Game Theory and Computational Complexity

Paul W. Goldberg\textsuperscript{1}

\textsuperscript{1}Department of Computer Science
University of Liverpool, U. K.

EWSCS’09 Winter School, Estonia
March 2009
- Problem statements, Nash equilibrium
- \textbf{NP}-completeness of finding certain Nash equilibria\textsuperscript{1}

\textsuperscript{1}I will give you definitions soon!
Problem statements, Nash equilibrium

**NP**-completeness of finding certain Nash equilibria\(^1\)

Total search problems, **PPAD** and related complexity classes

**PPAD**-completeness of finding unrestricted Nash equilibria

[Daskalakis, G and Papadimitriou, CACM Feb’09]

---

\(^1\)I will give you definitions soon!
Topics

- Problem statements, Nash equilibrium
- **NP**-completeness of finding certain Nash equilibria\(^1\)
- Total search problems, **PPAD** and related complexity classes
- **PPAD**-completeness of finding unrestricted Nash equilibria
  [Daskalakis, G and Papadimitriou, CACM Feb’09]
- The quest for fast algorithms to compute *approximate* Nash equilibria

and I really ought to mention “Potential games” and the complexity class **PLS**, but won’t have time to do the details.

\(^1\)I will give you definitions soon!
"The first step toward solving New York City's problems is to state them, which I will now proceed to do."
Modern CS and GT originated with John von Neumann at Princeton in the 1950s [Yoav Shoham, CACM Aug’08]
Modern CS and GT originated with John von Neumann at Princeton in the 1950s [Yoav Shoham, CACM Aug’08]

Common motivations:
- modeling *rationality* (interaction of selfish agents on Internet);
- AI: solve cognitive tasks such as negotiation
Modern CS and GT originated with John von Neumann at Princeton in the 1950s [Yoav Shoham, CACM Aug’08]

Common motivations:
- modeling *rationality* (interaction of selfish agents on Internet);
- AI: solve cognitive tasks such as negotiation

It turns out that GT gives rise to problems that pose very interesting mathematical challenges, e.g. w.r.t. computational complexity. Complexity classes **PPAD** and **PLS**.
Example 1: Prisoners’ dilemma

There’s a row player and a column player.

Nash equilibrium: no incentive to change
Example 1: Prisoners’ dilemma

There’s a row player and a column player.

Solution: both players defect. Numbers in red are probabilities.
Nash equilibrium: no incentive to change

Goldberg | Game Theory and Computational Complexity
Example 1: Prisoners’ dilemma, e.g. Oil cartel

There’s a row player and a column player.

Solution: both players defect. Numbers in red are probabilities.

Nash equilibrium: no incentive to change
Example 2: Rock-paper-scissors

2008 Rock-paper-scissors Championship (Las Vegas, USA)
### Rock-paper-scissors: payoff matrix

<table>
<thead>
<tr>
<th></th>
<th>Rock (0)</th>
<th>Paper (1)</th>
<th>Scissors (-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rock</td>
<td>0</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>Paper</td>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Scissors</td>
<td>1</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Solution:**
Both players randomize: probabilities are shown in red.
Rock-paper-scissors: payoff matrix

Solution: both players randomize: probabilities are shown in red.
Rock-paper-scissors: a non-symmetrical variant

<table>
<thead>
<tr>
<th></th>
<th>rock</th>
<th>paper</th>
<th>scissors</th>
</tr>
</thead>
<tbody>
<tr>
<td>rock</td>
<td>0</td>
<td>-1</td>
<td>2</td>
</tr>
<tr>
<td>paper</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>scissors</td>
<td>1</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

What is the solution?

(tanks to Rahul Savani's on-line Nash equilibrium solver.)
Rock-paper-scissors: a non-symmetrical variant

\[
\begin{array}{c|c|c}
\text{rock} & \text{paper} & \text{scissors} \\
1/3 & 5/12 & 1/4 \\
\end{array}
\]

What is the solution?
(thanks to Rahul Savani’s on-line Nash equilibrium solver.)
Example 3: Stag hunt

2 hunters; each chooses whether to hunt stag or rabbit...
Example 3: Stag hunt

2 hunters; each chooses whether to hunt stag or rabbit...
It takes 2 hunters to catch a stag,
Example 3: Stag hunt

2 hunters; each chooses whether to hunt stag or rabbit...
It takes 2 hunters to catch a stag, but only one to catch a rabbit.
### Stag hunt: payoff matrix

<table>
<thead>
<tr>
<th></th>
<th>Hunt stag</th>
<th>Hunt rabbit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hunt stag</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Solution:** both hunt stag (the *best* solution).
Solution: both hunt stag (the best solution). Or, both players hunt rabbit.
Solution: both hunt stag (the best solution). Or, both players hunt rabbit. Or, both players randomize (with the right probabilities).
**Game**: set of players, each player has his own set of allowed actions (also known as “pure strategies”). Any combination of actions will result in a numerical payoff (or value, or utility) for each player. (A game should specify the payoffs, for every player and every combination of actions.)
**Game:** set of players, each player has his own set of allowed actions (also known as “pure strategies”). Any combination of actions will result in a numerical payoff (or value, or utility) for each player. (A game should specify the payoffs, for every player and every combination of actions.)

Number the players $1, 2, ..., k$. 
**Game**: set of players, each player has his own set of allowed actions (also known as “pure strategies”). Any combination of actions will result in a numerical payoff (or value, or utility) for each player. (A game should specify the payoffs, for every player and every combination of actions.)

Number the players 1, 2, ..., k.

Let $S_p$ denote player $p$’s set of actions. e.g. in rock-paper-scissors, $S_1 = S_2 = \{\text{rock, paper, scissors}\}$. 

**Game**: set of players, each player has his own set of allowed actions (also known as “pure strategies”). Any combination of actions will result in a numerical payoff (or value, or utility) for each player. (A game should specify the payoffs, for every player and every combination of actions.)

Number the players 1, 2, ..., k.

Let $S_p$ denote player p’s set of actions. e.g. in rock-paper-scissors, $S_1 = S_2 = \{\text{rock, paper, scissors}\}$.

$n$ denotes the size of the largest $S_p$. (So, in rock-paper-scissors, $k = 2$, $n = 3$.) If $k$ is a constant, we seek algorithms polynomial in $n$. Indeed, much work studies special case $k = 2$, where a game’s payoffs can be written down in 2 matrices.

$S = S_1 \times S_2 \times \ldots \times S_k$ is the set of **pure strategy profiles**. i.e. if $s \in S$, $s$ denotes a choice of action, for each player.
**Game**: set of players, each player has his own set of allowed actions (also known as “pure strategies”). Any combination of actions will result in a numerical payoff (or value, or utility) for each player. (A game should specify the payoffs, for every player and every combination of actions.)

Number the players $1, 2, \ldots, k$.

Let $S_p$ denote player $p$’s set of actions. e.g. in rock-paper-scissors, $S_1 = S_2 = \{\text{rock, paper, scissors}\}$.

$n$ denotes the size of the largest $S_p$. (So, in rock-paper-scissors, $k = 2$, $n = 3$.) If $k$ is a constant, we seek algorithms polynomial in $n$. Indeed, much work studies special case $k = 2$, where a game’s payoffs can be written down in 2 matrices.

$S = S_1 \times S_2 \times \ldots \times S_k$ is the set of pure strategy profiles. i.e. if $s \in S$, $s$ denotes a choice of action, for each player.

Each $s \in S$ gives rise to utility or payoff to each player. $u_s^p$ will denote the payoff to player $p$ when all players choose $s$. 

---

Goldberg Game Theory and Computational Complexity
Two parameters, $k$ and $n$.

**normal-form game:** list of all $u^p_s$'s

- 2-player: $2 \times n \times n$ matrices; so $2n^2$ numbers
- $k$-player: $kn^k$ numbers

Thus, only good for games with *few players*, but OK to allow large $n$. (fix $k$; size of game is $\text{poly}(n)$.)
Two parameters, $k$ and $n$.

**normal-form game:** list of all $u^p_s$'s

- 2-player: $2 \times n$ matrices; so $2n^2$ numbers
- $k$-player: $kn^k$ numbers

Thus, only good for games with few players, but OK to allow large $n$. (fix $k$; size of game is poly($n$).)

**General issue:**

run-time of algorithms (to solve a game) in terms of $n$

$k$ is small constant; often we consider the case of $k = 2$.

**When can it be polynomial in $n$?**
This basic model may not always capture details of a situation. In a Bayesian game, $u_s^p$ could be probability distribution over $p$'s payoff, allowing one to represent uncertainty about a payoff.
Some comments

This basic model may not always capture details of a situation. In a Bayesian game, $u^p_s$ could be probability distribution over $p$’s payoff, allowing one to represent uncertainty about a payoff. This is not really intended combinatorial games like chess, where players take turns. One could define a strategy in advance, but it would be impossibly large to represent...
Nash equilibrium

- standard notion of “outcome of the game”
Nash equilibrium

- standard notion of “outcome of the game”
- it should specify a strategy for each player, such that each player is receiving optimal payoff in the context of the other players’ choices.
Nash equilibrium

- standard notion of “outcome of the game”
- it should specify a strategy for each player, such that each player is receiving optimal payoff in the context of the other players’ choices.
- A pure Nash equilibrium is one in which each player chooses a pure strategy — problem: for some games, there is no pure Nash equilibrium!

John Forbes Nash
Nash equilibrium

- standard notion of “outcome of the game”
- it should specify a strategy for each player, such that each player is receiving optimal payoff in the context of the other players’ choices.
- A pure Nash equilibrium is one in which each player chooses a pure strategy — problem: for some games, there is no pure Nash equilibrium!
- A mixed Nash equilibrium assigns, for each player, a probability distribution over his pure strategies, so that a player’s payoff is his expected payoff w.r.t. these distributions — Nash’s theorem shows that this always exists!
Computational problem

**Pure Nash**

**Input:** A game in normal form, essentially consisting of all the values $u_s^p$ for each player $p$ and strategy profile $s$.

**Question:** Is there a pure Nash equilibrium.

Goldberg  Game Theory and Computational Complexity
Pure Nash

Input: A game in normal form, essentially consisting of all the values $u^p_s$ for each player $p$ and strategy profile $s$.

Question: Is there a pure Nash equilibrium.

That decision problem has corresponding search problem that replaces the question with

Output: A pure Nash equilibrium.

If the number of players $k$ is a constant, the above problems are in $P$. If $k$ is not a constant, you should really study “concise representations” of games.
Another computational problem

**Nash**

**Input:** A game in normal form, essentially consisting of all the values $u^p_s$ for each player $p$ and strategy profile $s$.

**Output:** A (mixed) Nash equilibrium.

By Nash’s theorem, intrinsically a search problem, not a decision problem.
Another computational problem

\textbf{Nash}

\begin{itemize}
  \item \textbf{Input:} A game in normal form, essentially consisting of all the values $u^p_s$ for each player $p$ and strategy profile $s$.
  \item \textbf{Output:} A (mixed) Nash equilibrium.
\end{itemize}

By Nash’s theorem, intrinsically a search problem, not a decision problem.

3+ players: big problem: solution may involve irrational numbers.

So we will change the problem a bit:

\textbf{Useful Analogy}

(totol) search for root of (odd-degree) polynomial: look for approximation

Replace “no incentive to change” by “low incentive”
Re-state the problem

**Epsilon-Nash equilibrium:** Expected payoff + $\epsilon \geq$ exp’d payoff of best possible response

**Approximate Nash**

**Input:** A game in normal form, essentially consisting of all the values $u^p_s$ for each player $p$ and strategy profile $s$. $u^p_s \in [0, 1]$. small $\epsilon > 0$

**Output:** A (mixed) $\epsilon$-Nash equilibrium.

Notice that we restrict payoffs to $[0, 1]$ (why?) Formulate computational problem as: Algorithm to be polynomial in $n$ and $1/\epsilon$.

If the above is hard, then it’s hard to find a true Nash equilibrium.
A key feature of these lectures is the distinction between search problems and decision problems. We still have decision problems like: *Does there exist a mixed Nash equilibrium with total payoff \( \geq \frac{2}{3} \)?
$\mathcal{I}(X)$ denotes instances of problem $X$

For decision problems, where $x \in \mathcal{I}(X)$ has $\text{output}(x) \in \{\text{yes, no}\}$, to reduce $X$ to $X'$,

poly-time computable function $f: \mathcal{I}(X) \rightarrow \mathcal{I}(X')$

\[
\text{output}(f(x)) = \text{output}(x)
\]
Polynomial-time reductions

\(\mathcal{I}(X)\) denotes instances of problem \(X\)
For decision problems, where \(x \in \mathcal{I}(X)\) has \(\text{output}(x) \in \{\text{yes, no}\}\),
to reduce \(X\) to \(X'\),
poly-time computable function \(f: \mathcal{I}(X) \rightarrow \mathcal{I}(X')\)

\[\text{output}(f(x)) = \text{output}(x)\]

Search problems:
Given \(x \in \mathcal{I}(X)\), \(y = \text{output}(x)\) is a poly-length string.
Poly-time computable functions

\[f: \mathcal{I}(X) \rightarrow \mathcal{I}(X')\quad \text{and} \quad g: \text{solutions}(X') \rightarrow \text{solutions}(X)\]

If \(y = f(x)\) then \(g(\text{output}(y)) = \text{output}(x)\).
This achieves aim of showing that if \(X' \in \text{P}\) then \(X \in \text{P}\); equivalently if \(X \not\in \text{P}\) then \(X' \not\in \text{P}\).
All **NP** decision problems have corresponding **NP** search problems where \( y \) is certificate of “\( \text{output}(x) = \text{yes} \)” e.g. given boolean formula \( \Phi \), is it satisfiable? \( y \) is satisfying assignment (which is hard to find but easy to check)

**Total** search problems (e.g. **NASH** and others) are more tractable in the sense that for all problem instances \( x \), \( \text{output}(x) = \text{yes} \). So, every instance has a solution, and a certificate.
2-player game: specified by two $n \times n$ matrices; look for solutions that can be found in time polynomial in $n$. 

Gilboa and Zemel '89; Conitzer and Sandholm '03: it is \textit{NP}-hard to find (for 2-player games) the NE with highest social welfare. CS paper gives a class of games for which various restricted NE are hard to find, e.g. NE that guarantees player 1 a payoff of $\alpha$. The following is a brief sketch of their construction (note: after this, I will give 2 simpler reductions in detail).
2-player game: specified by two $n \times n$ matrices; look for solutions that can be found in time polynomial in $n$.

Gilboa and Zemel ’89; Conitzer and Sandholm ’03: it is $\text{NP}$-hard to find (for 2-player games) the NE with highest social welfare. CS paper gives a class of games for which various restricted NE are hard to find, e.g. NE that guarantees player 1 a payoff of $\alpha$. 
2-player game: specified by two $n \times n$ matrices; look for solutions that can be found in time polynomial in $n$.

Gilboa and Zemel '89; Conitzer and Sandholm '03: it is $\text{NP}$-hard to find (for 2-player games) the NE with highest social welfare. CS paper gives a class of games for which various restricted NE are hard to find, e.g. NE that guarantees player 1 a payoff of $\alpha$.

The following is a brief sketch of their construction (note: after this, I will give 2 simpler reductions in detail)
Reduce from *Satisfiability*: Given a CNF formula $\Phi$ with $n$ variables and $m$ clauses, find a satisfying assignment
Construct game $G_\Phi$ having $3n + m + 1$ actions per player (hence of size polynomial in $\Phi$)
NP-Completeness of finding “good” Nash equilibria

\[
x_1 \cdots x_n + x_1 \cdots + x_n C_1 \cdots - x_n C_1 \cdots C_m\]

\[
x_1 \cdots x_n + x_1 \cdots + x_n C_1 \cdots - x_n C_1 \cdots C_m\]

\[
\begin{array}{c|c|c|c|c|c|c}
\hline
& & & & & & f \\
\hline
x_1 & \ddots & & & & & 1 \\
x_n & & \ddots & & & & 0 \\
+x_1 & & & \ddots & & & 0 \\
\vdots & \ddots & \ddots & & & & \vdots \\
+x_n & & & & \ddots & & 1 \\
-x_1 & & & & & \ddots & 0 \\
\vdots & \ddots & \ddots & \ddots & & \ddots \\
-x_n & & & & & & 0 \\
C_1 & & & & & & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\
C_m & & & & & & 0 \\
\hline
f & 0 & 0 & \cdots & \cdots & \cdots & 0 \\
1 & 1 & \cdots & \cdots & \cdots & 1 \\
\varepsilon & 0 & 0 & \cdots & \cdots & \cdots & \varepsilon \\
\hline
\end{array}
\]
\[(f, f)\] is a Nash equilibrium.
NP-Completeness of finding “good” Nash equilibria

\[(f, f)\] is a Nash equilibrium.

Various other payoffs between 0 and \(n\) apply when neither player plays \(f\). They are chosen such that

- if \(\Phi\) is satisfiable, so also is a uniform distribution over a satisfying set of literals.
- No other Nash equilibria!
Comment: This shows it is hard to find “best” NE, but clearly $(f, f)$ is always easy to find.
Comment: This shows it is hard to find "best" NE, but clearly $(f, f)$ is always easy to find.

Should we expect it to be NP-hard to find unrestricted NE?
Comment: This shows it is hard to find “best” NE, but clearly $(f, f)$ is always easy to find.

Should we expect it to be $\textbf{NP}$-hard to find *unrestricted* NE?

General agenda of next lectures is to explain why we believe this is still hard, but not $\textbf{NP}$-hard.
In a zero-sum game, the total payoff of all the players is equal to 0. So, for 2-player games, the better the result for me, the worse it is for you. (example: rock-paper-scissors)

(Some background: General 2-player games are believed to be hard to solve, but zero-sum 2-player games are in P.)
Reduction between 2 versions of search for unrestricted NE: A simple example

In a zero-sum game, the total payoff of all the players is equal to 0. So, for 2-player games, the better the result for me, the worse it is for you. (example: rock-paper-scissors)

(Some background: General 2-player games are believed to be hard to solve, but zero-sum 2-player games are in \( \mathbf{P} \).)

Simple theorem

3-player zero-sum games are at least as hard as 2-player games.
In a zero-sum game, the total payoff of all the players is equal to 0. So, for 2-player games, the better the result for me, the worse it is for you. (example: rock-paper-scissors) (Some background: General 2-player games are believed to be hard to solve, but zero-sum 2-player games are in \( \text{P} \)).

Simple theorem

3-player zero-sum games are at least as hard as 2-player games. To see this, take any \( n \times n \) 2-player game \( \mathcal{G} \). Now add player 3 to \( \mathcal{G} \), who is “passive” — he has just one action, which does not affect players 1 and 2, and player 3’s payoff is the negation of the total payoffs of players 1 and 2.
Reduction between 2 versions of search for unrestricted NE: A simple example

In a zero-sum game, the total payoff of all the players is equal to 0. So, for 2-player games, the better the result for me, the worse it is for you. (example: rock-paper-scissors)
(Some background: General 2-player games are believed to be hard to solve, but zero-sum 2-player games are in \textbf{P}.)

Simple theorem

3-player zero-sum games are at least as hard as 2-player games.

To see this, take any \( n \times n \) 2-player game \( G \).
Now add player 3 to \( G \), who is “passive” — he has just one action, which does not affect players 1 and 2, and player 3’s payoff is the negation of the total payoffs of players 1 and 2. So, players 1 and 2 behave as they did before, and player 3 just has the effect of making the game zero-sum. Any Nash equilibrium of this 3-player game is, for players 1 and 2, a NE of the original 2-player game.
A symmetric game is one where “all players are the same”: they all have the same set of actions, payoffs do not depend on a player’s identity, only on actions chosen. For 2-player games, this means the matrix diagrams (of the kind we use here) should be symmetric (as in fact they are in the examples we saw earlier).

A slightly more interesting theorem

**symmetric 2-player games are as hard as general 2-player games.**
Given a $n \times n$ game $G$, construct a symmetric $2n \times 2n$ game $G' = f(G)$, such that given any Nash equilibrium of $G'$ we can efficiently reconstruct a NE of $G$. 

First step: if any payoffs in $G$ are negative, add a constant to all payoffs to make them all positive. 

Example:

\[
\begin{array}{cccccccc}
4 & -1 & 0 & 1 & 2 & 3 & -2 & 5 \\
7 & 2 & 3 & 4 & 5 & 6 & 1 & 8 \\
\end{array}
\]

Nash equilibria are unchanged by this (game is "strategically equivalent")
Given a $n \times n$ game $\mathcal{G}$, construct a symmetric $2n \times 2n$ game $\mathcal{G}' = f(\mathcal{G})$, such that given any Nash equilibrium of $\mathcal{G}'$ we can efficiently reconstruct a NE of $\mathcal{G}$.

First step: if any payoffs in $\mathcal{G}$ are negative, add a constant to all payoffs to make them all positive.

Example:

\[ \begin{array}{cc}
4 & 0 \\
2 & -2 \\
\end{array} \rightarrow \begin{array}{cc}
7 & 3 \\
5 & 1 \\
\end{array} \]

Nash equilibria are unchanged by this (game is “strategically equivalent”)

Goldberg | Game Theory and Computational Complexity
Reduction: 2-player to symmetric 2-player

So now let's assume $G$'s payoffs are all positive. Next stage:

\[ G' = \begin{pmatrix} 0 & G \\ G^T & 0 \end{pmatrix} \]

Example:
Reduction: 2-player to symmetric 2-player

Now suppose we solve the $2n \times 2n$ game $G' = \begin{pmatrix} 0 & G \\ G^T & 0 \end{pmatrix}$

Let $p$ and $q$ denote the probabilities that players 1 and 2 use their first $n$ actions, in some given solution.

\[
\begin{pmatrix} p & 1-q \\ 1-p & q \end{pmatrix} \begin{pmatrix} 0 & G \\ G^T & 0 \end{pmatrix}
\]

If $p = q = 1$, both players receive payoff 0, and both have incentive to change their behavior, by assumption that $G$’s payoffs are all positive. (and similarly if $p = q = 0$).

So we have $p > 0$ and $1 - q > 0$, or alternatively, $1 - p > 0$ and $q > 0$.

Assume $p > 0$ and $1 - q > 0$ (the analysis for the other case is similar).
Let \( \{p_1, \ldots, p_n\} \) be the probabilities used by player 1 for his first \( n \) actions, \( \{q_1, \ldots, q_n\} \) the probs for player 2’s second \( n \) actions.

\[
\begin{pmatrix}
  (p_1, \ldots, p_n) \\
  1 - p
\end{pmatrix}
\begin{pmatrix}
  q & (q_1 \ldots q_n) \\
  (0 & G) & (G^T & 0)
\end{pmatrix}
\]

Note that \( p_1 + \ldots + p_n = p \) and \( q_1 + \ldots + q_n = 1 - q \).
Let \( \{p_1, \ldots, p_n\} \) be the probabilities used by player 1 for his first \( n \) actions, \( \{q_1, \ldots q_n\} \) the probs for player 2’s second \( n \) actions.

\[
(p_1, \ldots p_n) \quad \begin{pmatrix} q & (q_1 \ldots q_n) \\ 1 - p & \begin{pmatrix} 0 & G \\ G^T & 0 \end{pmatrix} \end{pmatrix}
\]

Note that \( p_1 + \ldots + p_n = p \) and \( q_1 + \ldots + q_n = 1 - q \).

Then \( (p_1/p, \ldots, p_n/p) \) and \( (q_1/(1 - q), \ldots, q_n/(1 - q)) \) are a Nash equilibrium of \( G \)!

To see this, consider the diagram; they form a best response to each other for the top-right part.
**NP** decision problems: answer yes/no to questions that belong to some class. e.g. **SATISFIABILITY**: questions of the form Is boolean formula \( \Phi \) satisfiable?
**NP Search Problems**

**NP** decision problems: answer yes/no to questions that belong to some class. e.g. **Satisfiability**: questions of the form *Is boolean formula* $\Phi$ *satisfiable?*

Given the question *Is formula* $\Phi$ *satisfiable?* there is a fundamental asymmetry between answering **yes** and **no**.
**NP** decision problems: answer yes/no to questions that belong to some class. e.g. **SATISFIABILITY**: questions of the form Is boolean formula $\Phi$ satisfiable?

Given the question Is formula $\Phi$ satisfiable? there is a fundamental asymmetry between answering yes and no. If yes, there exists a small “certificate” that the answer is yes, namely a satisfying assignment. A certificate consists of information that allows us to check (in poly time) that the answer is yes.
**NP Search Problems**

NP decision problems: answer yes/no to questions that belong to some class. e.g. **Satisfiability**: questions of the form *Is boolean formula $\Phi$ satisfiable?*

Given the question *Is formula $\Phi$ satisfiable?* there is a fundamental asymmetry between answering *yes* and *no.*

If *yes*, there exists a small “certificate” that the answer is yes, namely a satisfying assignment. A certificate consists of information that allows us to check (in poly time) that the answer is yes.

A NP decision problem has a corresponding *search problem*: e.g. given $\Phi$, find $x$ such that $\Phi(x) = true$ (or say “no” if $\Phi$ is not satisfiable.)
Example of *Total* search problem in **NP**

### FACTORING

<table>
<thead>
<tr>
<th><strong>Input</strong></th>
<th>number ( N )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output</strong></td>
<td>prime factorisation of ( N )</td>
</tr>
</tbody>
</table>

\(2^{\text{polynomial in the number of digits in } N}\)
Example of *Total* search problem in \textbf{NP}

<table>
<thead>
<tr>
<th>FACTORING</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
</tr>
<tr>
<td><strong>Output</strong></td>
</tr>
</tbody>
</table>

\[ \text{e.g. Input 50 should result in output } 2 \times 5 \times 5. \]

\[ 2 \text{ polynomial in the number of digits in } N \]
Example of *Total* search problem in **NP**

**FACTORIZING**

<table>
<thead>
<tr>
<th><strong>Input</strong></th>
<th>number $N$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output</strong></td>
<td>prime factorisation of $N$</td>
</tr>
</tbody>
</table>

e.g. Input 50 should result in output $2 \times 5 \times 5$.

Given output $N = N_1 \times N_2 \times \ldots N_p$, it can be checked in polynomial time\(^2\) that the numbers $N_1, \ldots, N_p$ are prime, and their product is $N$.

\(^2\)polynomial in the number of digits in $N$
Example of *Total* search problem in **NP**

**Factoring**

<table>
<thead>
<tr>
<th>Input</th>
<th>number $N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>prime factorisation of $N$</td>
</tr>
</tbody>
</table>

e.g. Input 50 should result in output $2 \times 5 \times 5$.
Given output $N = N_1 \times N_2 \times \ldots N_p$, it can be checked in polynomial time\(^2\) that the numbers $N_1, \ldots, N_p$ are prime, and their product is $N$.
Hence, **Factoring** is in **FNP**. But, it’s a total search problem — every number has a prime factorization.

\(^2\)polynomial in the number of digits in $N$
Example of *Total* search problem in **NP**

**FACTORING**

**Input**  number $N$

**Output**  prime factorisation of $N$

e.g. Input 50 should result in output $2 \times 5 \times 5$.

Given output $N = N_1 \times N_2 \times \ldots N_p$, it can be checked in polynomial time\(^2\) that the numbers $N_1, \ldots, N_p$ are prime, and their product is $N$.

Hence, FACTORING is in **FNP**. But, it’s a *total* search problem — every number has a prime factorization.

It also seems to be hard! Cryptographic protocols use the belief that it is intrinsically hard. But probably not **NP**-complete

\(^2\)polynomial in the number of digits in $N$
Another **NP** total search problem

**EQUAL-SUBSETS**

<table>
<thead>
<tr>
<th><strong>Input</strong></th>
<th>positive integers $a_1, \ldots, a_n$; $\sum_i a_i \leq 2^n - 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output</strong></td>
<td>Two distinct subsets of these numbers that add up to the same total</td>
</tr>
</tbody>
</table>
Another **NP** total search problem

### **EQUAL-SUBSETS**

<table>
<thead>
<tr>
<th><strong>Input</strong></th>
<th>positive integers (a_1, \ldots, a_n; \sum_i a_i \leq 2^n - 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output</strong></td>
<td>Two distinct subsets of these numbers that add up to the same total</td>
</tr>
</tbody>
</table>

**Example:**

\[
42, 5, 90, 98, 99, 100, 64, 70, 78, 51
\]
Another **NP** total search problem

### EQUAL-SUBSETS

**Input**
positive integers $a_1, \ldots, a_n$; $\sum_i a_i \leq 2^n - 1$

**Output**
Two distinct subsets of these numbers that add up to the same total

**Example:**

42, 5, 90, 98, 99, 100, 64, 70, 78, 51

Solutions include $42 + 78 + 100 = 51 + 70 + 99$ and $42 + 5 + 51 = 98$. 
Another \textbf{NP} total search problem

\begin{center}
\textbf{EQUAL-SUBSETS}
\end{center}

\begin{tabular}{ | l | l | }
\hline
\textbf{Input} & positive integers $a_1, \ldots, a_n$; $\sum_i a_i \leq 2^n - 1$ \\
\textbf{Output} & Two distinct subsets of these numbers that add up to the same total \\
\hline
\end{tabular}

\textbf{Example:}

42, 5, 90, 98, 99, 100, 64, 70, 78, 51

Solutions include $42 + 78 + 100 = 51 + 70 + 99$ and $42 + 5 + 51 = 98$.

\textbf{EQUAL-SUBSETS} $\in \textbf{NP}$ (usual “guess and test” approach). But it is not known how to find solutions in polynomial time. The problem looks a bit like the NP-complete problem \textbf{SUBSET SUM}.
So, should we expect EQUAL SUBSETS to be \( \text{NP} \)-hard? No we should not. [Megiddo (1988)].

If any total search problem (e.g., EQUAL SUBSETS) is \( \text{NP} \)-complete, then it follows that \( \text{NP} = \text{co-NP} \), which is generally believed not to be the case.

To see why, suppose it is \( \text{NP} \)-complete, that \( \text{Sat} \leq_p \text{Equal subsets} \). Then there is an algorithm \( A \) for \( \text{Sat} \) that runs in polynomial time, provided that it has access to poly-time algorithm \( A' \) for \( \text{Equal subsets} \).

Now suppose \( A \) is given a non-satisfiable formula \( \Phi \). Presumably it calls \( A' \) some number of times, and receives a sequence of solutions to various instances of \( \text{Equal subsets} \), and eventually the algorithm returns the answer "no, \( \Phi \) is not satisfiable."
No we should not. [Megiddo (1988)].

If any total search problem (e.g. EQUAL SUBSETS) is NP-complete, then it follows that NP=co-NP, which is generally believed not to be the case.
So, should we expect \textsc{Equal Subsets} to be \textsc{NP}-hard?

No we should not. [Megiddo (1988)].

If any total search problem (e.g. \textsc{Equal Subsets}) is \textsc{NP}-complete, then it follows that \textsc{NP}=\textsc{co-NP}, which is generally believed not to be the case.

To see why, suppose it is \textsc{NP}-complete, that \textsc{Sat} \leq \textsc{p Equal Subsets}.

Then there is an algorithm \mathcal{A} for \textsc{Sat} that runs in polynomial time, provided that it has access to poly-time algorithm \mathcal{A}' for \textsc{Equal Subsets}.

Now suppose \mathcal{A} is given a \textit{non-satisfiable} formula \Phi. Presumably it calls \mathcal{A}' some number of times, and receives a sequence of solutions to various instances of \textsc{Equal Subsets}, and eventually the algorithm returns the answer “no, \Phi is not satisfiable”.
Now suppose that we replace $A'$ with the natural “guess and test” non-deterministic algorithm for \textsc{Equal-subsets}. We get a non-deterministic polynomial-time algorithm for \textsc{Sat}. Notice that when $\Phi$ is given to this new algorithm, the “guess and test” subroutine for \textsc{Equal-subsets} can produce the same sequence of solutions to the instances it receives, and as a result, the entire algorithm can recognize this non-satisfiable formula $\Phi$ as before. Thus we have \textbf{NP} algorithm that recognizes unsatisfiable formulae, which gives the consequence $\textbf{NP}=\textbf{co-NP}$. 
**TFNP**: **total** function problems in **NP**. We are interested in understanding the difficulty of **TFNP** problems. **Nash** and **Equal-subsets** do not seem to belong to **P** but are probably not **NP**-complete, due to being total search problems. Papadimitriou (1994) introduced a number of classes of total search problems.
**TFNP**: total function problems in **NP**. We are interested in understanding the difficulty of **TFNP** problems. **Nash** and **Equal-subsets** do not seem to belong to **P** but are probably not **NP-complete**, due to being total search problems. Papadimitriou (1994) introduced a number of classes of total search problems.

**General observation:**
Proofs that various search problems are total use a non-constructive step that is hard to compute.

**PPP** stands for “polynomial pigeonhole principle”; used to prove that **Equal-subsets** is a total search problem. “A function whose domain is larger than its range has 2 inputs with the same output”
A very general problem in **TFNP**

**Definition:**

_Pigeonhole circuit_ is the following search problem:

- **Input:** boolean circuit $C$, $n$ inputs, $n$ outputs
- **Output:** A boolean vector $x$ such that $C(x) = 0$, or alternatively, vectors $x$ and $x'$ such that $C(x) = C(x')$.

The “most general” computational total search problem for which the pigeonhole principle guarantees an efficiently checkable solution.
Various equivalent definitions of \textbf{Pigeonhole circuit}

With regard to questions of polynomial time computation, the following are equivalent

- $n$ inputs/outputs; $C$ of size $n^2$
- Let $p$ be a polynomial; $n$ inputs/outputs, $C$ of size $p(n)$
- $n$ is number of gates in $C$, number of inputs $=$ number of outputs.

Proof of equivalences: If version $i$ is in $\text{P}$ then version $j$ is in $\text{P}$. 
The complexity class **PPP**

**Definition**

A problem $X$ belongs to **PPP** if $X$ reduces to PIGEONHOLE CIRCUIT (in poly time).

Problem $X$ is **PPP**-complete is in addition, PIGEONHOLE CIRCUIT reduces to $X$. 

Analogy

Thus, **PPP** is to PIGEONHOLE CIRCUIT as **NP** is to satisfiability (or circuit sat, or any other **NP**-complete problem).

Pigeonhole circuit seems to be hard (it looks like Circuit sat) but (recall) probably not **NP**-hard.
The complexity class \textbf{PPP}

\section*{Definition}
A problem $X$ belongs to \textbf{PPP} if $X$ reduces to \textbf{Pigeonhole Circuit} (in poly time).
Problem $X$ is \textbf{PPP}-complete is in addition, \textbf{Pigeonhole circuit} reduces to $X$.

\section*{Analogy}
Thus, \textbf{PPP} is to \textbf{Pigeonhole circuit} as \textbf{NP} is to \textbf{satisfiability} (or \textbf{circuit sat}, or any other \textbf{NP}-complete problem).

\textbf{Pigeonhole circuit} seems to be hard (it looks like \textbf{Circuit sat}) but (recall) probably not \textbf{NP}-hard.
**What we know about** \textbf{EQUAL-SUBSETS}

**EQUAL-SUBSETS** belongs to \textbf{PPP}...

\[
y = 1 + \sum_{i} a_{i} x_{i}
\]

\[
y_{1} \quad y_{2} \quad \cdots \quad y_{n}
\]
What we know about **EQUAL-SUBSETS**

**EQUAL-SUBSETS** belongs to **PPP**...
but it is not known whether it is complete for **PPP**. (this is unsatisfying.)

\[
y = 1 + \sum_{i} a_i x_i
\]

\[
y_1 \quad y_2 \quad \ldots \quad y_n
\]
A cryptographic hash function $f$ has a parameter $n$; $f : \{0, 1\}^* \rightarrow \{0, 1\}^n$; $f$ should satisfy:

- Given “message” $M$, $f(M)$ is easy to compute.
- Given $b \in \{0, 1\}^n$, it is hard\(^3\) to find $M$ such that $f(M) = b$.
- Given $M$, it is hard to modify $M$ to get a different message with the same hash value $f(M)$.
- It is very unlikely that 2 random messages have the same hash value.

Usage: message authentication, digital signatures etc. (There is a nice Wikipedia page on this topic.) Suppose I solve a puzzle and want to prove to you I’ve solved it, without giving you any hint immediately, I should be able to use $f$ on my solution.

\(^3\)can’t be done in time polynomial in $n$
Birthday attack: Find 2 distinct strings $M$ and $M'$ such that $f(M) = f(M')$.
(so-called because it’s like the birthday problem of finding 2 people in a room who have the same birthday)
**Birthday attack:** Find 2 distinct strings $M$ and $M'$ such that $f(M) = f(M')$.

(so-called because it’s like the birthday problem of finding 2 people in a room who have the same birthday)

Resistance to birthday attack is a stronger property than the ones listed previously. Notice that birthday attack is in $\text{PPP}$. But, as far as I know it’s not known to be complete for $\text{PPP}$. 

Subclasses of \textbf{PPP}

Problem with \textbf{PPP}: no interesting \textbf{PPP}-completeness results. \textbf{PPP} fails to “capture the complexity” of apparently hard problems, such as \textbf{Nash}.

Here is a specialisation of the pigeonhole principle:

“\textit{Suppose directed graph }G\textit{ has indegree and outdegree at most 1. Given a source, there must be a sink.}”
Subclasses of PPP

Problem with PPP: no interesting PPP-completeness results. 
PPP fails to “capture the complexity” of apparently hard problems, such as Nash.

Here is a specialisation of the pigeonhole principle:

“This suppose directed graph $G$ has indegree and outdegree at most 1. Given a source, there must be a sink.”

Why is this the pigeonhole principle?

$G = (V, E); f : V \rightarrow V$ defined as follows:

For all $e = (u, v)$, let $f(u) = v$. If $u$ is a sink, let $f(u) = u$.

Let $s \in E$ be a source. So $s \not\in \text{range}(f)$. The pigeonhole principle says that 2 vertices must be mapped by $f$ to the same vertex.
Subclasses of PPP

\[ G = (V, E), \quad V = \{0, 1\}^n. \]

\(G\) is represented using 2 circuits \(P\) and \(S\) ("predecessor" and "successor") with \(n\) inputs/outputs.

\(G\) has \(2^n\) vertices (bit strings); \(0\) is source. \((x, x')\) is an edge iff \(x' = S(x)\) and \(x = P(x')\).

Thus, \(G\) is a BIG graph and it’s not clear how best to find a sink, even though you know it’s there!
\( G = (V, E), \ V = \{0, 1\}^n. \)

\( G \) is represented using 2 circuits \( P \) and \( S \) ("predecessor" and "successor") with \( n \) inputs/outputs.

\( G \) has \( 2^n \) vertices (bit strings); \( 0 \) is source. \((x, x')\) is an edge iff \( x' = S(x) \) and \( x = P(x') \).

Thus, \( G \) is a BIG graph and it’s not clear how best to find a sink, even though you know it’s there!
Search the graph for a sink

“brute-force” search will work but could easily take exponential time.
But, if you find a sink, it’s easy to check it’s genuine! So, search is in FNP.
Parity argument on a graph

A weaker version of the “there must be a sink”:

“Suppose directed graph $G$ has indegree and outdegree at most 1. Given a source, there must be another vertex that is either a source or a sink.”

**Definition:**

**INPUT:** graph $G$, source $v \in G$

**OUTPUT:** $v' \in G$, $v' \neq v$ is either a source or a sink

**PPAD** is defined in terms of the same way that **PPP** is defined in terms of **Pigeonhole circuit**.

Equivalent (more general-looking) formulation: If $G$ (not necessarily of in/out-degree 1) has an “unbalanced vertex”, then it must have another one. “parity argument on a directed graph”
and returning to the stronger version of this principle:

**Definition:** \textsc{Find A Sink}

\textbf{INPUT:} graph $G$, source $v \in G$

\textbf{OUTPUT:} $v' \in G$, $v'$ is a sink

This problem is clearly at least as hard. But we still believe \textsc{End of the Line} is hard.

\textbf{PPADS} is the complexity class defined \textit{w.r.t.} \textsc{Find A Sink}
So, we have discussed 3 complexity classes

$$PPAD \subseteq PPADS \subseteq PPP$$

because

End of the line $\leq_p$ Find a sink $\leq_p$ Pigeonhole circuit.

If we could e.g. reduce Find a sink back to End of the line, then that would show that PPAD and PPADS are the same, but this has not been achieved...
So, we have discussed 3 complexity classes

\[ \text{PPAD} \subseteq \text{PPADS} \subseteq \text{PPP} \]

because

End of the line \( \leq_p \) Find a sink \( \leq_p \) Pigeonhole circuit.

If we could e.g. reduce Find a sink back to End of the line, then that would show that PPAD and PPADS are the same, but this has not been achieved...

In the mean time, it turns out that PPAD is the sub-class of PPP that captures the complexity of Nash and related problems.
Given a graph $G$ (in terms of these two circuits $S$ and $P$) with source 0, there exists a sink $x$ such that $x = S(S(\ldots(S(0))\ldots))$. this is a total search problem, but completely different...
Given a graph $G$ (in terms of these two circuits $S$ and $P$) with source 0, there exists a sink $x$ such that $x = S(S(\ldots(S(0))\ldots))$. This is a total search problem, but completely different...

not a member of **NP** (apparently)
Given a graph $G$ (in terms of these two circuits $S$ and $P$) with source $0$, there exists a sink $x$ such that $x = S(S(\ldots(S(0))\ldots))$.

this is a total search problem, but completely different...

not a member of $\textbf{NP}$ (apparently)

In fact, $\textbf{PSPACE}$-complete — the search for this $x$ is computationally equivalent to search for the final configuration of a polynomially space-bounded Turing machine.
Finally, here is why we care about PPAD. It seems to capture the complexity of a number of problems where a solution is guaranteed by *Brouwer’s fixed point Theorem*.
Finally, here is why we care about **PPAD**. It seems to capture the complexity of a number of problems where a solution is guaranteed by *Brouwer’s fixed point Theorem*.

Two parts to the proof:

1. **Nash** is in **PPAD**, i.e. \( \text{Nash} \leq_p \text{End of the line} \)
2. **End of the line** \( \leq_p \text{Nash} \)

Done for 4, then 3 players by Daskalakis, G and Papadimitriou, then for 2 players by Chen, Deng and Teng.
Reducing Nash to End of the Line

We need to show \( \text{Nash} \leq_p \text{End of the line} \).
That is, we need two functions \( f \) and \( g \) such that given a game \( G \),
\[ f(G) = (P, S) \]
where \( P \) and \( S \) are circuits that define an End of the Line instance...
Given a solution \( x \) to \( (P, S) \), \( g(x) \) is a solution to \( G \).

Notes

- Nash is taken to mean: find an approximate NE
- Reduction is a computational version of Nash’s theorem
- Nash’s theorem uses Brouwer’s fixed point theorem, which in turn uses Sperner’s lemma; the reduction shows how these results are proven...
For a $k$-player game $G$, solution space is compact domain $(\Delta_n)^k$

Given a candidate solution $(p_1^1, \ldots, p_n^1, \ldots, p_1^k, \ldots, p_n^k)$, a point in this compact domain, $f_G$ displaces that point according to the direction that player(s) prefer to change their behavior.

$f_G$ is a *Brouwer* function, a continuous function from a compact domain to itself.

Brouwer FPT: There exists $x$ with $f_G(x) = x$ — why?
Reduction to **Brouwer**

$$\text{domain } (\Delta_n)^k$$

divide into simplices of size $\epsilon/n$

Arrows show direction of

Brouwer function, e.g. $f_G$

If $f_G$ is constructed sensibly, look for simplex where arrows go in all directions — *sufficient* condition for being near $\epsilon$-NE.
Reduction to Sperner

Color “grid points”:
- **red** direction away from top;
- **green** away from bottom RH corner
- **blue** away from bottom LH corner

$\left( \Delta_n \right)^k$: polytope in $R^{nk}$; $nk + 1$ colors.
Reduction to Sperner

Sperner’s Lemma (in 2-D): promises “trichromatic triangle”

If so, trichromatic triangles at increasingly higher and higher resolutions should lead us to a Brouwer fixpoint...
Reduction to Sperner

Let’s try that out (and then we’ll prove Sperner’s lemma)
Reduction to Sperner

Black spots show the trichromatic triangles
Reduction to \textit{Sperner}

Higher-resolution version

Goldberg  Game Theory and Computational Complexity
Reduction to **SPERNER**

Again, black spots show trichromatic triangles.
Reduction to Sperner

Once more — again we find trichromatic triangles!

Next: convince ourselves they always can be found, for any Brouwer function.
Sperner’s Lemma

Suppose we color the grid points under the constraint shown in the diagram. Why can we be sure that there is a trichromatic triangle?
Add some edges such that only one red/green edge is open to the outside.
Reduction to Sperner

red/green edges are "doorways" that connect the triangles

Move from triangle to triangle crossing red-green edges
Reduction to **SPERNER**

Keep going — we end up at a trichromatic triangle!
Reduction to Sperner

We can do the same trick w.r.t. the red/blue edges
Reduction to Sperner

Now the red/blue edges are doorways
Reduction to Sperner

Keep going through them — eventually find a panchromatic triangle!
Essentially, Sperner’s lemma converts the function into an END OF THE LINE graph!
Reduction to Sperner Degree-2 Directed Graph

Each little triangle is a vertex

Graph has one known source

Other than the known source, there must be an odd number of degree-1 vertices.
Reducing \textbf{End of the line to Nash}

- \textbf{End of the line} $\leq_p$ Brouwer
- Brouwer $\leq_p$ \textbf{Graphical Nash}
- \textbf{Graphical Nash} $\leq_p$ Nash

Some details in handouts; let me try to do some on the whiteboard...
Graphical games

Players 1, ..., n
Players: nodes of graph
G of low degree d
strategies 1, ..., t
utility depends on
strategies in
neighborhood
n.t^(d+1) numbers
describe game

Compact representation of game with many players.
Graphical Nash $\leq_{p} \text{Nash}$

Color the graph s.t.
- proper coloring
- each vertex’s neighbors get distinct colors

Normal-form game:
- one “super-player” for each color
- Each super-player simulates entire set of players having that color

Naive bound of $d^2 + 1$ on number of colors needed
So we have a small number of super-players (given that $d$ is small). **Problem:** If blue super-player chooses an action for each member of his “team” he has $t^n$ possible actions — can’t write that down in normal form!

**Solution:** Instead, he will just choose one member $v$ of his team at random, and choose an action for $v$, just $t^n$ possible actions! So what we have to do is:

Incentivize each super-player to pick a random team member $v$; and further, incentivize him to pick a best response for $v$ afterwards. This is done by choice of payoffs to super-players (in our graph, $\{\text{red}, \text{blue}, \text{green}, \text{brown}\}$).
So we have a small number of super-players (given that $d$ is small).

**Problem:** If blue super-player chooses an action for each member of his “team” he has $t^n$ possible actions — can’t write that down in normal form!

**Solution:** Instead, he will just choose one member $v$ of his team at random, and choose an action for $v$, just $t.n$ possible actions!
So we have a small number of super-players (given that \( d \) is small). **Problem:** If blue super-player chooses an action for each member of his “team” he has \( t^n \) possible actions — can’t write that down in normal form!

**Solution:** Instead, he will just choose one member \( v \) of his team at random, and choose an action for \( v \), just \( t \cdot n \) possible actions!

**so what we have to do is:** Incentivize each super-player to pick a random team member \( v \); and further, incentivize him to pick a best response for \( v \) afterwards.

This is done by choice of payoffs to super-players (in our graph, \( \{ \text{red, blue, green, brown} \} \)
Graphical Nash $\leq_p$ Nash

If we have coloring \{red, blue, green, brown\}
The actions of the red super-player are of the form: Choose a red vertex on the graph, then choose an action in \{1, ..., s\}.

Payoffs:

- If I choose a node $v$, and the other super-players choose nodes in $v$’s neighborhood, then red gets the payoff that $v$ would receive.

- Also, if red chooses the $i$-th red vertex (in some given ordering) and blue chooses his $i$-th vertex, then red receives (big) payoff $M$ and blue gets penalty $-M$ (and similarly for other pairs of super-players).

The 2nd of these means a super-player will randomize amongst nodes of his color in $G$. The first means that when he is chosen $v \in G$, his choice of $v$’s action should be a best response.
Why did we need a proper coloring? Because when a super-player chooses $v$, there should be some positive probability that $v$’s neighbors get chosen; AND these choices should be made independently.
Approximate Nash equilibria

Hardness results apply to $\epsilon = 1/n$; generally $\epsilon = 1/p(n)$ for polynomial $p$.

What if e.g. $\epsilon = 1/3$?

- 2 players - let $R$ and $C$ be matrices of row/column players’s utils.
- Let $x$ and $y$ denote the row and column players’ strategies; let $e_i$ be vector with 1 in component $i$, zero elsewhere.
- For all $i$, $x^T R y \geq e_i^T R y - \epsilon$.
- For all $j$, $x^T C y \geq x^T C e_j - \epsilon$.
- Remember: payoffs are re-scaled into $[0, 1]$. 

A simple algorithm

Daskalakis, Mehta and Papadimitriou, WINE’06: compute $\frac{1}{2}$-NE

Player 1 chooses arbitrary strategy $i$; gives it probability $\frac{1}{2}$. 
Daskalakis, Mehta and Papadimitriou, WINE’06: compute $\frac{1}{2}$-NE

1. Player 1 chooses arbitrary strategy $i$; gives it probability $\frac{1}{2}$.
2. Player 1 chooses best response $j$; gives it probability 1.
A simple algorithm

Daskalakis, Mehta and Papadimitriou, WINE’06: compute $\frac{1}{2}$-NE

<table>
<thead>
<tr>
<th></th>
<th>0.2</th>
<th>0.9</th>
<th>0.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.2</td>
<td>0.1</td>
<td>0.2</td>
</tr>
<tr>
<td>0.2</td>
<td>0.2</td>
<td>0.1</td>
<td>0.2</td>
</tr>
<tr>
<td>0.3</td>
<td>0.4</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>0.2</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td>0.7</td>
<td>0.8</td>
<td></td>
</tr>
</tbody>
</table>

1. Player 1 chooses arbitrary strategy $i$; gives it probability $\frac{1}{2}$.
2. Player 1 chooses best response $j$; gives it probability 1.
3. Player 1 chooses best response to $j$; gives it probability $\frac{1}{2}$.
How to find approximate solutions with $\epsilon < \frac{1}{2}$?

The previous algorithm was so simple that it looks like we should easily be able to improve it!

The support of a probability distribution is the set of events that get non-zero probability — for a mixed strategy, all the pure strategies that may get chosen. In the previous algorithm, player 1's mixed strategy had support $\leq 2$ and player 2's had support 1.
How to find approximate solutions with \( \epsilon < \frac{1}{2} \)?

The previous algorithm was so simple that it looks like we should easily be able to improve it!
But, it’s not so easy... next we will see that an algorithm for \( \epsilon < \frac{1}{2} \) would need to find mixed strategies having more than a constant support size.
The support of a probability distribution is the set of events that get non-zero probability — for a mixed strategy, all the pure strategies that may get chosen. In the previous algorithm, player 1’s mixed strategy had support \( \leq 2 \) and player 2’s had support 1.
Why you need more than constant support size to get $\epsilon < \frac{1}{2}$:

Feder, Nazerzadeh and Saberi (EC’07) consider random zero-sum win-lose games of size $n \times n$:
Why you need more than constant support size to get $\epsilon < \frac{1}{2}$:

Feder, Nazerzadeh and Saberi (EC’07) consider random zero-sum win-lose games of size $n \times n$:

With high probability, for any pure strategy by player 1, player 2 can “win”
Why you need more than constant support size to get $\epsilon < \frac{1}{2}$:

Feder, Nazerzadeh and Saberi (EC’07) consider random zero-sum win-lose games of size $n \times n$:

With high probability, for any pure strategy by player 1, player 2 can “win”:

1. Indeed, as $n$ increases, this is true if player 1 may mix 2 of his strategies.
Why you need more than constant support size to get $\epsilon < \frac{1}{2}$:

- But, for large $n$, player 1 can guarantee a payoff of about $1/2$ by randomizing over his strategies (w.h.p., as $n$ increases)
- Given any constant support size $\kappa$, there is $n$ large enough such that the other player can win against any mixed strategy that uses $\kappa$ pure strategies. So, small-support strategies are $1/2$ worse than the fully-mixed strategy.
If less than $\log(n)$ strategies are used by player 1, there is a high probability that player 2 can win...
Hence $\Omega(\log(n))$ is a lower bound on support size needed.

And in fact, $O(\log(n))$ is also sufficiently large...
Work on approximate equilibria for the 2-player case has led to some interesting algorithms, obtaining $\epsilon$-values of about $1/3$. (This seems surprisingly weak; and also, since $\epsilon$-NE can be found in sub-exponential time for any fixed $\epsilon > 0$, we suspect there should be a PTAS). For $k > 2$ players, even less is known.

Work by Daskalakis and Papadimitriou on *anonymous games*: PPAD-complete to solve exactly, but they do find a PTAS.