Computability and Complexity Theory: An Introduction

Meena Mahajan
meena@imsc.res.in
http://www.imsc.res.in/~meena
Understanding Computation

Kinds of questions we seek answers to:

- Is a given graph planar?
- Can a given integer matrix be made singular by changing at most one entry?
- Is a given number prime?
- Does a player in a certain game have a winning strategy?
- Is a given formula a tautology?
- Does a given Diophantine equation have a solution?
Kinds of answers we seek:

• A list of answers to all instances?
  Too long a list; infinite.
Kinds of answers we seek:

- A list of answers to all instances? Too long a list; infinite.
- A list of answers to interesting instances? Still too long.
**Understanding Computation ...**

Kinds of answers we seek:

- A list of answers to all instances?  
  Too long a list; infinite.

- A list of answers to **interesting** instances?  
  Still too long.

- A finitely described procedure that, when applied to any instance, gives the correct answer in finite time.  
  i.e. an **effective procedure**.
Example: Testing planarity

- Input: A Graph $G$
Example: Testing planarity

- Input: A Graph $G$
- Strategy: Try out all possible combinatorial embeddings (cyclic ordering of edges incident at a vertex)
Example: Testing planarity

- Input: A Graph $G$
- Strategy: Try out all possible combinatorial embeddings (cyclic ordering of edges incident at a vertex)
- For each combinatorial embedding, compute its genus.
Example: Testing planarity

- Input: A Graph $G$
- Strategy: Try out all possible combinatorial embeddings (cyclic ordering of edges incident at a vertex)
- For each combinatorial embedding, compute its genus.
- If for some embedding, the genus is 0, stop and say Yes.
Example: Testing planarity

- Input: A Graph \( G \)
- Strategy: Try out all possible combinatorial embeddings (cyclic ordering of edges incident at a vertex)
- For each combinatorial embedding, compute its genus.
- If for some embedding, the genus is 0, stop and say Yes.
- If no such embedding is found, stop and say No.
A Non-Example

• Input: An $n \times n$ matrix $A$ with integer entries. Decide if $A$ can be made singular by changing at most one entry.
A Non-Example

• Input: An $n \times n$ matrix $A$ with integer entries. Decide if $A$ can be made singular by changing at most one entry.

• If $A$ is already singular, stop and say Yes. Otherwise ...
A Non-Example

• Input: An $n \times n$ matrix $A$ with integer entries. Decide if $A$ can be made singular by changing at most one entry.

• If $A$ is already singular, stop and say Yes. Otherwise ...

• Strategy: The set $\mathbb{Z}$ of all integers is countably infinite. For each $x \in \mathbb{Z}$ and each $i, j \in [n]$, check if replacing $A[i, j]$ by $x$ gives a singular matrix. If Yes, halt and report Yes, else try the next triple $x', i', j'$. 
A Non-Example

- Input: An $n \times n$ matrix $A$ with integer entries. Decide if $A$ can be made singular by changing at most one entry.
- If $A$ is already singular, stop and say Yes. Otherwise ...
- Strategy: The set $\mathbb{Z}$ of all integers is countably infinite. For each $x \in \mathbb{Z}$ and each $i, j \in [n]$, check if replacing $A[i, j]$ by $x$ gives a singular matrix. If Yes, halt and report Yes, else try the next triple $x', i', j'$.
- If $A$ can be made singular, we will find a witness and report Yes. Otherwise, the procedure will run forever.
Another Non-Example

• Input: A Diophantine equation $E$. Decide if it has an integer solution.
Another Non-Example

- **Input:** A Diophantine equation $E$. Decide if it has an integer solution.

- **Strategy:** Let $E$ have $n$ variables. The set $\mathbb{Z}^n$, of all possible integer assignments to the variables, is countably infinite. Try out each assignment.
Another Non-Example

• Input: A Diophantine equation $E$. Decide if it has an integer solution.

• Strategy: Let $E$ have $n$ variables. The set $\mathbb{Z}^n$, of all possible integer assignments to the variables, is countably infinite. Try out each assignment.

• If there is a solution, we will eventually find it and stop. If there is none, the procedure will run forever.
Decidability

- Effective procedures (also known as recursive procedures): finitely described procedures that eventually halt on every input. The standard formalism is via Turing machines, but there are many other equivalent ones.
Decidability

• Effective procedures (also known as recursive procedures): finitely described procedures that eventually halt on every input. The standard formalism is via Turing machines, but there are many other equivalent ones.

• Problems that can be solved by effective procedures are said to be decidable.
Decidability

- **Effective procedures** (also known as recursive procedures): finitely described procedures that eventually halt on every input. The standard formalism is via Turing machines, but there are many other equivalent ones.

- Problems that can be solved by effective procedures are said to be **decidable**.

- **Undecidable** problems have no effective procedures.
Decidability

- **Effective procedures** (also known as recursive procedures): finitely described procedures that eventually halt on every input. The standard formalism is via Turing machines, but there are many other equivalent ones.
- Problems that can be solved by effective procedures are said to be **decidable**.
- **Undecidable** problems have no effective procedures.
- Some problems can be proved to be undecidable by a diagonalization argument.
A diagonalization example

• Suppose there is an effective procedure $P$ to decide the Halting Problem:

$P$

$Q$ $\rightarrow$ Yes; $Q$ halts on $w$

$w$ $\rightarrow$ No; $Q$ loops on $w$
A diagonalization example

• Suppose there is an effective procedure $P$ to decide the Halting Problem:

• Obtain $P'$ from $P$ as follows:
A diagonalization example

• Suppose there is an effective procedure P to decide the Halting Problem:

• Obtain P’ from P as follows:

- P’(Q) loops iff P(Q,Q) says Yes iff Q halts on Q.
- P’(Q) halts iff P(Q,Q) says No iff Q loops on Q.
A diagonalization example

- Suppose there is an effective procedure P to decide the Halting Problem:
- Obtain P’ from P as follows:

  - P’(Q) loops iff P(Q,Q) says Yes iff Q halts on Q.
  - P’(Q) halts iff P(Q,Q) says No iff Q loops on Q.
  - What does P’ do on input P’?
Some decidable problems

• Is a given graph planar? We saw an effective procedure for this.

• Can a given matrix over $\mathbb{Z}$ be made singular by changing at most one entry? The procedure we saw was not effective, but there is another procedure that is effective.

• Is a given number prime?

• Given a finite set of rewrite rules $\alpha \rightarrow \beta$, strings $w, x$ and a natural number $k$, can $x$ be obtained from $w$ through at most $k$ applications of rewrite rules?
Some undecidable problems

• The Halting Problem: Does a given program never get into an infinite loop?
Some undecidable problems

• The Halting Problem: Does a given program never get into an infinite loop?

• Given a finite set of rewrite rules $\alpha \rightarrow \beta$ and strings $w, x$, can $x$ be obtained from $w$ through a sequence of applications of rewrite rules?
Some undecidable problems

- The Halting Problem: Does a given program never get into an infinite loop?
- Given a finite set of rewrite rules $\alpha \rightarrow \beta$ and strings $w, x$, can $x$ be obtained from $w$ through a sequence of applications of rewrite rules?
- Does a given Diophantine equation have a solution?
Some undecidable problems

- The Halting Problem: Does a given program never get into an infinite loop?
- Given a finite set of rewrite rules $\alpha \rightarrow \beta$ and strings $w, x$, can $x$ be obtained from $w$ through a sequence of applications of rewrite rules?
- Does a given Diophantine equation have a solution?
- Given a matrix with entries from $\{0, 1\} \cup \{x_1, \ldots, x_t\}$, is there an integer assignment to the variables $x_i$ that makes the matrix singular?
Use of Resources

• How much space / time does a given procedure use? — space / time complexity of the given procedure.

• Is there an equivalent procedure that uses less? What is the minimum needed? i.e. How much is necessary, how much is sufficient? — inherent space / time complexity of the problem.

• To show sufficiency, describe any procedure that uses that much.

• To show necessity, a lower bound proof is needed.
Planarity testing

- The planarity testing procedure we saw earlier used space proportional to $|E|$. It tried out all combinatorial embeddings one by one, so it needed enough space to write down the embedding currently being tested.
Planarity testing

• The planarity testing procedure we saw earlier used space proportional to $|E|$. It tried out all combinatorial embeddings one by one, so it needed enough space to write down the embedding currently being tested.

• It also needed time as large as $c|E| + |V|$, since there are those many embeddings.
Planarity testing

• The planarity testing procedure we saw earlier used space proportional to $|E|$. It tried out all combinatorial embeddings one by one, so it needed enough space to write down the embedding currently being tested.

• It also needed time as large as $c|E| + |V|$, since there are those many embeddings.

• But planarity testing is in fact possible in space proportional to $\log(|V| + |E|)$, and time proportional to $|V||E|$. 
Some complexity classes

Space bounds: logarithmic, polynomial, exponential ...
Time bounds: polynomial, exponential ...

Log → PSPACE → EXPSPACE

P → EXPTIME
Some complexity classes

Space bounds: logarithmic, polynomial, exponential ...
Time bounds: polynomial, exponential ...

Log ↓ PSPACE
   ↓ P
   ↓ EXPSPACE

P↑↑↑↑↑↑↑↑↑↑↑
EXPTIME

**Some complexity classes**

Space bounds: logarithmic, polynomial, exponential ...

Time bounds: polynomial, exponential ...

Log → P → PSPACE → EXPTIME → EXPSPACE

Polynomial time $P$ is considered to capture **feasible** or **tractable** computation.

Many important natural problems are in $P$:

- Decide whether a graph has a perfect matching.
- Evaluate a circuit with AND, OR, NOT gates.
- Decide whether there is a path from $s$ to $t$ in a given directed graph.
Short certificates and NP

- Many important natural classification problems are not known to be in P: Are two given graphs isomorphic? Does a linear program have an integer solution?
Short certificates and NP

• Many important natural classification problems are not known to be in P: Are two given graphs isomorphic? Does a linear program have an integer solution?

• However, some of these have a nice property: when the input belongs to the specified class, there is a short, convincing proof of this.
Short certificates and NP

• Many important natural classification problems are not known to be in P: Are two given graphs isomorphic? Does a linear program have an integer solution?

• However, some of these have a nice property: when the input belongs to the specified class, there is a short, convincing proof of this.

• e.g. To prove that $G_1$ and $G_2$ are isomorphic, just give the mapping $f : V(G_1) \rightarrow V(G_2)$. A polynomial-time verifier can check that $f$ is indeed an isomorphism.
Short certificates and NP

• Many important natural classification problems are not known to be in P: Are two given graphs isomorphic? Does a linear program have an integer solution?

• However, some of these have a nice property: when the input belongs to the specified class, there is a short, convincing proof of this.

• To prove that a linear program has an integer solution, just give such a solution. Verifying that it is indeed a solution is in P.
Short certificates and NP

- Many important natural classification problems are not known to be in \( P \): Are two given graphs isomorphic? Does a linear program have an integer solution?
- However, some of these have a nice property: when the input belongs to the specified class, there is a short, convincing proof of this.
- \( \text{NP} \) is exactly the class of such problems: properties \( C \) for which membership (statements of the form \( x \in C \)) has short, efficiently verifiable proofs.
- \( \text{NP} \) stands for Non-deterministic Polynomial time. A non-deterministic procedure can guess the short convincing proof in polynomial time, and then verify that it is indeed a proof.
The classes NP and co-NP

- Clearly, $P \subseteq NP$. The million dollar question is whether all problems in NP are in fact in P.
The classes NP and co-NP

• Clearly, $P \subseteq NP$. The million dollar question is whether all problems in NP are in fact in P.

• A related question: Are NP and co-NP the same? i.e. if a property $C$ has short efficiently verifiable proofs, does the property $\overline{C}$ have such proofs too?

What would be a short verifiable proof that $G_1$ and $G_2$ are not isomorphic?
The classes \textit{NP} and \textit{co-NP}

- Clearly, $P \subseteq \text{NP}$. The million dollar question is whether all problems in \text{NP} are in fact in \text{P}.

- A related question: Are \text{NP} and \text{co-NP} the same? i.e. if a property $C$ has short efficiently verifiable proofs, does the property $\overline{C}$ have such proofs too?

  What would be a short verifiable proof that $G_1$ and $G_2$ are not isomorphic?

- It is generally believed that $P \neq \text{NP} \cap \text{co-NP} \neq \text{NP}$. 
Other nondeterministic classes

- NLog: problems with nondeterministic logarithmic space algorithms.
Other nondeterministic classes

- $N\text{Log}$: problems with nondeterministic logarithmic space algorithms.
- This is more restrictive than having a short proof checkable in logspace. Even the process of nondeterministically guessing a proof should not need more than logspace. Can a graph isomorphism be guessed in logspace?
Other nondeterministic classes

- **NLog**: problems with nondeterministic logarithmic space algorithms.

- This is more restrictive than having a short proof checkable in logspace. Even the process of nondeterministically guessing a proof should not need more than logspace. Can a graph isomorphism be guessed in logspace?

- **NPSPACE**: problems with nondeterministic polynomial space algorithms. Proofs can be exponentially long, but need to be guessable and checkable in polynomial space.
Relating the classes

From the definitions,

\[
\text{Log} \rightarrow \text{P} \rightarrow \text{PSPACE} \\
\downarrow \quad \downarrow \quad \downarrow \\
\text{NLog} \quad \text{NP} \cap \text{co-NP} \rightarrow \text{NP} \\
\downarrow \quad \downarrow \quad \downarrow \\
\text{NLogNP} \quad \text{NP} \quad \text{NPSPACE}
\]
**Relating the classes**

From the definitions,

\[
\begin{align*}
\text{Log} &\quad \rightarrow \quad \text{P} &\quad \rightarrow \quad \text{PSPACE} \\
\text{NLog} &\quad \rightarrow \quad \text{NP} \cap \text{co-NP} &\quad \rightarrow \quad \text{NP} &\quad \rightarrow \quad \text{NPSPACE}
\end{align*}
\]

In fact, much more is known:

\[
\begin{align*}
\text{Log} &\quad \rightarrow \quad \text{P} &\quad \rightarrow \quad \text{PSPACE} \\
\text{NLog} &\quad \rightarrow \quad \text{co-NLog} &\quad \rightarrow \quad \text{NP} \cap \text{co-NP} &\quad \rightarrow \quad \text{NP} &\quad \rightarrow \quad \text{NPSPACE}
\end{align*}
\]
The notion of reductions

How does one compare the computational difficulty of problems? We want to make statements like:

- Graph Isomorphism is no harder than Integer Programming, and may be easier.
- Integer Programming and Hamiltonian Cycle are equally hard.

Testing property \( C \) is no harder than testing property \( D \) if any instance \( x \) can be efficiently (in \( P \), or \( \log \)) transformed to an instance \( y \) such that \( x \in C \iff y \in D \). Such a transformation is called a many-one reduction, \( \leq_m \).
What reductions achieve

- Reductions impose a partial order on the set of all problems. Problems in an equivalent class $\equiv_m$ reduce to each other.
What reductions achieve

- Reductions impose a partial order on the set of all problems. Problems in an equivalent class $\equiv_m$ reduce to each other.

- Reasonable complexity classes do not slice through an equivalence class $\equiv_m$. They equal the union of some such classes, and are usually downward closed.
What reductions achieve

• Reductions impose a partial order on the set of all problems. Problems in an equivalent class $\equiv_m$ reduce to each other.

• Reasonable complexity classes do not slice through an equivalence class $\equiv_m$. They equal the union of some such classes, and are usually downward closed.

• In the partial order induced by $\leq_m$, a complexity class $C$ can have many upper bounds.
What reductions achieve

• Reductions impose a partial order on the set of all problems. Problems in an equivalent class \( \equiv_m \) reduce to each other.

• Reasonable complexity classes do not slice through an equivalence class \( \equiv_m \). They equal the union of some such classes, and are usually downward closed.

• In the partial order induced by \( \leq_m \), a complexity class \( C \) can have many upper bounds.

• Problems in an \( \equiv_m \) class that is an upper bound for a complexity class \( C \) are said to be hard for \( C \).
What reductions achieve

• Reductions impose a partial order on the set of all problems. Problems in an equivalent class $\equiv_m$ reduce to each other.

• Reasonable complexity classes do not slice through an equivalence class $\equiv_m$. They equal the union of some such classes, and are usually downward closed.

• In the partial order induced by $\leq_m$, a complexity class $C$ can have many upper bounds.

• Problems in an $\equiv_m$ class that is an upper bound for a complexity class $C$ are said to be hard for $C$.

• If an upper bound is inside $C$, then its $\equiv_m$ class is a least upper bound. Such problems are said to be complete for $C$. 
NP-completeness

A problem is NP-complete if

- it is in NP, and
- it is an upper bound for NP. i.e. every problem in NP reduces to it.

The Cook-Levin theorem says that Boolean satisfiability SAT is NP-complete.

Subsequently, several other problems in NP have been shown to be NP-complete by reducing SAT, or any other NP-complete problem, to them. These problems are from diverse fields.
Combinatorial NP-complete problems

- **Boolean Formula Satisfiability**: Is there a True/False assignment to variables $x_1, \ldots, x_n$ that makes the Boolean formula $F(x_1, \ldots, x_n)$ true?

- **Integer programming**: Is there an integer solution to the system of inequalities $Ax \leq b$?

- **Hamiltonian circuit problem**: Does a directed graph $G$ have a Hamiltonian cycle?

- **Sorting by Reversals**: Can a string $x_1, \ldots, x_n$ be sorted in fewer than $k$ moves, where a move consists of picking a substring and reversing it?
Algebraic NP-complete problems

• Radius of Non-Singularity
  
  **Instance:** A rational square matrix $A$, a rational $\theta$.
  
  **Decide:** Can $A$ be made singular by changing each entry by at most $\theta$?

• Satisfiability of Quadratic Polynomials:
  
  **Instance:** $t$ degree-2 polynomials $P_1, \ldots, P_t$ on $n$ elements from $\mathbb{F}_2$.
  
  **Decide:** Do these polynomials have a common zero?
NP-completeness in coding theory

- Maximum Likelihood Decoding: Given a binary linear code (via its parity check matrix $H$), and a word $y$, find the codeword $c$ nearest to it.
NP-completeness in coding theory

- **Maximum Likelihood Decoding**: Given a binary linear code (via its parity check matrix $H$), and a word $y$, find the codeword $c$ nearest to it.

- **For a codeword $c$, $Hc^T = 0$.** So if $y = c + e$, then $He^T = Hy^T = s$. For the codeword nearest $y$, the error term $e$ has smallest Hamming weight. So

**Input**: A binary $m \times n$ matrix $H$, a vector $s \in \mathbb{F}_2^m$, and an integer $w > 0$.

**Decide**: Is there a vector $x \in \mathbb{F}_2^n$ of weight $\leq w$, such that $Hx^T = s$?
**NP-completeness in coding theory**

- **Maximum Likelihood Decoding**: Given a binary linear code (via its parity check matrix $H$), and a word $y$, find the codeword $c$ nearest to it.

- For a codeword $c$, $Hc^T = 0$. So if $y = c + e$, then $He^T = Hy^T = s$. For the codeword nearest $y$, the error term $e$ has smallest Hamming weight. So

**Input**: A binary $m \times n$ matrix $H$, a vector $s \in \mathbb{F}_2^m$, and an integer $w > 0$.

**Decide**: Is there a vector $x \in \mathbb{F}_2^n$ of weight $\leq w$, such that $Hx^T = s$?

- **The Minimum Distance problem**: $s = 0$ and we require $x \neq 0$. Also NP-hard.
More NP-completeness in coding

• Maximum Distance Separable Code Any linear code encoding \( k \) bits into \( n \) bits can have distance \( d \leq n - (k - 1) \). Does a given linear code achieve this maximum separation?
More NP-completeness in coding

- **Maximum Distance Separable Code** Any linear code encoding $k$ bits into $n$ bits can have distance $d \leq n - (k - 1)$. Does a given linear code achieve this maximum separation?

- The following related decision version is NP-hard:
More NP-completeness in coding

- Maximum Distance Separable Code: Any linear code encoding \( k \) bits into \( n \) bits can have distance \( d \leq n - (k - 1) \). Does a given linear code achieve this maximum separation?

- The following related decision version is NP-hard:

  **Input:** A prime \( p \), positive integers \( m \) and \( w \), and an \( r \times p^m \) matrix \( H \) over \( \mathbb{F}_{p^m} \).
More NP-completeness in coding

• Maximum Distance Separable Code Any linear code encoding \( k \) bits into \( n \) bits can have distance \( d \leq n - (k - 1) \). Does a given linear code achieve this maximum separation?

• The following related decision version is NP-hard:

  Input: A prime \( p \), positive integers \( m \) and \( w \), and an \( r \times p^m \) matrix \( H \) over \( \mathbb{F}_{p^m} \).

  Decide: Is there a non-zero vector \( x \) of weight \( \leq w \) and length \( p^m \) over \( \mathbb{F}_{p^m} \), such that \( Hx^T = 0 \)?
Completeness and implications

- If $\Pi$ is complete for $C$, and an efficient algorithm for $\Pi$ is found, then composing it with the reduction gives an efficient algorithm for every problem in $C$. 
Completeness and implications

• If $\Pi$ is complete for $C$, and an efficient algorithm for $\Pi$ is found, then composing it with the reduction gives an efficient algorithm for every problem in $C$.

• Since SAT is $\text{NP}$-complete, $\text{SAT} \in P \iff P = \text{NP}$. 
Completeness and implications

• If $\Pi$ is complete for $C$, and an efficient algorithm for $\Pi$ is found, then composing it with the reduction gives an efficient algorithm for every problem in $C$.

• Since SAT is NP-complete, $\text{SAT} \in P \iff P = \text{NP}$.

• Horn-SAT: the restriction of SAT to formulae in conjunctive normal form (CNF) where each clause has at most one negated literal. Horn-SAT is complete for $P$ under $\frac{\log m}{m}$ reductions. Thus, $\text{HornSAT} \in \text{NLog} \iff \text{NLog} = P$. 
Completeness and implications

• If $\Pi$ is complete for $C$, and an efficient algorithm for $\Pi$ is found, then composing it with the reduction gives an efficient algorithm for every problem in $C$.

• Since SAT is NP-complete, $\text{SAT} \in P \iff P = NP$.

• Horn-SAT: the restriction of SAT to formulae in conjunctive normal form (CNF) where each clause has at most one negated literal. Horn-SAT is complete for $P$ under $\leq_{\log m}$ reductions. Thus, $\text{HornSAT} \in \text{NLog} \iff \text{NLog} = P$.

• 2-SAT: The restriction of SAT to formulae in CNF where each clause has at most 2 literals. 2-SAT is complete for NLog under $\leq_{\log m}$ reductions. Thus, $\text{2SAT} \in \text{Log} \iff \text{Log} = \text{NLog}$.
NLog-\textit{hardness}

Some NLog-complete problems:

\begin{itemize}
  \item Given a directed layered acyclic graph $G$ and vertices $s, t$, does $G$ have a path from $s$ to $t$?
  \item 2SAT: Is a given 2CNF formula satisfiable?
  \item Given a perfect matching $M$ in a bipartite graph $G$, does $G$ have any other perfect matching?
\end{itemize}

Some NLog-hard problems in $P$:

\begin{itemize}
  \item Does a given graph have a perfect matching?
  \item Is a given integer matrix singular?
\end{itemize}
NLog-hardness of Singularity

We reduce Reachability to Non-Singularity.

**Input:** a directed layered acyclic graph $G$, vertices $s, t$ at first and last layer respectively.

**Output:** a matrix $A$ satisfying

$$\text{Det} A = 0 \iff s \not\rightarrow_G t$$

**Strategy:**

- Subdivide each edge of $G$.
- Add self-loops at all vertices except $s$.
- Add edge $t \rightarrow s$.
- Output adjacency matrix $A$ of this graph $H$. 
The construction
Why this works

- $s \sim t$ paths in $G \iff$ Cycle covers in $H$
Why this works

- $s \sim t$ paths in $G \iff$ Cycle covers in $H$
- cycle covers in $H \iff$ Terms in $\text{Det}(A)$
Why this works

• $s \rightsquigarrow t$ paths in $G \iff$ Cycle covers in $H$
• cycle covers in $H \iff$ Terms in $\text{Det}(A)$
• Value of term = Weight of cycle cover = 1
Why this works

- $s \sim t$ paths in $G \iff$ Cycle covers in $H$
- cycle covers in $H \iff$ Terms in $\text{Det}(A)$
- Value of term = Weight of cycle cover = 1
- Sign of term = $(-1)^{\#\text{even cycles}}$
Why this works

- $s \leadsto t$ paths in $\mathcal{G} \iff$ Cycle covers in $\mathcal{H}$
- cycle covers in $\mathcal{H} \iff$ Terms in $\det(A)$
- Value of term = Weight of cycle cover = 1
- Sign of term = $(-1)^{\text{#even cycles}}$
- But in $\mathcal{H}$, there are no even cycles.
  Hence $\det(A) = \#s \leadsto_{\mathcal{G}} t$
Radius of Singularity

Instance: A rational square matrix $A$, a rational $\theta$.

Decide: Can $A$ be made singular by changing each entry by at most $\theta$?
Radius of Singularity in NP

Suppose there is a singular $B$ with $|B_{ij} - A_{ij}| \leq \theta$ for each $i, j$. Can $B$ be guessed in polynomial time? Depends on what precision $B$ needs.
Radius of Singularity in NP

Suppose there is a singular $B$ with $|B_{ij} - A_{ij}| \leq \theta$ for each $i, j$. Can $B$ be guessed in polynomial time? Depends on what precision $B$ needs.

**Theorem:** If there is such a singular matrix, then in fact there is a position $k, l$ and singular $B$ such that

- $A_{kl} - \theta \leq B_{kl} \leq A_{kl} + \theta$
- For $ij \neq kl$, $B_{ij} \in \{A_{ij} - \theta, A_{ij} + \theta\}$
Radius of Singularity in NP

Suppose there is a singular $B$ with $|B_{ij} - A_{ij}| \leq \theta$ for each $i, j$. Can $B$ be guessed in polynomial time? Depends on what precision $B$ needs.

**Theorem:** If there is such a singular matrix, then in fact there is a position $k, l$ and singular $B$ such that

- $A_{kl} - \theta \leq B_{kl} \leq A_{kl} + \theta$
- For $ij \neq kl$, $B_{ij} \in \{A_{ij} - \theta, A_{ij} + \theta\}$

This implies that $B_{kl}$, and hence all of $B$, requires precision polynomial in $A, \theta$. 

Zero-on-an-Edge Lemma

- $p(x_1 \ldots x_t)$: a multilinear polynomial over $\mathbb{Q}$.
- Promise: $p(.)$ has a zero in the hypercube $H$ defined by $[\ell_1, u_1], \ldots [\ell_t, u_t]$.
- Then: $p(.)$ has a zero $(a_1, \ldots, a_t)$ on an edge of $H$ i.e. for some $k$, $\forall (i \neq k), a_i \in \{\ell_i, u_i\}$.
- Proof: By induction.
NP-hardness of Radius of Singularity

Reduction from MaxCUT

Instance: An undirected simple graph $G = (V, E)$, an integer $k$.

Decide: Does $G$ have a cut of size at least $k$? i.e. Can $V$ be partitioned into $S, \overline{S}$ so that $|E \cap (S \times \overline{S})| \geq k$?
The Construction

Define the matrix $N = (2m + 1)I - A$, where $A$ is the adjacency matrix of $G$; i.e.

$$N_{ij} = \begin{cases} 
2m + 1 & \text{if } i = j \\
-1 & \text{if } i \neq j \text{ and } (i, j) \in E \\
0 & \text{otherwise}
\end{cases}$$
The Construction

Define the matrix $N = (2m + 1)I - A$, where $A$ is the adjacency matrix of $G$; i.e.

$$N_{ij} = \begin{cases} 
2m + 1 & \text{if } i = j \\
-1 & \text{if } i \neq j \text{ and } (i, j) \in E \\
0 & \text{otherwise}
\end{cases}$$

$N$ is not singular.
(Any strictly diagonally dominant matrix is non-singular.
$|A_{ii}| > \sum_{j \neq i} |A_{ij}|$)
The Construction

Define the matrix $N = (2m + 1)I - A$, where $A$ is the adjacency matrix of $G$; i.e.

$$N_{ij} = \begin{cases} 
2m + 1 & \text{if } i = j \\
-1 & \text{if } i \neq j \text{ and } (i, j) \in E \\
0 & \text{otherwise}
\end{cases}$$

$N$ is not singular.

Theorem: $G$ has a cut of size $k$ iff $M = N^{-1}$ can be made singular by changing each entry by at most

$$\frac{1}{(2m+1)n+4k-2m}.$$
Why it works:  

Fix any $y, z \in \{-1, +1\}^n$.  

$\lambda$: a non-zero eigen-value of $Nyz^T; Nyz^T x = \lambda x$.  

• Claim 1: $\lambda = z^T Ny$.  

Why it works:  

Fix any \( y, z \in \{-1, +1\}^n \).

\( \lambda \): a non-zero eigen-value of \( Nyz^T \); \( Nyz^Tx = \lambda x \).

- Claim 1: \( \lambda = z^TNy \).
- Claim 2: Changes of \( 1/\lambda \) suffice to make \( N^{-1} \) singular.

\[(\lambda I - Nyz^T)x = 0, \text{ so } (N^{-1} - \frac{1}{\lambda}yz^T)x = 0.\]
Why it works:  

Fix any $y, z \in \{-1, +1\}^n$.

$\lambda$: a non-zero eigen-value of $Nyz^T$; $Ny z^T x = \lambda x$.

- **Claim 1:** $\lambda = z^T Ny$.
- **Claim 2:** Changes of $1/\lambda$ suffice to make $N^{-1}$ singular.
- **Claim 3:** For $S = \{i \mid y_i = +1\}$, let cut size be $\delta(S)$. Then $y^T Ny = 4\delta(S) + (2m + 1)n - 2m$.

Straightforward manipulation; rewrite $y_i y_j$ as $(-1/2)((y_i - y_j)^2 - 2)$.
Why it works: ⇒

Fix any $y, z \in \{-1, +1\}^n$.

$\lambda$: a non-zero eigen-value of $Nyz^T$; $Nyz^Tx = \lambda x$.

• Claim 1: $\lambda = z^TNy$.
• Claim 2: Changes of $1/\lambda$ suffice to make $N^{-1}$ singular.
• Claim 3: For $S = \{i \mid y_i = +1\}$, let cut size be $\delta(S)$. Then $y^TNy = 4\delta(S) + (2m + 1)n - 2m$.

$G$ has a cut $(S, \overline{S})$ of size $k$ ⇒ for the corresponding $-1, +1$ vector $y$: $y^TNy \geq 4k + (2m + 1)n - 2m$. Changes of $1/y^TNy$ suffice to singularize $N^{-1}$. 
Why it works:

• Suppose a singular $A$ satisfies

$$|A_{ij} - N_{ij}^{-1}| \leq \alpha = \frac{1}{4k + (2m + 1)n - 2m}$$
Why it works:  

- Suppose a singular $A$ satisfies
  \[ |A_{ij} - N^{-1}_{ij}| \leq \alpha = \frac{1}{4k+(2m+1)n-2m} \]

- There is a $t \in [-1, +1]^n$, $z \in \{-1, +1\}^n$:
  $N^{-1} - \alpha t z^T$ is singular.

Why? Let $Ax = 0$ for some non-zero $x$. Choose

$z_i = \text{sgn}(x_i)$, $t_i = (N^{-1}x)_i/\alpha X$ where

$X = \sum_j |x_j| = z^T x$. 
Why it works:

- Suppose a singular $A$ satisfies
  $$|A_{ij} - N_{ij}^{-1}| \leq \alpha = \frac{1}{4k+(2m+1)n-2m}$$

- There is a $t \in [-1, +1]^n$, $z \in \{-1, +1\}^n$: $N^{-1} - \alpha tz^T$ is singular.

- There is a $y \in \{-1, +1\}^n$, $0 < \beta \leq 1$ such that $N^{-1} - \alpha \beta yz^T$ is singular.

Why? Use the Zero-on-an-Edge Lemma to get a $t'$ on an edge. One endpoint $y$ of this edge has sign opposite that of $\det(N^{-1})$; interpolate between $\det$ of $N^{-1}$ and $N^{-1} - \alpha yz^T$ to obtain $\beta$. 
Why it works:

- Suppose a singular $A$ satisfies
  $$|A_{ij} - N_{ij}^{-1}| \leq \alpha = \frac{1}{4k+(2m+1)n-2m}$$

- There is a $t \in [-1, +1]^n$, $z \in \{-1, +1\}^n$: $N^{-1} - \alpha tz^T$ is singular.

- There is a $y \in \{-1, +1\}^n$, $0 < \beta \leq 1$ such that $N^{-1} - \alpha \beta yz^T$ is singular.

- $\max_{y,z\in\{-1,+1\}^n} z^T Ny$ is achieved at $z = y$. 
Why it works: 

- Suppose a singular $A$ satisfies $|A_{ij} - N_{ij}^{-1}| \leq \alpha = \frac{1}{4k + (2m+1)n - 2m}$

- There is a $t \in [-1, +1]^n$, $z \in \{-1, +1\}^n$: $N^{-1} - \alpha t z^T$ is singular.

- There is a $y \in \{-1, +1\}^n$, $0 < \beta \leq 1$ such that $N^{-1} - \alpha \beta y z^T$ is singular.

- $\max_{y,z \in \{-1, +1\}^n} z^T N y$ is achieved at $z = y$.

- $\frac{1}{\alpha \beta} N N^{-1} - N y z^T$ is singular; so $z^T N y = \frac{1}{\alpha \beta} \geq \frac{1}{\alpha}$, so $\exists u \in \{-1, +1\}^n : u^T N u \geq \frac{1}{\alpha}$, and the corresponding cut has size at least $k$. 
Radius for smaller rank

Instance: A rational square matrix \( A \), a rational \( \theta \), an integer \( r \).

Decide: Can the rank of \( A \) be brought down to below \( r \) by changing each entry by at most \( \theta \)?

Complexity is still unknown. In fact, it is not even known to be decidable!
Thank You