way. A related complexity class is the class $\text{NASH}_{\{0,1\}}$, which is the subset of the class NASH where the matrices are restricted to have values in $\{0,1\}$.

The main result of this section is that there is a polynomial reduction from NASH to $\text{NASH}_{\{0,1\}}$.

## 3.1 Reduction to Mimicking Games

The first step of our reduction is the well-known reduction from general games $(R, C)$ to *mimicking* games $(M, I)$, where $I$ is an identity matrix. These games are called "mimicking" because the payoff of the second player is non-zero iff she plays the same move as the first player. This implies the following simple lemma.

**Lemma 10** *In any Nash equilibrium $(x^*, y^*)$ of the mimicking game $(M, I)$, Supp $(y^*) \subset$ Supp $(x^*)$.*

**Proof:** Suppose for the sake of contradiction that the second player sometimes plays a strategy not in the support of $x^*$. The second player will get 0 payoff in this case, and could profitably change her play to mimic some strategy in $x^*$. Thus $(x^*, y^*)$ is not a Nash equilibrium, the desired contradiction. ▌

The next lemma trivially implies the reduction to mimicking games.

**Lemma 11** *For any bimatrix game $(R, C)$ with $R$ and $C$ $m \times n$ matrices, there is a mimicking game $(M, I)$ with $M = \begin{pmatrix} 0 & C^T \\ R & 0 \end{pmatrix}$ and $I$ the $(m+n) \times (m+n)$ identity matrix such that the Nash equilibria of $(R, C)$ correspond exactly to the Nash equilibrium strategies of the column player in the game $(M, I)$.*

**Proof:** (sketch, see [CS2] for details) Given an equilibrium $(x^*, y^*)$ of the game $(M, I)$, we decompose $y^*$ as $y^* = (c^*, r^*)$ where $r^*$ and $c^*$ have $m$ and $n$ entries respectively. We then note that the condition that $(x^*, y^*)$ is an equilibrium of $(M, I)$ trivially implies the conditions that that make $(r^*, c^*)$ an equilibrium of $(R, C)$, after we scale $r^*$ and $c^*$ to have sum 1.

Similarly, if $(r^*, c^*)$ is an equilibrium of $(R, C)$, we may scale the two vectors to vectors $(\alpha r^*, \beta c^*)$ with $\alpha, \beta > 0$ so that the scaled incentives are equal:

$$\alpha \max_i (Cr^{*T})_i = \beta \max_i (Rc^{*T})_i,$$

and so that $(\alpha r^*, \beta c^*)$ sums to 1. Let $y^* = (\alpha r^*, \beta c^*)$ and let $x^*$ be uniform on the support of $y^*$. It is straightforward to show that $(x^*, y^*)$ is a Nash equilibrium of the game $(M, I)$. ▌

The above lemma implies that NASH is equivalent the the mimicking-game version of NASH .

## 3.2 Reduction to $\{0, 1\}$ Games

We now describe the reduction to $\{0, 1\}$ games. The first ingredient of our construction is a method of representing each rational entry of a payoff matrix using only zeros and ones.

We first observe that given an $n \times n$ game $(M, I)$, we can scale $M$ by any positive constant, or add any number to its entries without changing the Nash equilibrium strategies by Lemma 6. Thus we may put all the rational entries in $M$ under a common denominator to produce a new matrix $M'$ whose entries are integers. We further note that while each entry may now take more bits to express, the number of new bits that are needed per entry is at most the number of bits in the common denominator, which is at most the number of bits needed to express $M$. Thus the total number of bits has increased at most quadratically.

We now have a game $(M', I)$ where each entry is integral. We will express each entry in binary by replacing the $n \times n$ matrix $M'$ with an $kn \times kn$ block matrix $M''$, for some $k$ bigger than the binary length of any entry in $M'$. We then encode each entry of $M'$ into a binary string, and place it in the corresponding $k \times k$ block of $M''$, using the rest of the entries to ensure that the binary string is correctly "interpreted" as representing an integer payoff.

We now describe the part of the construction that enforces this interpretation: a $(k-1) \times (k-1)$ matrix $G$ with the property that the game $(G, I)$ for a $(k-1) \times (k-1)$ identity matrix $I$ has a *unique* Nash equilibrium $(r^*, c^*)$, and furthermore, there exist $\frac{k-1}{6}$ columns whose probabilities of being played are in the ratio

$$1 : 2 : 4 : ... : 2^{\frac{k-1}{6}}.$$

This property is proven in the following lemma.

**Lemma 12** *Define matrices $A, B$ as*

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}.$$

*For $j = \frac{k-1}{6}$ define the $\frac{k-1}{2} \times \frac{k-1}{2}$ matrix $G'$ to have the following $j \times j$ block form:*

$$G' = \begin{pmatrix} A & A & \cdots & A & B \\ A & A & \cdots & B & 0 \\ \vdots & \vdots & \iddots & \vdots & \vdots \\ A & B & \cdots & 0 & 0 \\ B & 0 & \cdots & 0 & 0 \end{pmatrix}.$$

*Explicitly, $G'$ has block $B$ on the minor diagonal, block $A$ above, and 0 below. Further, let* $G = \begin{pmatrix} 0 & 1 - G'^T \\ G' & 0 \end{pmatrix}$. *Then the game $(G, I)$ (equivalent to the constant-sum game $(G', 1 - G')$ has a unique equilibrium $(r^*, c^*)$ with*

$$r^* = \frac{1}{6j}(1, 1, ..., 1)$$

*and*

$$c^* = (\frac{2 \cdot 2^j - 3}{3 \cdot 2^j - 3} v, \frac{2^j}{3 \cdot 2^j - 3} v),$$

*where* $v = \frac{1}{3 \cdot 2^j - 3}(2^{j-1}, 2^{j-1}, 2^{j-1}, 2^{j-2}, 2^{j-2}, 2^{j-2}, ..., 2, 2, 2, 1, 1, 1)$.

**Proof:** Consider the game $(G', 1 - G')$. This will have a Nash equilibrium $(x, y)$ with full support only if all entries of $G'y^T$ and all entries of $x(1 - G')$ are equal.

We show by induction that these constraints are satisfied iff both $x$ and $y$ are scaled versions of $v$. Suppose we know the first $3i$ entries of $y$ are in the proportions of $v$. Then in the $i + 1$st block row of $G'$, we have three rows, which must all have equal incentives for the row player since the row player's strategy has full support, by hypothesis. For each of those rows, the incentives from the $A$-blocks are the same by the inductive hypotheses, so we must have that the payoffs from the $B$-block are also the same, i.e.

$$y_{3i+1} + y_{3i+2} = y_{3i+1} + y_{3i+3} = y_{3i+2} + y_{3i+3},$$

which implies $y_{3i+1} = y_{3i+2} = y_{3i+3}$. To show the ratio of $2 : 1$ between adjacent blocks, we note that the payoff for this block of rows must be the same as that for the previous one; the payoffs between the two blocks differ by $2y_{3i} - (y_{3i} + 2y_{3i+3})$; setting that to zero proves the inductive step. Thus we have shown that the entries of $y$ are in the ratio specified by $v$.

Clearly, the same argument applies to $x$. Thus since both $x$ and $y$ must sum to 1, $x = y = v$ is the only Nash equilibrium with full support.

We now note that the game $(G', 1 - G')$ is in fact a constant-sum game, so its Nash equilibria are the solutions to a linear program. This implies that the set of Nash equilibria is convex. If we suppose for the sake of contradiction that there is another Nash equilibrium in addition to the one $x = y = v$, then all linear combinations of these two equilibria must also be equilibria, and hence by standard topology arguments there are a continuum of full support equilibria, which contradicts the uniqueness argument of the previous paragraph.

Thus $x = y = v$ is the unique equilibrium for the game $(G', 1 - G')$, and by lemma 11 the game $(G, I)$ has the unique equilibrium described above. ▌

Thus we can express integers $t$ in our game by representing them in binary as $(t_{j-1}...t_1t_0)$, and putting digit $t_i$ in column $3(j - i)$ of a matrix based on $G$.

We now show how to embed the matrix $G$ in larger games so as to allow the binary representation described above.

Given a game $(M', I)$, with $M'$ an $n \times n$ matrix with integral entries, construct the $kn \times kn$ $\{0, 1\}$ -matrix $M''$ as follows. Construct the $k - 1 \times k - 1$ matrix $G$ defined in the above lemma, and append a column of $k - 1$ ones to the right end to create a $k - 1 \times k$ matrix $\bar{G}$. Place this matrix along the main diagonal of $M''$, (with the upper-left corner on the diagonal) filling the rest of these $(k - 1)n$ rows with zeros. Note that this leaves $n$ rows unaccounted for. Since $M''$ may be considered as a $n \times n$ block matrix, we fill in each block's $k$ unspecified entries with the binary expression of the corresponding entry in the $n \times n$ matrix $M'$, putting the $i$th digit of entry $M'_{r,s}$ in $M''_{kr,k(s-1)+3(j-i)}$ as described above. We make the slight modification of making first two entries in these rows one, $M''_{kr,k(s-1)+1} = M''_{kr,k(s-1)+2} = 1$, so that the number represented is actually $M'_{r,s} + 2^j$. The rest of the entries are 0.

We prove the following lemma, which implies our main result.

**Lemma 13** *Given any game $(M, I)$ with $M$ rational, construct the game $(M'', I')$ by first rescaling $M$ to an integral matrix $M'$, and then shifting $M'$ so as to make its entries integers in the range $0 < M'_{r,s} < 2^j$ for some $j$. Let $k = 6j + 1$, and construct the matrix $M''$ as above. Then, up to scaling, the Nash equilibrium strategies for the column player of the game $(M, I)$ are* identical *to the elements $(1, k+1, ..., (n-1)k+1)$ of the Nash equilibrium strategies for the column player of the $\{0, 1\}$ -game $(M'', I')$.*

We note that since the size of $M''$ is polynomial in the number of bits used to express $M$, and lemma 11 proves that a mimicking game $M = \begin{pmatrix} 0 & C^T \\ R & 0 \end{pmatrix}$ may be constructed so that the Nash equilibrium strategies for its column player correspond exactly to the Nash equilibrium strategies for both players in the game $(R, C)$, this lemma trivially implies the following theorem.

**Theorem 14** $NASH = NASH_{\{0,1\}}$.

We now prove the lemma.

**Proof:** Consider any Nash equilibrium $(x', y')$ of the game $(M'', I')$. Motivated by the block decomposition of $M''$, we consider $y'$ in blocks of $k$. Recall from the construction of $M''$ that each occurrence of $\bar{G}$ stands alone in its corresponding rows, and that the corresponding entries in $I'$ form a $k-1 \times k-1$ identity matrix with a column of zeros added. A straightforward application of the definition of a Nash equilibrium reveals that the $k-1$ corresponding weights in $y'$ are either all zero, or are a scaled Nash equilibrium for the game $(G, I)$. Thus these $k-1$ weights $\{y'_{(r-1)k+i}\}_{i=1}^{k-1}$ equal $y''_r(2^{j-1}, 2^{j-1}, 2^{j-1}, ..., 1, 1, 1, ...)$ for some $y''_r \geq 0$ as shown in lemma 12. Thus we have found a block representation for any Nash equilibrium strategy $y'$. Note, however, that we have not yet discussed the $n$ entries $y'_{(r-1)k+k} = y'_{rk}$.

Recall from lemma 10 that $y'_{(r-1)k+i} > 0$ implies that $x'_{(r-1)k+i} > 0$, which implies from the definition of a Nash equilibrium that the $((r-1)k+i)$th entry of $M''y'^T$ is at least as big as any other entry. We apply this technique to prove a sequence of useful results.

Suppose for the sake of contradiction that for every $r$, $y''_r = 0$. This means that the only nonzero entries are those of the form $y'_{rk}$. Note, however, that this implies that $(rk)$th entries of $M''y'^T$ are all 0, since for any $r, s$, $M''_{rk,sk} = 0$. However, the $(rk)$-th columns have ones everywhere else, so every other row gets positive payoff. Thus this is not a Nash equilibrium. We conclude that some $y''_r > 0$.

For some $y''_r > 0$, consider the payoffs of the $k$ rows $(r-1)k+1, ..., rk$. From the construction of the matrix $G$, we conclude that the first $k-1$ of these payoffs equal $2^j y''_r + r'_{rk}$ and that the last payoff is at least the sum of the entries in the $k$ corresponding columns: $(M'_{r,r} + 2^j)y''_r$, where by construction, $M'_{r,r} > 0$. Thus this may be a Nash equilibrium only if $r'_{rk} > 0$. Thus whenever $y''_r > 0$, we must have $y'_{rk} > 0$. The crucial consequence of this is that the payoff of the $(rk)$th row must now be optimal by the mimicking argument.

Note that the incentive of the $(rk)$th row is just

$$\sum_s (M'_{r,s} + 2^j) y''_s, \tag{1}$$

which equals the incentive in the game $(M' + 2^j, I)$ when the second player plays strategy $y''_s$ (up to scaling). Since as noted above, for every nonzero $y''_r$, the corresponding row must have optimal incentive, we conclude that the strategy $y''_s$, properly scaled, is in fact a Nash equilibrium of the game $(M, I)$. We have proven one direction of the correspondence.

The other direction is fairly straightforward. Given a Nash equilibrium $(x, y)$ of the game $(M, I)$, let $y'' = \alpha y$ for some scaling constant $\alpha$, and let

$$\{y'_{(r-1)k+i}\}_{i=1}^{k-1} = y''_r(2^{j-1}, 2^{j-1}, 2^{j-1}, ..., 1, 1, 1, ...),$$

as above. From equation 1 we see that all the optimal incentives in $(M, I)$ will remain optimal in $(M'', I')$ when we restrict our attention to rows $rk$. Further, each $k-1$ block of rows will have equal payoffs since their corresponding columns have probabilities proportional to the full-support Nash equilibrium of the game $(G, I)$. To make all these blocks have equal payoffs, we need only pick the additive constants $y'_{rk}$ so that the total payoffs are equal. We then scale these $y'_{rk}$ and $\alpha$ so as to make $\sum_i y'_i = 1$, and we have a Nash equilibrium, as desired.

Thus we have constructed the desired correspondence between the column player strategies in Nash equilibria of the games $(M, I)$ and $(M'', I')$. ▌

These results suggest that the problem of finding a Nash equilibrium may be easier than was previously thought. In the next section, however, we provide some intuition showing how the problem may still be hard.

## 4 Why NASH may be Hard

### 4.1 The Lemke-Howson Algorithm

We now explicate the Lemke-Howson algorithm for finding a Nash equilibrium in a bimatrix game, which is the standard algorithm used in practice. This algorithm is perhaps the canonical way to find an equilibrium, and has the further property that the construction of the algorithm yields a surprising proof of the existence of Nash equilibria. This style of proof has been generalized to define the complexity class PPAD , which we discuss later.

Consider a mimicking game $(M, I)$. As a trivial consequence of Lemma 8, a pair of equilibrium strategies $(x^*, y^*)$ exists iff for every column $i$ in the support of $y^*$, the $i$th entry of $My^*$ is optimal. In such a case, we call $i$ a *best response.*

For the remainder of this section, we shall assume that $M$ is *nondegenerate*, by which we mean that for any strategy $y$, the number of pure best responses is at most the size of the support. This notion is a slightly stronger than a condition that $M$ have full rank. In practice, there are methods of adding slight perturbations to $M$ to ensure nondegeneracy while preserving Nash equilibria. See for example [Ste1] We further assume without loss of generality that all entries in $M$ are positive, since adding a constant to each entry does not change the game, by Lemma 6

We may thus rephrase the above condition to say that each pure strategy is either not played, or is a best response. This motivates the idea of labelling a strategy $y$ as follows: for each $i$, label $y$ with $i$ if either $i$ is never played, or is a best response. A strategy $y^*$ represents a Nash equilibrium pair $(x^*, y^*)$ if it is *completely labelled.* We say that a strategy $y$ is *k-almost completely labelled* if $y$ has every label except for possibly $k$.

We now express these conditions algebraically. Given a vector $y^*$ that satisfies the above conditions, we can scale it to a vector $z = \mu y^*$ so that $Mz \leq 1$, with equality only for the rows in the support. Thus for any row $i$, either $z_i = 0$ or $(1 - Mz)_i = 0$. Thus the above conditions imply

$$z \geq 0, \quad z \neq 0, \quad Mz \leq 1, \quad z^T(1 - Mz) = 0.$$

We now observe that we can transform any $z$ satisfying the above *linear complementarity problem* (LCP), there exists a scaled version $y^*$ that produces a Nash equilibrium. Specifically, the third condition, that $Mz \leq 1$ implies that the set of best response rows are those for which $Mz = 1$. Thus the fourth condition implies that every index is either not in the support, or a best response. The first two conditions imply that $z$ can be scaled to a proper probability distribution.

It is important to note here that the second condition, that $z$ is not uniformly 0, is not one that we can enforce with quadratic programming, and that we must therefore remove this condition. Upon doing so, however, we note that we now have a trivial solution to the LCP, namely the all zero solution. The problem now becomes to find a *second* solution.

We now have the LCP

$$z \geq 0, \quad Mz \leq 1, \quad z^T(1 - Mz) = 0,$$

which may be interpreted geometrically as follows. Given a $n \times n$ matrix $M$, the first two conditions define a polytope in $\mathbb{R}^n$ with $2n$ $(n-1)$-dimensional hyperfaces, or more specifically, $n$ pairs of hyperfaces. The third condition then labels a point by the indices of the hyperfaces to which it belongs, and asks for completely labelled points. Assuming nondegeneracy, each vertex has exactly $n$ (possibly duplicate) labels, and each edge has $n - 1$.

The Lemke-Howson algorithm starts at the trivial completely labelled vertex $z = 0$, and picks an arbitrary edge leading from this vertex. Since the points on this edge still have $n-1$ coordinates 0, this edge will be $k$-almost completely labelled, for the remaining coordinate $k$. From here on, the algorithm is deterministic, and runs as follows: given a $k$-almost completely labelled edge, where we have already processed one endpoint, consider the other endpoint. If it picks up the label $k$, then we are done. Otherwise, it must have picked up a duplicate label. To fix this, we find the hyperface corresponding to that duplicate label, and find the (unique) edge going away from this hyperface. Repeat until a fully labelled vertex is found.

We note that this algorithm defines a unique path from $z = 0$. Further, the algorithm may equivalently be run in reverse, with the same arguments implying that there is at most one (k-almost completely labelled) edge going into any vertex. We note that the $k$-almost completely labelled edges are truly directed, and we may define their direction by the sign of a determinant, the details of which are not important here.

Thus given a matrix $M$ and an index $k$, we define an implicit directed graph as follows: the vertices of the graph are the vertices of the polytope, which may be represented by $2n$-bit strings denoting which hyperfaces they are members of, that are $k$-almost completely labelled. The edges of the graph are as defined above, and satisfy the properties that the in-degree and out-degree of any vertex is at most one, and these edges may be computed (in polynomial time) given the vertex. We are given one endpoint, $z = 0$, with in-degree 0, and asked to find another endpoint. The Lemke-Howson algorithm does this by simply following the path until it reaches a sink. Note that there cannot be any cycles.

The above explication essentially proves that Nash equilibria exist, and is a version of a more general argument called the *directed parity argument* that goes as follows: A di-graph $G$ with in-degree and out-degree at most 1 is a disjoint union of paths and cycles. Specifically, the number of endpoints must be even. Thus, given one endpoint, there must be another.

We note that the "algorithmic" part of the Lemke-Howson algorithm, namely the decision to trace a single path, is more a heuristic than an algorithm, as the length of the path that starts at $z = 0$ is not guaranteed to be sub-exponential, and we could just as easily traverse the vertices in any other order. Indeed, it is an open problem whether Nash equilibria may be found in polynomial time, but it is known that the Lemke Howson algorithm fails at this task [SS].

## 4.2 The Class PPAD

As mentioned above, we may cast the Nash equilibrium problem for some matrix $M$ in graph theory terms by defining the $k$-almost completely labelled subgraph of the edge graph of the induced polytope, and try to find a nontrivial endpoint. This problem falls in the complexity class known as PPAD , defined in [Pap2] which is the directed version of the class PPA , an acronym for the *polynomial parity argument*. The class PPAD consists of those problems expressible in the following terms.

Given a set of strings $\{0,1\}^k$, we implicitly define a digraph $G = (V, E)$ of in and out-degree at most one as follows. We are given a *vertex recognition* algorithm, that in polynomial time returns whether or not a string in $\{0,1\}^k$ is a vertex in $V$. Further, given a vertex $v \in V$, we are given a pair of polynomial time algorithms that find the (at most one) edge going into and out of $v$ respectively. We are further given an algorithm to find a starting vertex of total degree 1. Given such a graph, the class PPAD asks us to find another vertex of total degree 1. We note that the vertex recognition algorithm is not strictly necessary, since we may incorporate it into the edge finding algorithms, so that "fake" vertices lack any edges.

Thus far, the class PPAD is not known to be in P , and is also not known to be reducible to any canonical "hard" problems. However, if the definition of PPAD is modified slightly in any of several ways, then many standard hard problems result. We discuss this below.

Problems in PPAD are characterized by three abilities: given a vertex on a path we can go forward, and we can go back; further we can find a starting vertex. We consider what happens if we relax any of these conditions.

Intuitively, it would appear that being given a vertex with in-degree 0 does not help, and that we might even prefer a vertex with in-degree 1, since the vertex with in-degree 0 is as far as possible from the other end of the path. However, this modification would render PPAD as hard as the search version of NP , TFNP , as evidenced by the following construction. Consider some polynomial-time predicate $F_Y(X)$, with $X \in \{0,1\}^k$ and data $Y$, such that we want to find a satisfying $X$. Connect each string $X$ to the strings lexicographically before and after it, wrapping around at the ends, with the exception that nothing is connected to satisfying strings. Thus finding a source or sink is as hard as finding a satisfying $X$.

Another way to modify the definition of PPAD is to remove our ability to go backwards. Intuitively, this ability does not seem that useful. However, if one-way permutations $p(x)$ exist — a fairly standard complexity assumption — then we can define the successor of a node $v$ to be the image of that node under permutation $p(v)$ unless the image is some specified randomly drawn number $x$, in which case we call $v$ a sink. Then given the source $x$, the problem of finding the sink is equivalent to finding $p^{-1}(x)$, a hard problem.

A third reasonable modification to the definition would be to insist that the vertex returned be at the other end of the path of the initial vertex. However, finding such a vertex is PSPACE hard, as we can encode the execution of a Turing machine into the di-graph. We outline this construction below, as it has some important consequences.

It is easy and quite standard to encode the possible states of a Turing machine as strings in $\{0,1\}^k$. Further, given a state, we can easily compute the successor of a state by simulating the Turing machine for one iteration. However, in general it is hard to compute the predecessor of a Turing machine state.

Towards this end, we introduce the notion of *reversible* computation. A computation is reversible if from each state we can uniquely compute the next state and the previous state in polynomial time. As an interesting side note, this notion was introduced by physicists who observed that the energy use of a computer can by lower-bounded by the number of *irreversible* operations, so that if a computer were designed using only reversible operations, it could run on negligible power.

## 4.3 The Pebble Game

Suppose we wish to simulate a Turing machine in a reversible manner. At any instant in our simulation, we will store the complete memory state of the Turing machine at a selection of times $t_1, ..., t_n$ for some $n$. Given that we know the memory state at time $t_i$, one reversible operation that we will allow ourselves is the ability to compute the memory state at time $t_i + 1$ and store it in some empty memory slot $t_j$. This operation is reversible since the reverse of this is just clearing the memory slot $t_j$. We further allow ourselves the ability to run the reverse of this operation, namely given a record of the machine at time $t_j$, we may delete it if and only if for some $i$, $t_i + 1 = t_j$ and we have a record of the machine at state $t_i$. These are the two operations we allow ourselves. We now consider how much reversible time and space it takes to simulate a Turing machine using time and space $(T, S)$, and show that

$$\text{Turing-}(T, S) \subset \text{reversible-}(T^{log_2 3}, S \log_2 T).$$

Consider the following *pebble game*, sometimes called the *east model*[DGC]. We are given a handful of $n$ pebbles, and a sequence of squares, $\{a_i | i > 0\}$, each of which may be occupied by a pebble. The rules of the game are as follows: we may add or remove a pebble at square $i + 1$ only if there is a pebble at square $i$, or $i = 0$. The question is how far we can get with $n$ pebbles, and in how many moves. This is clearly analogous to the above model of reversible computation,' — we let a pebble at position $i$ correspond to storing the state of the simulated machine at time $i$.

We now introduce an algorithm to get a pebble out to position $2^n - 1$, which is optimal, using the optimal time $\frac{1}{2}(3^n - 1)$. Optimality is proven in [Ben]. The construction is recursive, with the base case of one pebble consisting of placing the pebble in position 1. Given a construction to get a pebble to position $2^n - 1$ in time $\frac{1}{2}(3^n - 1)$ using $n$ pebbles, consider the following procedure: run the construction once; then place the $n+1$st pebble in position $2^n$; then reverse the construction to remove all the pebbles but this from the board; then run the construction shifted $2^n$ stones to the right, to place a stone at $2^{n+1} - 1$ in time $\frac{1}{2}(3^{n+1} - 1)$, as desired.

Thus we may construct an instance of PPAD where the vertices represent positions in the pebble game, namely a record of up to $n$ states of a Turing machine $M$; we connect each position to the positions immediately before and after it in the optimal pebble game strategy. Thus if we let $M$ be a Turing machine that solves a PSPACE complete problem, then the other endpoint of the path starting at its input will be PSPACE hard to find.

We note that this does not prove that PPAD itself is hard, since our construction provides no way to verify whether a vertex indeed lies on this path, and the PPAD may have many trivial solutions corresponding to "fake" pebble game simulations. We address this in the next section.

### 4.4 Half-Baked Hardness Constructions for PPAD

Supposing we had a polynomial-time function $f : \{0, 1\}^k \to \{0, 1\}^k$ with some designated generator $e \in \{0, 1\}^k$ with the following two properties:

- $f^n(e)$ is *hard* to compute for some exponentially large $n = N$.
- We may verify that $f^n(e) = y$ in polynomial time, for any $n$.

In this case, we construct the following PPAD . Let each vertex be a sequence of pairs $(t_i, f^{t_i}(e))$ such that the set $\{t_i\}$ represents a valid position in the optimal pebble game play, and such that $\forall i, t_i \leq N$. To form a graph, we connect each pebble position to the positions immediately preceding and following it in optimal play.

From [DGC] we can easily determine whether the sequence $\{t_i\}$ appears in optimal play of the pebble game, and from the second property of $f$ we can ensure that only valid pairs $(t_i, f^{t_i}(e))$ appear. Thus we have constructed an instance of PPAD where the graph consists of a single path, which starts at a position computable from input $e$, and ends at a position from which we could compute $f^N(e)$. Thus any algorithm for PPAD must solve the *hard* problem of finding $f^N(e)$.

We provide an incomplete construction of such an $f$. Define a *binary block* of length $2^j$ to be a sequence of $2^j$ integers such that all but the last $j$ binary digits of each number are equal.

We present a construction for $f$ if a function $g : \{0,1\}^k \to \{0,1,...,k\}$ exists with the following properties:

- $g^{-1}(k)$ is *hard.*
- For exactly one $x$ in any binary block of length $2^j$, $0 \leq j \leq k$, is $g(x) \geq j$.

Thus $g(x)$ is $k$ for one $x$, is at least $k-1$ for two $x$s, is at least $k-2$ for four $x$s, etc. This special form of $g$ enables us to find a simple proof of statements of the form

$$\max_{i<n} g(i) = y$$

in the following manner: for any binary block of size $2^j$ we can prove the maximum value of $g$ in that block by exhibiting the $x$ for which $g(x) \geq j$. Thus, by subdividing the integers between 0 and $n$ into at most $k$ binary blocks, we can construct a polynomial time proof of the value of $\max_{i<n} g(i)$. We further note that if we iterate through $n$, such a proof may be incrementally constructed by keeping track of the best value of $g(x)$ found so far in each block.

Thus, if such a $g$ exists, we could define $f^n(e)$ to consist of this proof. Thus, since these proofs are incrementally constructed, $f$ is polynomial-time computable. Further, by definition we can verify the value of $f^n(e)$ for any $n$. And finally, we can read off from $f^{2^k}(e)$ the value of $g^{-1}(k)$, which was hard by assumption. Thus, if such a $g$ exists, then an $f$ exists, and PPAD is hard.

## 4.5 Obfuscating $g$

Suppose we weaken the first condition on $g$ to the following:

- $g^{-1}(k)$ is hard to compute given only oracle access to $g$.

We note that we can easily construct such a $g$ if we are given a strong (pseudo-) random bit generator. Namely, for each $j$, $1 \leq j \leq k$, and each binary block of size $2^j$ in $\{0,1\}^k$, flip a coin, and assign this value to the block (note that we do not have to store these bits since they can be computed using the pseudo-random bit generator). Thus each $x \in \{0,1\}^k$ is contained in $k$ binary blocks, one for each $j$. Then to evaluate $g(x)$, compute the bits assigned to each of the $k$ binary blocks containing $x$, and interpret these $k$ bits as a $k$-bit binary number $\hat{x}$, by letting the bit from the block of size $2^j$ correspond to the $j$th bit from the right in $\hat{x}$. Let $g(x)$ be the number of trailing ones in

$$x \text{ xor } \hat{x}.$$

From a trivial induction argument, $g$ satisfies its second property. We now argue that $g^{-1}(j)$ is hard to compute for any length $2^j$ binary block when given only oracle access to $g$. We note that until we have found an $x$ such that $g(x) \geq j-1$, none of the values of $g$ will depend on the binary bit pseudo-randomly chosen for this block, and hence the parts of the algorithm that access the left and right halves of this block may as well run independently, and we may as well just examine one half (of size $2^{j-1}$) first. Suppose it takes expected time $T(j-1)$ to find an $x$ with $g(x) \geq j-1$ in one of the two halves. If this $g(x)$ is in fact at least $j$, then we are also done for the whole block. However, with probability half, the $x$ for which $g(x) \geq j$ is in the other half, and we need to expend $T(j-1)$ additional effort to find it. Thus

$$T(j) = \frac{3}{2}T(j-1),$$

and we conclude that finding $g^{-1}(k)$ is exponentially hard, given only oracle access to a $g$ generated with strongly random bits.

However, we have not been able to prove similar results when the player is given the code to $g$. Indeed, as mentioned above, such results would imply that PPAD is NP -hard. If such a $g$ did not exist, however, then we would have a curious distinction between what we could compute given oracle access to a function $g$ as in the previous paragraph, and what we could compute given *any* Turing machine implementation. Indeed this distinction is at the heart of the fundamental problem of *software obfuscation.*

The goal of software obfuscation is to derive a procedure such that given a function defined by a Turing machine that computes it, we can polynomially transform the Turing machine into another Turing machine that computes the same function, but acts as a *virtual black box.* A Turing machine $P$ is a *virtual black box* if knowing the code of $P$ allows us no more abilities than having only access to an evaluation oracle $\mathscr{O}(P)$. This is a formalization of the heuristic that "the only useful thing you can do with software is run it," an impression which most major software companies have huge interest in promoting. If this were false, then any software that accomplishes some given task might be reverse engineered

It was a major result of [BGI] that there exist functions that cannot be obfuscated. However, these functions were constructed similarly to the canonical uncomputable function that simulates itself, and thus do not provide a natural class of un-obfuscatable functions. Thus it might be hoped that the impossibility of software obfuscation applies only to such contrived examples.

The function $g$ described above has no such general simulation abilities, and is the kind of pseudo-random function computed in a variety of software. We have thus shown that either PPAD is hard, pseudo-random numbers may not be generated, or that $g$ is un-obfuscatable.

We note that it remains open whether NASH is complete in PPAD .

## 5 Finding a Nash Equilibrium by Support Enumeration

We provide an interesting well-known algorithm for finding a Nash equilibrium of a bimatrix game, in exponential time, which relies on the following result:

**Proposition 15** *Given that there exists a Nash equilibrium with supports $S1 = Supp\ (x)$ and $S2 = Supp\ (y)$, there is a polynomial time algorithm to compute a Nash equilibrium with exactly those supports.*

**Proof:** Let $R_i$ be the $i$th row of $R$, and $C_j$ be the $j$th column of $C$.
We solve the following linear program on $2n + 3$ variables:
Variables: $x_i, y_i \geq 0, U, V, \delta$
Maximize: $\delta$
Constraints:
$x_i = 0$ if $i \notin S1$, $y_i = 0$ if $i \notin S2$,
$\sum_{i=1}^{m} x_i = \sum_{i=1}^{n} y_i = 1$ if $i \in S1$
$x_i \geq \delta$ if $i \in S1$, $y_i \geq \delta$ if $i \in S2$.

for each row $i$
$R_i y = U$ if $i \in S1$
$R_i y \leq U$ if $i \notin S1$
for each column $j$
$x^T C_j = V$ if $j \in S2$
$x^T C_j \leq V$ if $j \notin S2$

We show that any solution to those conditions is a Nash equilibrium with supports $(S1, S2)$. The first set of constraints require that the probabilities $x_i$ are non-negative and sum to one; further, they are 0 outside the desired support, and at least $\delta$ inside the desired support. We will show that if a Nash equilibrium with that support exists, then $\delta > 0$, so that the $x_i$ give a strategy that has the desired support.

The second set of conditions requires that each of the rows in the support is a best response to $y$, since they obtain $U$, the payoff for the row player, and no other pure strategy gives a higher payoff than $U$. This implies $x$ is a best response to $y$, by Lemma 8.

We also have the equivalent symmetric conditions for $y$. Since each player's strategy is a best response to the other's, any solution to the LP is a Nash equilibrium.

The linear program has a solution since the Nash equilibrium we were given has supports $(S1, S2)$, and thus obviously satisfies all those constraints, with $\delta > 0$ since each of the $x_i$ are for $i \in S1$. Linear programming is solvable in polynomial time (using, for example, the ellipsoid algorithm), so our claim holds. ∎

**Corollary 16** *There exists an exponential-time algorithm to find a Nash equilibrium of a bimatrix game.*

**Proof:** The algorithm is simple – enumerate all pairs $(S1, S2)$ where $S1$ is a subset of the pure strategies for the row player, and $S2$ for the column player. For each such pair use the above algorithm to find a Nash equilibrium (if one exists) with those supports. If no Nash equilibrium exists with those supports, the algorithm will still terminate in polynomial time, and will either assert no solution exists, or find one with $\delta = 0$. In the latter case, the solution will be a valid Nash equilibrium, but it will not have the support we input, which for this purpose is irrelevant. There always exists a Nash equilibrium, so the algorithm will necessarily find a Nash equilibrium when it uses the previous algorithm on the support of the Nash equilibrium.

There are at most $2^m \times 2^n$ such pairs of sets, so we get total $(n+m)^{O(1)} 2^{(m+n)}$ total runtime. ∎

A powerful consequence of the linear programming formulation is that we can require additional properties of our Nash equilibrium through adding additional constraints to the linear program. For example, we could require the Nash equilibrium to have each pure strategy played by the row player be played with probability at least $\frac{1}{2k}$, where $k$ is the size of the support (say, if the row player doesn't want to use a complicated strategy), and our algorithm would find a Nash equilibrium with that additional property, if it exists. Many of the properties of restricted classes of Nash equilibria that are shown to be NP-hard to find in 3 are similarly trivial to find in exponential time using this linear programming formulation, without having to actually find all Nash equilibria, which could be unpleasant since the set of Nash equilibria can in general be infinite (in cases with degeneracy).

## 6 Approximate Nash Equilibria

### 6.1 A Pseudopolynomial Algorithm

Here we exposit several results from the excellent paper [LMM].

In general, Nash equilibria can be very complicated. In particular, there exist games in which there is a unique Nash equilibrium of full support (for an example, see Lemma 12). In the real world, one does not want to play such a complicated strategy. It is thus reasonable to consider sacrificing a small amount of the payoff in a game in exchange for the advantage of playing a simple strategy. The first notion of simplicity that might come to mind is having small support. However, we also don't want to have strategies that use some pure strategies with very low probability, since then it is likely the cost of implementing that option is greater than the value it provides. Thus, one comes to develop the notion of a $k$-uniform strategy $x$, where each strategy is played with probability an integral multiple of $\frac{1}{k}$.

One also must consider what we mean by an approximation of a Nash equilibrium. The commonly accepted notion of a good approximation to a Nash equilibrium is as follows.

**Definition 17** *An $\epsilon-$*Nash equilibrium *is a pair of strategies $(x', y')$ such that for any strategies $x, y$, we have that*

$$x^T Ry' \leq x'^T Ry' + \epsilon \text{ and } x'^T Ry \leq x'^T Ry' + \epsilon$$

The real-world application might be two countries that sign a treaty restricting them to a certain subset of the pure strategies, and they'd want to make sure that there is a treaty where both countries get almost the same payoffs in the game, with a $k$-uniform strategy. The amazing result in [LMM] is that there always exists a $k$-uniform $\epsilon$-Nash equilibrium with $k$ logarithmic in the size of the game.

**Theorem 18 (Lipton, Markakis, Mehta)** *Assume that $R$ and $C$ have entries all in $[0, 1]$, and are both square of size $n \times n$. Then for any Nash equilibrium $(x^*, y^*)$, and any real number $\epsilon$, there exists a pair $(x', y')$ of $k$-uniform strategies (for $k \geq \frac{12 \ln n}{\epsilon^2}$) such that*

*(1) $(x', y')$ is an $\epsilon$-Nash equilibrium*

*(2) $|x'^T Ry' - x^{*T} Ry^*| \leq \epsilon$ (the row player's payoff is within $\epsilon$ of the equilibrium payoff)*

*(3) $|x'^T Cy' - x^{*T} Cy^*| \leq \epsilon$ (the column player's payoff is within $\epsilon$ of the equilibrium payoff)*

**Proof:** The argument uses the probabilistic method – we prove that an $\epsilon$-Nash equilibrium exists by giving a construction with a nonzero probability of producing it. First, pick an integer $k \geq \frac{12 \ln n}{\epsilon^2}$. Then, sample $k$ values independently at random from the Nash equilibrium probability distribution $x^*$, to form the multiset $A$. Construct the distribution $x'$ from $A$ by giving each of the pure strategies in $A$ probability $\frac{r}{k}$, where $r$ is the number of times that strategy appears in $A$. Similarly, construct the strategy $y'$.

Now, one would expect that the distributions $x'$ and $y'$ would be close to the Nash equilibrium distributions $x^*$ and $y^*$. In fact, one can prove through a rather straightforward argument that with positive probability, they do in fact form a $k$-uniform $\epsilon$-Nash equilibrium. It follows that such an equilibrium must exist, for if it did not, then that probability would be zero. ▌

**Corollary 19** *There exists an algorithm that finds an $\epsilon$-Nash equilibrium for a $[0, 1]$ game in $O(n^{1+\frac{24 \ln n}{\epsilon^2}})$ time.*

**Proof:** We enumerate all multisets of size at most $k = \frac{12 \ln n}{\epsilon^2}$ for each of the players, and for each of those, simply check whether it is an $\epsilon$-Nash equilibrium by testing whether all deviations to pure strategies do not improve payoffs by more than $\epsilon$.

By the last proposition, a $k$-uniform $\epsilon$-Nash equilibrium must exist, so by an argument similar to that in Corollary 16, our algorithm will find one. There are at most $n^{2k}$ pairs of multisets of our pure strategies of size $k$ each, which amounts to $n^{2k+O(1)}$ runtime, since we can check whether we have an $\epsilon$-Nash equilibrium in $n^{O(1)}$ time.

▌

Note this allows us to find all $k$-uniform $\epsilon$-Nash equilibria in $n^{2k+O(1)}$ time, and we have by our theorem that each exact Nash equilibrium has payoff within $\epsilon$ of that of a $k$-uniform one. Thus, since it is NP-hard to determine whether there exists a Nash equilibria where the row player receives payoff at least some constant, we have the following theorem:

**Theorem 20** *Either there are $\epsilon$-approximate Nash equilibria with payoffs for both players better (by more than $\epsilon$) than the maximum Nash equilibrium payoff, or we have a sub-exponential algorithm for solving satisfiability.*

**Proof:** [CS1] constructs games with the property that for an arbitrary boolean formula $\phi$ in Conjunctive Normal Form, if $\phi$ is satisfiable, there is a Nash equilibrium with payoff 1, and otherwise there is a unique Nash equilibrium of payoff 0. Consider the following algorithm to determine if there is a Nash equilibrium of payoff 1 for the row player in such a game. Pick $\epsilon < 1$, and with that corresponding $k$, compute all $k$-uniform $\epsilon$-Nash equilibria. If some $k$-uniform $\epsilon$-Nash equilibrium has payoff for each player of at least $1-\epsilon$, report that there is one, and otherwise report that none exists. If $\phi$ is satisfiable, the algorithm will report correctly that it is, by Theorem 18. If $\phi$ is not satisfiable, there is a unique Nash equilibrium with payoff 0, so either this algorithm reports correctly that $\phi$ is not satisfiable, or it finds a $k$-uniform $\epsilon$-Nash equilibrium with payoff at least $1-\epsilon$ for each player. Since this algorithm runs in $n^{2k+O(1)}$, and with a choice of $\epsilon < 1$, we have $k = O(\ln n)$, so that our runtime is $n^{O(\ln n)}$. ▌

We've not been able to resolve which of the two possibilities in the previous theorem holds, but we strongly suspect the former. A provably subexponential algorithm for solving satisfiability would be a very exciting result. Let us assume for the moment the more likely possibility that the opposite is true, since it reflects more on game theory, the subject of our paper. This would imply that in the class of games studied by [CS1], there are pairs of strategies where both players receive high payoffs, and both have very small incentives (of size at most $\epsilon$) to change their strategies so as to increase their payoffs, but if they do so, they will fall into an equilibrium with payoff 0.

We can also use the linear programming formulation we gave in Proposition 15, with some modifications, to find the $\epsilon$-Nash equilibrium with support of size at most $k$, and force additional restrictions on our $\epsilon$-Nash equilibria, as before.

Related results that follow directly from those in [LMM] include the following, for which we omit the proof.

**Theorem 21** *If $R$ and $C$ have ranks $r$ and $c$, respectively, then for every Nash equilibrium $(x^*, y^*)$, there exists another Nash equilibrium $(x, y)$ with $|Supp\ (x)| \leq c+1$ and $|Supp\ (y)| \leq r+1$ such that each player receives the same payoff as in the original equilibrium.*

This theorem has the following immediate corollary.

**Corollary 22** *If both of the payoff matrices have constant rank, then there is a polynomial time algorithm to find a Nash equilibrium. In particular, if one of the players has a constant number of pure strategies, then there is a polynomial time algorithm to find a Nash equilibrium.*

**Proof:** The first assertion follows immediately from the previous theorem. The second follows from the first, since the rank of the payoff matrices is bounded by the number of pure strategies of either player. ▌

**Corollary 23** *In a zero-sum game $(R, -R)$, where $R$ has rank $r$, there exists a Nash equilibrium with support including at most $r+1$ pure strategies.*

**Proof:** This follows from the previous theorem, since $R$ and $-R$ have the same rank $r$. ▌

## 6.2 Fixed Parameter Tractable with Respect to Treewidth and Maximum Degree

Here we develop another algorithm for the computation of approximate Nash equilibria. This algorithm works well when the graph of strategies that effect each other has small treewidth $k$, and the number of strategies affecting the payoff of a given strategy is at most $m$, which is also small. For fixed $k$ and $m$, our algorithm has linear dependence on $n$ and has dependence on $\epsilon$ of the form $\log(\frac{1}{\epsilon})(\frac{1}{\epsilon})^{2k+2}$. On the other hand, the algorithm in section 6.1 has dependence in the order of $n^{O(1/\epsilon^2)}$. Hence the algorithm here should be more effective for small $k$, $m$ and $\epsilon$, and large $n$ than the previously known algorithm.

**Definition 24** *For a vector $y$ define the response to $y$, Resp $(y)$ to be the strategy $x$ whose components are $\frac{1}{|Supp\ (y)|}$ on the support of $y$, and 0 elsewhere.*

**Definition 25** *Let $(R, I)$ be a mimicking game. Define a strategy $x$ for the column player to be an $\epsilon$-relative Nash equilibrium for some $\epsilon > 0$ if Resp $(y)^T Ry(1 + \epsilon) \geq x^T Ry$ for any strategy $x$.*

The idea of an $\epsilon$-relative approximate Nash equilibrium is somewhat analogous to that of an $\epsilon$-Nash equilibrium. In particular, notice that if the entries of the payoff matrix are in $[0, 1]$, then for an $\epsilon$-relative approximate Nash equilibrium, $y$, (Resp $(y), y$) is an $\epsilon$-Nash equilibrium.

**Definition 26** *Let $(R, I)$ be a mimicking game. Define a strategy $y$ for the column player to be a strong $\epsilon$-relative Nash equilibrium for some $\epsilon > 0$ if for every strategy $z$ with Supp $(z) \subseteq$ Supp $(y)$, $z^T Ry(1+\epsilon) \geq x^T Ry$ for any strategy $x$.*

This says that the strategies in the support of $y$ are not worse for the row player to play than any other strategy by a factor of more than $\epsilon$.

Note that any strong $\epsilon$-relative approximate Nash equilibrium is an $\epsilon$-relative approximate Nash equilibrium.

Note also that to check for a strong $\epsilon$-relative approximate Nash equilibrium it is sufficient to check only the basis vectors $z$ for elements of Supp $(y)$.

Lastly, we note that if $y$ is a vector with non negative coefficients, not all 0, that do not necessarily add to 1, and otherwise satisfies the conditions in Definition 25 (resp. 26), then upon dividing $y$ by the sum of its coefficients we get a (strong) $\epsilon$-relative approximate Nash equilibrium.

We would like to be able to assume that the largest entry in each row of $R$ is 1. We are able to reduce to this case after making the observation that if we scale the rows of $R$ and the corresponding entries of $y$ correctly, this does not effect our problem. This is formalized in the following Lemma.

**Lemma 27** *If $A$ is a diagonal matrix with positive entries along the diagonal , then $y$ is a (strong) $\epsilon$-relative approximate Nash equilibrium of $(R, I)$ if and only if the strategy proportional to $A^{-1}y$ is a (strong) $\epsilon$-relative approximate Nash equilibrium of $(RA, I)$.*

**Proof:** We have that Supp $(A^{-1}y) =$ Supp $(y)$, therefore if $y$ is an $\epsilon$-relative approximate Nash equilibrium, Resp $(A^{-1}y)^T RAA^{-1}y(1 + \epsilon) =$ Resp $(y)^T Ry(1 + \epsilon) \geq x^T Ry = x^T RAA^{-1}y$, and hence the strategy proportional to $A^{-1}y$ is an $\epsilon$-relative approximate Nash equilibrium of $(I, RA)$. If $y$ is a strong $\epsilon$-relative approximate Nash equilibrium of $(R, I)$, then if $z$ is a strategy, and Supp $(z) \subseteq$ Supp $(y) =$ Supp $(A^{-1}y)$, then we have that $z^T RAA^{-1}y(1 + \epsilon) = z^T Ry(1 + \epsilon) \geq x^T Ry = x^T RAA^{-1}y$, and hence $A^{-1}y$ is a strong $\epsilon$-relative approximate Nash equilibrium of $(RA, I)$. ▌

**Definition 28** *Assign to a mimicking game $(R, I)$ the digraph $G_R$ on the pure strategies of the column player, by putting an edge from any pure strategy $e_i$ to another pure strategy $e_j$ if and only if ${e_j}^T R e_i \neq 0$.*

**Theorem 29** *Let $(R, I)$ be a mimicking game where $R$ has non-negative entries and is an $n \times n$ matrix. If $G_R$ has treewidth $k$ and vertices having maximum in-degree $m$, then there is an algorithm which given $(R, I)$, an elimination order of $G_R$, and an $0 < \epsilon < 1$ returns a strong $\epsilon$-relative approximate Nash equilibrium in $O(k \log(\frac{m}{\epsilon})(\frac{20m}{\epsilon})^{2k+2} n)$ time.*

First if any column of $R$ is all 0, then the vector $y$ that is 1 on this entry and zero elsewhere and Resp $(y)$ define a Nash equilibrium, and also a 0-relative approximate Nash equilibrium.

Otherwise, we can reduce to the case where the largest entry in each column of $R$ is equal to 1 using Lemma 27 to replace each column of $R$ by this column divided by the size of its largest entry (this corresponds to multiplying $R$ by a diagonal matrix on the right).

**Definition 30** *For a vector, $y$ and matrix $R$, define $\lfloor Ry \rfloor_j = \sum_i \lfloor R_{j,i} y_i \rfloor$ to be the vector obtained by multiplying $R$ by $y$ in the normal way, only taking the floor of the products of components.*

Our next definition is somewhat technical, but it is useful as we will see in the next two lemmas. We define a class of vectors given $\epsilon$ and a parameter $N$, that we will see for $N$ sufficiently large, both exist and are strong $\epsilon$-relative approximate Nash equilibria.

**Definition 31** *If $(R, I)$ is a mimicking game, $N$ is a positive integer, and $0 < \epsilon < 1$, then a non-zero vector $y$ is an $N$-discrete $\epsilon$-approximation of $(R, I)$ if $y$ has coefficients in $\{0, \frac{1}{2}, \frac{3}{2}, \ldots, \frac{2N-1}{2}\}$, $\lfloor Ry \rfloor$ has entries at least $(1 - \frac{\epsilon}{3})N$ on Supp $(y)$, and at most $(1 + \frac{\epsilon}{9})N$ everywhere.*

**Lemma 32** *If $y$ is an $N$-discrete $\epsilon$-approximation of a mimicking game $(R, I)$, where $R$ has non-negative entries, vertices in $G_R$ have in-degree at most $m$, and $N \geq \frac{9m}{2\epsilon}$, then $y$ when normalized to 1 is a strong $\epsilon$-relative approximate Nash equilibrium of $(R, I)$.*

**Proof:** Let $\mathbb{1}$ be the vector all of whose coefficients are 1. For any strategy $x$,

$$x^T Ry \leq x^T(\lfloor Ry \rfloor + m\mathbb{1}) \leq (1 + \frac{\epsilon}{9})N + m \leq (1 + \frac{\epsilon}{3})N$$

. On the other hand, for any strategy $z$ with Supp $(z) \subseteq$ Supp $(y)$, $z^T Ry \geq z^T \lfloor Ry \rfloor \geq (1 - \frac{\epsilon}{3})N$. Since $1 + \epsilon > \frac{1+\epsilon/3}{1-\epsilon/3}$ for $0 < \epsilon < 1$, these together imply that $z^T Ry(1 + \epsilon) > x^T Ry$, which proves our lemma. ▌

**Lemma 33** *If $(R, I)$ is a mimicking game where $R$ has non-negative entries, the largest entry in each column of $R$ is 1, and the maximum in-degree of a vertex in $G_R$ is $m$, then for any $0 < \epsilon < 1$ and any integer $N > \frac{9m}{2\epsilon}$ there exists an $N$-discrete $\epsilon$-approximation of $(R, I)$.*

**Proof:** Let (Resp $(y*), y*$) be a Nash equilibrium of the game $(R, I)$. Note that row player's payoff in this equilibrium must be positive, since for $y$ non-zero with non-negative coefficients, $Ry$ has non-negative coefficients at least one of which is non-zero, since all columns of $R$ are non-zero with non-negative coefficients. Since this is a Nash equilibrium, $Ry*$ has its maximum entries on Supp $(y*)$. Scale $y*$ to a non-zero vector $y$ with non-negative coefficients so that the entries of $Ry$ are $N$ on Supp $(y)$ and smaller elsewhere. Notice that since each column in $R$ has some entry equal to 1, all coefficients of $y$ are at most $N$. Let $z$ be the vector obtained by rounding the non-zero components in $y$ to the nearest half integer. Clearly the coefficients of $z$ are in $\{0, \frac{1}{2}, \dots, \frac{2N-1}{2}\}$.

Notice that the components of $\lfloor Rz \rfloor$ are $\sum_i \lfloor R_{j,i} z_i \rfloor = \sum_{(i,j)\in G_R} \lfloor R_{j,i} z_i \rfloor \geq \sum_{(i,j)\in G_R} R_{j,i} y_i - \frac{R_{j,i}}{2} - 1$ which is at least the corresponding coefficient of $Ry$ minus $\frac{3m}{2}$. Similarly, the coefficients of $\lfloor Rz \rfloor$ are at most the corresponding coefficients of $Ry$ plus $\frac{m}{2}$. Therefore, the coefficients of $\lfloor Rz \rfloor$ on Supp $(y)$ = Supp $(z)$ are at least $N - \frac{3m}{2} > (1 - \frac{\epsilon}{3})N$, and everywhere these coefficients are at most $N + \frac{m}{2} < (1 + \frac{\epsilon}{9})N$. Therefore, $z$ is an $N$-discrete $\epsilon$-approximation of $(R, I)$. ▌

Assign to the edge $(i, j)$ of $G_R$ the value $w(i, j) = R_{ij}$. Then, by definition, finding an $N$-discrete $\epsilon$-approximation to $(R, I)$ is equivalent to finding an assignment of the values, $v(i) \in \{0, \frac{1}{2}, \dots, \frac{2N-1}{2}\}$ to the vertices, $i \in G_R$ so that $\sum_{(i,j)\in G_R} \lfloor v(i)w(i, j) \rfloor \geq (1 - \frac{\epsilon}{3})N$ if $v(j) > 0$ and $\sum_{(i,j)\in G_R} \lfloor v(i)w(i, j) \rfloor \leq (1 + \frac{\epsilon}{9})N$ for all $j$.

We generalize this to the following problem: given a digraph $G = (V, E)$ of treewidth at most $k$ and an elimination order (here $G_R$), a finite set $S$ (here $\{0, \frac{1}{2}, \dots, \frac{2N-1}{2}\}$), a function $f$ mapping $S \times E \to \{0, 1, 2, \dots\}$ (here $(s, e) \to \lfloor sw(e) \rfloor$) , a set $P$ of cliques of $V$ (here the empty sets), a positive integer $M$ (here $\lfloor (1 + \frac{\epsilon}{9})N \rfloor$), for each $v \in V$ a subset of $S \times \{0, 1, \dots, M\}$, here

$$\{0\} \times \{0, 1, \dots, M\} \cup (S - \{0\}) \times \{\lceil (1 - \frac{\epsilon}{3})N \rceil, \dots, M\},$$

and for each clique in $P$, $V'$, a subset of $(S \times \{0, 1, \dots, M\})^{V'}$.

The problem is to find, if one exists, a function $v : V \to S$ and elements of the subsets of $(S \times \{0, 1, \dots, M\})^{V'}$ so that: for $a \in V$ where $a \in V_i \in P$ and the elements of $(S \times \{0, 1, \dots, M\})^{V_i}$ we picked maps $a$ to $(v(a), f_i)$ for some $f_i(a)$, and

$$(v(a), \sum_{(b,a)\in E} f(v(b), (b, a)) + \sum_i f_i(a))$$

is in the subset of $S \times \{0, 1, \dots, M\}$ associated with $a$.

We present an algorithm, that given an elimination order of $G$ solves this problem in $O(k \log M(2|S|(M + 1))^{k+1}(2|V| + |P|))$ time. It is clear that a solution to our specific case of this generalized problem produces an $N$-discrete $\epsilon$-approximation of $(R, I)$ if one exists.

First we need an algorithm that given any $l$ subsets of $\{0, 1, \dots, M\}^a$ finds the intersection of all possible sums of one element from each set with $\{0, 1, \dots, M\}^a$ in $O(a \log M(4M)^a l)$ time. Given any two such sets, $S_1$ and $S_2$, we can compute all of the sums of pairs of elements from them lying in $\{0, \dots, 2^{\lceil \log(2M) \rceil} - 1\}^a$ in $O(a \log M(4M)^a$ time using the fast Fourier transform to perform the convolution. We then take the intersection of this set with $\{0, 1, \dots, M\}^a$ to get $S_1 + S_2$ (the set of all sums in $\{0, \dots, M\}^a$). Hence if we are given such sets $S_1, S_2, \dots, S_l$, we compute all the correct sums of $S_1, S_2$, then take all the sums of these with $S_3$, and so on. Furthermore, if we keep all this information and are given a sum of values, we can work backwards to find values $s_i \in S_i$ so that $\sum_i s_i$ is equal to this sum in $O(l(M + 1)^a)$ time.

**Algorithm:**

If $|V| = 1$, then try all assignments of values in $S$ and values in $(S \times \{0, 1, \dots, M\})^{V'}$ to see if they work, and output one that does.

Otherwise, consider the first $v \in V$ in the elimination order. Let $V'$ be the set of vertices adjacent to $V$. Let $V_i$ be the elements of $P$ containing $v$. We wish to compute the subset $U$ of $(S \times \{0, \dots, M\})^{V'}$ corresponding to a map $w : V' \to S \times \{0, \dots, M\}$ so that there exists an assignment of elements of $S$ to $V' \cup \{v\}$ that agrees with the first component of $w$, and an element, $w_i : V_i \to S \times \{0, \dots, M\}$ from each of the subsets of $(S \times \{0, \dots, M\})^(V_i)$ whose first component also agrees with $w$ and

1. $(w(v), \sum_{(u,v)\in E} f(w(u),(u,v)) + \sum_i w_i(v)_2)$ is one of the elements of the subset of $S \times \{0,\dots,M\}$ associated with $v$.

2. for $u \in V'$, $w(u)_2$ is $\sum_{u\in V_i} w_i(u)_2$ plus an additional $f(w(v),(v,u))$ if $(v,u)\in E$.

We do this by scanning through all of the at most $|S|^{k+1}$ functions from $V'$ to $S$. For each of these consider the subsets of the $V_i$ that agree with this assignment. We treat these subsets as subsets of $\{0,\dots,M\}^{V'\cup\{v\}}$. We then compute all of the possible sums of these vectors that lie in $\{0,\dots,M\}^{V'\cup\{v\}}$ using the algorithm mentioned above. We then add these to the vector that assigns the values $\sum_{(u,v)\in E} f(w(u),(u,v))$ to $v$ and assigns $f(w(v),(v,u))$ to $u$ if $(v,u)\in E$ and 0 otherwise for $u \in V'$. For all of these values that are still in $\{0,\dots,M\}^{V'\cup\{v\}}$ and assign to $v$ values that along with $w(v)$ is in the subset of $S\times\{0,\dots,M\}$ associated with $v$, we add $w$ cross the projection of our element of $\{0,\dots,M\}^{V'\cup\{v\}}$ to $\{0,\dots,M\}^{V'}$ to our subset $U$.

Using the algorithm above, this list takes $O(|V'|\log M(4|S|M)^{|V'+1|}) = O(k\log M(4|S|M)^{k+1})$ time to produce for each $V_i$ and an additional copy of at most this long to record the values of $U$ in a table. Notice that set $U$ stores for each assignment of values in $S$ to elements of $V'$ the elements of $\{0,\dots,M\}^{V'}$ that can be obtained as contributions to $\sum_{(b,u)\in E} f(v(b),(b,u)) + \sum_i f_i(u)$ coming from edges of the form $(v,u)$ and elements of our subsets of $(S\times\{0,\dots,M\})^{V_i}$ so that our assignments at $v$ are consistent.

Therefore, we can reduce to solving our problem on $G$ with $v$ removed, edges added between all vertices in $V'$ (with $f(s,e)=0$ for any new edge $e$), all of the $V_i$ removed from $P$, and $V'$ added to $P$ with the associated set $U$. After solving our problem on this reduced graph (with fewer vertices) recursively, we are given assignments of values of $S$ to vertices of $G$ other than $v$, elements in sets associated with elements in $P$ not containing $v$, and an element of $U$ so that if we can find an assignment of a value in $S$ to $v$ and elements of the sets associated with the $V_i$ that are consistent with this element of $U$, we will have solved the problem. Working backwards, through our previous calculations, we can find an element of the subset of $S\times\{0,\dots,M\}$ associated with $v$, that led to this value in $U$. Then we just have to find elements of the appropriate subsets of $\{0,\dots,M\}^{V'\cup\{v\}}$ with the appropriate sum, which we can do.

The runtime of this algorithm is $O(k\log M(4|S|M))^{k+1})$ times the number of $V_i$ plus one to create the new table, plus the time to run the algorithm on the new graph. The new graph has $|P|$ one smaller for each $V_i$ and one larger due to its new element. The new graph has $|V|$ one smaller. Therefore, by induction on $|V|$, it is easy to show that this algorithm terminates in $O(k\log M(4|S|M)^{k+1}(2|V|+|P|))$ time.

For the actual case that we need to run, $|S| = N+1$,$M \le N(1+\frac{\epsilon}{9}) \le N\frac{10}{9}$,$N \le \frac{9m}{2\epsilon}$, $|V| = n$ and $|P| = 0$. Hence the total runtime is $O(k\log(\frac{m}{\epsilon})(\frac{20m}{\epsilon})^{2k+2}n)$.

## References

[Ben] C. H. Bennett, Time/space trade-offs for reversible computation, SIAM J. Comput. 18, 766 (1989)

[BGI] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang, ”On the (im)possibility of obfuscating programs,” in Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology (CRYPTO ’01), J. Kilian, Ed. Santa Barbara, California: Springer-Verlag, Aug. 19-23 2001, pp. 1–18. http://citeseer.ist.psu.edu/barak01impossibility.html

[CS1] V. Conitzer and T. Sandholm. Complexity results about nash equilibria. In Proc. of IJCAI, pages 765–771, 2003.

[CS2] Bruno Codenotti, Daniel Stefankovic. On the computational complexity of Nash equilibria for (0,1)-bimatrix games. Submitted.

[DGC] Diaconis, P., Graham, R., Chung, Fan., Combinatorics for the East model, Adv. Appl. Math. 27, 192-206.

[LMM] Richard J. Lipton , Evangelos Markakis , Aranyak Mehta, Playing large games using simple strategies, Proceedings of the 4th ACM conference on Electronic commerce, p.36-41, June 09-12, 2003, San Diego, CA, USA

[Pap1] C. H. Papadimitriou, Algorithms, Games, and the Internet. STOC 2001.

[Pap2] 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.

[Ste1] B. von Stengel (2002), Computing equilibria for two-person games. Chapter 45, Handbook of Game Theory, Vol. 3, eds. R. J. Aumann and S. Hart, North-Holland, Amsterdam, 1723-1759

[SS] R. Savani and B. von Stengel (2004), Exponentially Many Steps for Finding a Nash Equilibrium in a Bimatrix Game. In: Proc. 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2004), 258-267.