A clique in an undirected graph is a subgraph, where every two nodes are connected by an edge.

A $k$-clique is a clique that contains $k$ nodes.

The Clique problem is to determine whether a graph contains a clique of a specified size.

CLIQUE = \{\langle G, k \rangle | G is an undirected graph with a \( k \)-clique of specified size\}

Theorem 7.24: CLIQUE is in NP.

Proof Idea: The clique is the certificate.
• An NTM \( N \) = "On input \( \langle G, k \rangle \):
  1. Nondeterministically select a subset \( c \) of \( k \) nodes of \( G \)
  2. Test whether \( G \) contains all edges connecting nodes in \( c \)
  3. If both pass, accept; otherwise, reject"

- Remember, reject means to reject this computation path
- Some other computation path might accept the input
- If no computation path accepts, then input is not in CLIQUE

• Note the similarity between the two proofs
- Any problem which you can come up with a certificate, you could just as easily nondeterministically guess the certificate

Alternate Proof

Proof

• Let the certificate be the nodes in the clique
  - Just have to show that a TM can use certificate to decide whether \( G \) has a clique of size \( k \)
  1. Test whether \( c \) is a set of nodes in \( G \)
  2. Test whether \( G \) contains all edges connecting nodes in \( c \)
  3. If both pass, accept; otherwise, reject

\( V \) runs in polynomial time in size of \( \langle G, k \rangle \)

- Remember, if we reject, it means that \( c \) is not a certificate
- Input \( \langle G, k \rangle \) is in CLIQUE if there is some certificate
- Input \( \langle G, k \rangle \) is not in CLIQUE if there is no certificate
The following is a verifier for \textsc{Subset-Sum}.

\begin{itemize}
  \item On input $\langle \langle S, t \rangle, c \rangle$:
    \begin{enumerate}
      \item Test whether $c$ is a collection of numbers that sum to $t$
      \item Test whether $S$ contains all numbers in $c$
    \end{enumerate}
  \end{itemize}

\begin{itemize}
  \item If both tests pass, accept; otherwise, reject
  \end{itemize}

Alternate Proof: An NTM $N = \langle \langle S, t \rangle \rangle$:

\begin{itemize}
  \item On input $\langle S, t \rangle$:
    \begin{enumerate}
      \item Nondeterministically select a subset $c$ of $S$
      \item Test whether $c$ sums to $t$
    \end{enumerate}
  \end{itemize}

\begin{itemize}
  \item If test passes, accept; otherwise, reject.
  \end{itemize}

The following is a verifier for \textsc{Subset-Sum}.

\textbf{Proof}

Theorem 7.25: \textsc{Subset-Sum} is in \textsc{NP}

\begin{itemize}
  \item Proof Idea: the subset is the certificate
  \end{itemize}

\section*{Theorem 7.25: \textsc{Subset-Sum} is in \textsc{NP}}

\begin{itemize}
  \item Note that these can be \textsc{multisets} and so allow repetition
  \end{itemize}

\begin{itemize}
  \item Example: $\langle \{4, 11, 16, 21, 27\} \rangle$
  \end{itemize}

\begin{itemize}
  \item We have $\langle \{i, \ldots, i\} \rangle = S | \{ i \}$
  \end{itemize}

\begin{itemize}
  \item Given a collection of numbers, and a target number, is there a
  \end{itemize}

\begin{itemize}
  \item Subset that adds up to the target?
  \end{itemize}

\textbf{Subset-Sum}
Complements

• What about CLIQUE and SUBSET-SUM?
  - Not obvious whether these are in NP
  - Verifying that something is not present seems to be more difficult than
    verifying that something is present
  - No one knows whether coNP is different from NP

Definition: coNP are the languages whose complements are in NP.

Overview

• Problems in NP
  ⇒ Complexity Classes
  • NP-Completeness
  • Cook-Levin
  • Additional NP-Complete Problems

- coNP: is the language whose complements are in NP.
- If there is a NTM N to accept L, then there is a NTM N' that accepts L' for all L.
- If there is a NTM C that accepts L, then there is a NTM C' that rejects L' for all L.
- Switching accept and reject of N does not result in C (homework)
NP problems can be decided in deterministic exponential time. Proved this by complexity of simulating the nondeterminism.

\[ \text{NP} \subseteq \text{EXPTIME} = \bigcup_k \text{TIME}(2^{n^k}) \]

We can also prove \( \text{coNP} \subseteq \text{EXPTIME} \). If \( A \in \text{coNP} \), then \( A \in \text{NP} \) and is decided by NTM \( N \). There is a deterministic TM \( N' \) that decides \( A \) in exponential time.

Construct \( M' \) by switching \( M \)'s accept and reject states. \( M' \) runs in exponential time and decides \( A \).

Note that switching accept and reject just works for a DTM to have it accept the complement of the language and decide \( A \).

If \( A \in \text{coNP} \), then \( \forall L \in \text{NP} \), \( L \) is decided by \( M \).

- We can also prove \( \text{coNP} \subseteq \text{EXPTIME} \).
- If \( A \in \text{coNP} \), then \( A \in \text{NP} \) and is decided by \( M \).
- This doesn't work for a NTM. If it did then \( \text{coNP} = \text{NP} \).

\[ \text{NP} \subseteq \text{EXPTIME} \]

Most researchers believe they are not equal.

- \( \text{NP} \) is the class of languages where membership can be decided in polynomial time.
- \( \text{P} \) is the class of languages where membership can be decided in polynomial time.
- \( \text{P} \subseteq \text{NP} \).
- \( \text{P} = \text{NP} \) if and only if \( \forall L \in \text{NP} \), \( L \) is decided by a polynomial-time TM.
- \( \text{P} \neq \text{NP} \) if and only if \( \exists L \in \text{NP} \), \( L \) is not decided by a polynomial-time TM.
- \( \text{P} \neq \text{NP} \) if and only if \( \exists L \in \text{NP} \), \( L \) is not decided in polynomial time.
- \( \text{P} \neq \text{NP} \) if and only if there exists a language in \( \text{NP} \) that is not in \( \text{P} \).
Motivation

• We don't know if P = NP
  - So, we do not know if there is a problem in NP that is not in P
  - How can we get shed light on this?
  - Are there problems that capture how difficult NP is?

• On theoretical side
  - To show P is equal to NP, show an NP-complete problem is in P
  - To show P is not equal to NP, show an NP-complete problem is not in P

• On Practical side
  - If you can show your problem is NP-complete, don't bother looking for a polynomial algorithm

Overview

• Additional NP-complete Problems
  - Cook-Levin
  - NP-completeness
  - Complexity Classes
  - Problems in NP
Definitions

Definition 7.28:
A function $f : \Sigma^* \rightarrow \Sigma^*$ is a polynomial time computable function if some polynomial time Turing machine $M$, on every input $w$ halts with just $f(w)$ on its tape.

Definition 7.29:
Language $A$ is polynomial time mapping reducible, or simply polynomial time reducible, to language $B$, written $A \leq_p B$, if there is a polynomial time computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every $w$, $w \in A \iff f(w) \in B$.

Function $f$ is called the polynomial time reduction of $A$ to $B$.

An NP-complete problem

• Satisfiability problem for boolean formula - a bunch of boolean variables, which can be 'true' or 'false'.
• boolean operations: 'and' $\land$, 'or' $\lor$, 'not' $\neg$.
• boolean formula is an expression involving boolean variables and operations. $\phi$ is a satisfiable boolean formula if some assignment of 1 and 0 to variables makes the formula true.
• SAT = \{ $\langle \phi \rangle$ | $\phi$ is a satisfiable boolean formula \}

Cook-Levin Theorem: SAT $\in$ NP $\iff$ P = NP

So, SAT is an NP-complete problem.

An NP-complete problem

• A boolean formula is an expression involving boolean variables and boolean operations: 'and', 'or', 'not'.
• A bunch of boolean variables, which can be true, false. 0
• satisfiability problem for boolean formula
Proof

Let \( M \) be a polynomial time algorithm deciding \( B \) and \( f \) be the polynomial time reduction from \( A \) to \( B \).

We describe the polynomial time algorithm \( N \) for deciding \( A \):

- \( \:\text{On input } w \):
  1. Compute \( f(w) \)
  2. Run \( M \) on input \( f(w) \) and output whatever \( M \) outputs

Does \( N \) decide \( A \)?

- \( N \) accepts \( w \) whenever \( M \) accepts \( f(w) \) (line 2)
- \( M \) accepts \( f(w) \) whenever \( f(w) \in B \) (since \( M \) is a decider of \( B \))
- \( f(w) \in B \) whenever \( w \in A \) (since \( f \) is a reduction)

So \( N \) accepts \( w \) whenever \( w \in A \)

\( N \) runs in polynomial time as does \( f \) and so does \( N \).

Theorem 7.31

If \( A \leq_P B \) and \( B \in P \), then \( A \in P \).
Theorem 7.32: 3SAT is polynomial time reducible to CLIQUE

Proof idea:
- Convert formulas to graphs
- Structures within the graph are designed to mimic the behavior of each variable and of each clause

Definitions
- A literal is a boolean variable or a negated boolean variable: \( x \) or \( \neg x \)
- A clause is several literals connected with \( \lor \)'s
- A boolean formula is in conjunctive normal form, called a cnf-formula, if it comprises several clauses connected with \( \land \)
- A boolean formula in conjunctive normal form, called a 3cnf-formula, if all the clauses have three literals

3SAT = \{ \phi \mid \exists \phi \}

3SAT vs. CLIQUE

Let's practice using polynomial time reducibility
Is it a mapping reduction?

- Any variable not specified, truth assignment does not matter
- All literals are negations of each other, so any assignment is consistent
- Since no edges between negations of literals, clique does not contain
  - Any literal from each clause: which makes clause true
  - Since no edges between literals in same clause, clique must contain + Positive literal makes entire clause make entire clause true
  - No edges between negations of literals, so clique does not contain
  - Any variables not specified, truth assignment does not matter

If there is a clique of size \( k \):

- So there is a clique of size \( k \)
- There are no two literals from same clause
  + None of the picked literals will be negations of each other
  + No edge between picked literals will be negations of each other
- There will be an edge between each node because
- Pick one literal from each clause to be in clique (will be at least one)
- If a formula has a satisfying assignment

If there is a clique of size \( k \):

- Look for a clique of size \( k \)
  - Except between literals that are negations of each other
- For each node, add an edge to every other node not in its clause
  - That way both nodes cannot be included in a clique
  - No edges between literals that are negations of each other
- Clique will indicate which literals are true
- One node for each literal in each clause (\( 3k \) nodes)

- How do we map this to a graph?
  - Truth value of variable must be consistent
  - Each of the \( k \) clauses must have at least one literal that is true
  - What is the essence of 3SAT?

\[
(\exists x \wedge \exists y \wedge \exists z)(x \wedge (\exists x \wedge \exists y \wedge \exists z) (\wedge (\exists x \wedge \exists y \wedge \exists z) (\wedge y) \wedge y) \wedge y)\]

How to convert formulas to graphs
**Definition 7.34:** A language $B$ is NP-complete if it satisfies two conditions:
1. $B$ is in NP
2. Every $A$ in NP is polynomial time reducible to $B$

**Theorem 7.35:** If $B$ is NP-complete and $B \in P$, then $P = NP$. 

---

**Polynomial Time Reduction?**

- Yes
- No if $CLIQUE$ is solvable in polynomial time, so is $3SAT$
- Can construct the graph from the formula very easily
- At first glance, seems quite remarkable as they are very different problems
Another Theorem

Theorem 7.36: If $B$ is NP-complete and $B \leq P_C$ for $C$ in NP, then $C$ is NP-complete.

Proof - Let $A \in NP$. Since $B$ is NP-complete, there is a polynomial time mapping reduction $f_{AB}$ from $A$ to $B$. Since $B \leq P_C$, there is a polynomial time mapping reduction $f_{BC}$ from $B$ to $C$. Let $f(w) = f_{BC}(f_{AB}(w))$. This will be a mapping reduction from $A$ to $C$. Running two polynomial time algorithms in a row is still polynomial time. So $f$ is a polynomial time reduction from $A$ to $C$. Since $C$ is in NP, and for every $A \in NP$, there is a polynomial time reduction to $C$, $C$ is NP-complete.

Proof. • Let $A$ be any language in NP. • Since $B$ is NP-complete, $A \leq P_B$. So can convert a problem in $A$ to a problem in $B$ in polynomial time. • Since $B \in P$, $B$ can be decided in polynomial time. So a machine that first converts problem in $A$ to one in $B$, and then decides the problem in $B$ can decide any language in NP. Since $B \in NP$, $B$ is NP-complete. Since $B \leq P_C$, $C$ is NP-complete. So $B \leq P_C$, $C$ is NP-complete.

Proof. • We already know that $P \subseteq NP$. So $P = NP$.

So $P \subseteq NP$. So every language in NP is in $P$. Both steps can be done in polynomial time. So a machine that first converts problem in $A$ to a problem in $B$, and then decides the problem in $B$ can decide any language in NP. Since $B \in NP$, $B$ is NP-complete.
The Quest for an NP-Complete Problem

- We now know that:
  + there are problems in NP that capture how difficult NP is
  + If $P \neq NP$, NP-complete problems will not be in P once we have one NP-complete problem, we can use polynomial reductions to show other problems are NP-complete
  + If $P \neq NP$ is NP-complete, so is $NP$-
  - $3SAT$ is NP-complete

- Proof of Condition 1:
  - $SAT$ is polynomial-time verifiable
  + Certificate is truth assignment of variables
  + $SAT$ is polynomial-time verifiable
  + From previous theorems, same as: $SAT$ is NP-complete
  - Cook-Levin Theorem: $SAT \in P$ iff $P = NP$

But how do we prove the first one?

We now know that:

### Overview

- Problems in NP
- Complexity Classes
- NP-Completeness
- Additional NP-Complete Problems
  - Cook-Levin
  - NP-completeness
  - Complexity Classes
  - Problems in NP
Reduction from $w$ to $\phi$

Let $C = Q \cup \Gamma \cup \{\#\}$ - states, tape alphabet, begin/end symbol to mark computation

Each of the $\binom{n}{k}^2$ cells in table are called a cell -

Each cell $(i, j) \in C$ -

Variable for each possible value in each cell: $x_{i,j,s}$ is true if cell $(j, j)$ = $s$

$\phi$ composed of 4 parts -

Variable for each possible value in each cell: one clause with $|C|$ literals

+ At least one value is true in cell: one clause with $|C|$ literals with variables $\in C$

+ At most one value is true in cell: one clause with exactly one variable $\in C$

For each cell make sure exactly one value is true

- For each row is the start configuration: start state and input $w$

- Each cell $(i, j) \in C$ is true if cell $(i, j)$ = $s$

- Each cell $(i, j) \in C$ is false if cell $(i, j)$ = $t$ for all $t \in \Gamma$

- Each of the cells in table are called a cell

- States, input alphabet, beginning symbol to mark computation

Each row should follow previous according to $N$'s transition function -

Think of all the configurations as rows in a table -

Accepting tableau if one of the rows is an accepting configuration -

Let $C = \{\#\}$ \cup \Gamma \cup \emptyset

Second Condition

Let $A$ be a language in NP -

Need to show that $A \leq_P \text{SAT}$ -

Need to show that some polynomial time reduction exists -

All that we know about $A$ is that it can be decided by an NTM. Say $N$

All that we need to show is that $f$ exists, don't have to give actual $f$

For input $w$ of length $n$, will be of length at most $n^k$ for some $k$

Tell me, do I need to know what value $k$ is?

So, we don't need to know what value $k$ is.

All that we need to show is that $f$ exists, don't have to give actual $f$

For input $w$ of length $n$, will be of length at most $n^k$ for some $k$

Tell me, do I know what $N$ will have an accepting computation?

Tell me, do I know what $N$ will have an accepting computation?

Tell me, do I know that $N$ is true?

Tell me, do I know that $N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?

$N$ is true?
Corollary 7.42: 3SAT is NP-Complete

Previous proof can be altered to produce a formula in 3SAT.

\[ \exists \phi \text{ composed of 4 parts} \]

- There is an accepting configuration \[ (q_2, u_1) \]
  - One cell is in the accept state
- Make sure each configuration legally follows previous one
  \[ \forall x, y \in \{0, 1\} \]
- Similar to what we did with PCP
  \[ \exists \phi \text{ composed of } 4 \text{ parts} \]

Size of formula will be polynomial in size of input:
\[ O(n^2 k) \]

- Has a very repetitive structure, so we can generate it in time
- Legal windows for \[ (q_2, u_1) \]
  - Create clauses to capture how each \[ 3 \times 2 \] set of cells can change
- Make sure each configuration legally follows previous one
- There is an accepting configuration
  \[ \exists \phi \text{ composed of } 4 \text{ parts} \]

Continued
NP-Complete problems

- SAT is NP-complete
- 3SAT is NP-complete
- CNF-SAT is NP-complete
- CLIQUE is NP-complete
- HAMPATH is NP-complete
- SUBSET-SUM is NP-complete

Overview

- Problems in NP
- Complexity Classes
- NP-Completeness
- Cook-Levin
- Additional NP-Complete Problems