Overview

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

k-Clique

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

$CLIQUE = \{ \langle G, k \rangle | G \text{ is an undirected graph with a } k\text{-clique} \}$

**Theorem 7.24:** $CLIQUE$ is in NP.
• Proof Idea: The clique is the certificate
Alternate Proof

• An NTM $N = \langle G, k \rangle$:
  1. Nondeterministically guess 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:
- If we reject, it means that $c$ is not a certificate
- Input $\langle G, k \rangle$ is in CLIQUE if there is some certificate $c$
- Input $\langle G, k \rangle$ is not in CLIQUE if there is no certificate

Proof

Let the certificate be the nodes in the clique $c$.

$V = \langle G, k \rangle$:
  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

The following is a verifier for CLIQUE:

$\gamma(V) = L$
Subset-Sum

Given a collection of numbers, and a target number, is there a subset that adds up to the target?

\[ \text{SUBSET-SUM} = \{ \langle S, t \rangle | S = \{x_1, \ldots, x_k\} \text{ and for some } \{y_1, \ldots, y_l\} \subseteq S, \text{ we have } \sum y_i = t \} \]

- Example: \( \langle \{4, 11, 16, 21, 27\}, 25 \rangle \)
- Note that these can be multisets and so allow repetition

**Theorem 7.25**: \( \text{SUBSET-SUM} \) is in NP

- Proof Idea: the subset is the certificate
Proof

• The following is a verifier for $\text{SUBSET-SUM}$.  
  $V = \text{"On input } \langle \langle S, t \rangle, c \rangle: \text{"}$  
  1. Test whether $c$ is a collection of numbers that sum to $t$  
  2. Test whether $S$ contains all numbers in $c$  
  3. If both pass, accept; otherwise, reject"  

• Alternate Proof: An NTM $N =$  
  "On input $\langle S, t \rangle$:  
  1. Nondeterministically select a subset $c$ of $S$  
  2. Test whether $c$ sums to $t$  
  3. If test passes, accept; otherwise, reject"

Overview

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

• P is class of languages where membership can be decided in polynomial time
• NP is the class of languages where membership can be decided in polynomial time on a nondeterministic TM
• But is P = NP?
  - Have not been able to find any languages in NP that are provable to not be in P!!
  - If equal, any problem solvable in nondeterministic polynomial time are solving in polynomial time!!
  - Most researchers believe they are not equal

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 then verifying it is present

**Definition:** coNP are languages whose complement is in NP.

• Unknown if coNP different from NP (i.e., closed under complementation)
  - \( L \in NP \) if there is a NTN \( N \) that is a decider s.t. \( w \in L \) iff \( N \) has at least one computation path that will accept
  - \( L \in coNP \) if there is a NTN \( C \) that is a decider s.t. \( w \in L \) iff all computation paths reject
  - Switching accept and reject of \( N \) does not result in \( C \) (HW question)
Final Point on coNP

- Both NP and coNP are in EXPTIME
  - Not equivalent, but very similar
  - NP is a bit nicer as NP languages have a verifier
    + You can give an existence proof for it
    + No existence proof possible for coNP

EXPTIME

- NP problems can be decided in deterministic exponential time
  - Proved this by complexity of simulating the nondeterminism

NP ⊆ EXPTIME = ∪_k TIME(2^{n^k})

- We can also prove coNP ⊆ EXPTIME
  - If \( A \in \text{coNP} \), \( \overline{A} \in \text{NP} \) and is decided by NTM \( N \)
  - There is a deterministic TM \( M \) that decides \( \overline{A} \) in exponential time
  - Construct \( M' \) by switching \( M \)'s accept and reject state
  - \( M' \) 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
  - This doesn’t work for a NTM. If it did then coNP = NP
  - So, P is closed under complementation. \( \text{NP} \) is probably not
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 shed light on this?
  - Are there problems that capture how difficult NP is
    + If such a problem is solvable in P then P=NP
  - Such problems will be called **NP-complete**

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

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

• satisfiability problem for boolean formula
  - a bunch of boolean variables, which can be ‘true’ 1, or ‘false’ 0
  - boolean operations: ‘and’ ∧, ‘or’ ∨, ‘not’ ¬ (¬x also written as ¯x)
  + defined by truth tables
  - boolean formula is an expression involving boolean variables and operations. e.g. φ = (x∧y) ∨ (x∧¯x)
  - a boolean formula is satisfiable if some assignment of 1 and 0 to variables makes the formula true (1)
  + what assignment makes above formula true?
  $SAT = \{\langle \phi \rangle | \phi \text{ is a satisfiable Boolean formula}\}$

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

• So, $SAT$ is an NP-complete problem!
  - Will take a bit of effort to prove this!

Definitions

• In chapter 5, we defined mapping reducibility
  - Now define a version that takes time into account

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 a polynomial time reduction of $A$ to $B$. 
Polyomial Time Reducibility

- As with ordinary mapping reduction, polynomial time reduction of $A$ to be $B$ provides a way to convert membership testing in $A$ to membership testing in $B$
  - To test whether $w \in A$, we use reduction $f$ to map $w$ to $f(w)$ and test whether $f(w) \in B$
  - But, where conversion is done efficiently

**Theorem 7.31:** If $A \leq_P B$ and $B \in P$, then $A \in P$

- Compare with Chapter 5:
  **Theorem 5.22:** If $A \leq_m B$ and $B$ is decidable, then $A$ is decidable.

---

**Proof**

- If $A \leq_P B$ and $B \in P$, then $A \in P$
  - 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 $A$
    $N =$ “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 $f$ does and so does $M$
3SAT

- Let’s practice using polynomial time reducibility
  - But on a simpler problem than SAT

- Definitions
  - A literal is a boolean variable or a negated boolean variable: \( x \) or \( \overline{x} \)
  - 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 \( \land \)
    + Aside: Any boolean expression can be written in conjunctive normal form
  - It is 3cnf-formula if all the clauses have three literals
    \((x_1 \lor x_2 \lor x_3) \land (x_3 \lor x_5 \lor x_6) \land (x_3 \lor x_6 \lor x_4) \land (x_4 \lor x_5 \lor x_6)\)

\[3SAT = \{ \langle \phi \rangle \mid \phi \text{ is a satisfiable 3cnf-formula} \}\]

3SAT and CLIQUE

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
How to convert formulas to graphs

\[(x_1 \lor \overline{x_2} \lor \overline{x_3}) \land (x_3 \lor \overline{x_5} \lor x_6) \land (x_3 \lor \overline{x_6} \lor x_4)\]

• What is the essence of 3SAT?
  - Each of the \(k\) clauses must have at least one literal that is true
  - Truth value of variables must be consistent

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

Is it a mapping reduction?

• If a formula has a satisfying assignment
  - Pick one true literal from each clause to be in subset (will be at least one)
  - There will be an edge between each node because
    + None of the picked literals will be negations of each other
    + There are not two literals from same clause
  - So there is a clique of size \(k\)

• If there is a clique of size \(k\)
  - Make all of the literals in the clique to be true
    + Positive literal makes var true, negative literal makes var false
  - Since no edges between literals in same clause, clique must contain exactly one literal from each clause; which makes clause true
  - Since no edges between negations of literals, clique does not contain literals that are negations of each other; so truth assignment is consistent
  - Any variables not specified, truth assignment does not matter
Polynomial Time Reduction?

- Yes
  - Can construct the graph from the formula very easily
- So if CLIQUE is solvable in polynomial time, so is 3SAT
  - At first glance, seems quite remarkable as they are very different problems

Definition

**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 $P = NP$. 
Theorem 7.36: If $B$ is NP-complete and $B \leq_p C$ for $C$ in NP, then $C$ is NP-complete

Proof

- Let $A$ 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, make a machine that first converts problem in $A$ to one in $B$, and then decides the problem in $B$
- Both steps can be done in polynomial time
- So $A \in P$. So every language in NP is in P.
- So NP $\subseteq$ P
- We already know that P $\subseteq$ NP. So, P = NP

**Another Theorem**

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

- 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 $f(w)$ be $f_{BC}(f_{AB}(w))$
  + This will be a mapping reduction from $A$ to $C$
  + Running two polynomial time algorithms in a row is still polynomial
  + So $f$ is a polynomial time reduction from $A$ to $C$
- Since $C$ is in NP, and for every $A \in$ NP, there is polynomial time reduction to $C$, $C$ 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 3SAT is NP-complete, so is CLIQUE

• But how do we prove the first one?
  - Cook-Levin Theorem: SAT $\in P$ iff $P = NP$
  - From previous theorems, same as: SAT is NP-complete

• Proof of Condition 1:
  - SAT is polynomial-time verifiable
    + Certificate is truth assignment of variables
    + Can be verified in polynomial time
Second Condition

- Let $A$ be a language in $\text{NP}$
  - 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 $\text{NTM}$, say $N$
  - But, we do know that $N$ will have an accepting computation
    - For input $w$ of length $n$, will be of length at most $n^k$ for some $k$
    - All that we need to show is that $f$ exists, don’t have to give actual $f$
    - So, we don’t need to know what value $k$ is
  - Think of all the configurations as rows in a table
    - $n^k$ configurations (rows) each $n^k$ long (columns)
    - Each row should follow previous according to $N$’s transition function
    - Accepting tableau if one of the rows is an accepting configuration

Reduction from $w$ to $\phi$

- Let $C = Q \cup \Gamma \cup \{\#\}$
  - states, tape alphabet, begin/end symbol to mark computation
- Each of the $(n^k)^2$ 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
  - For each cell, make sure exactly one value is true
    - At least one value is true in cell: one clause with $|C|$ literals
    - $x_{i,j,s_1} \lor x_{i,j,s_2} \lor \ldots \lor x_{i,j,s_m}$
    - At most one value is true in cell: $|C|^2$ clauses for each pair of values $s, t$
    - $x_{i,j,s} \land \neg x_{i,j,t}$
  - Make sure first row is the start configuration: start state and input $w$
    - $x_{1,1,#} \land x_{1,2,#} \land w_1, w_2 \land \ldots \land w_1, n+2, w_n \land w_1, n+3, \# \land \ldots \land w_1, n^k, \#$
Continued

- There is an accepting configuration
  + One of the cells is the accept state
  \[ x_{1,1}q_{accept} \lor x_{1,2}q_{accept} \lor \ldots \lor x_{n^2,n^2}q_{accept} \]
  - Make sure each configuration row legally follows previous row
    + Create clauses to capture how each 3x2 set of cells can change
    + Where there is no change (for any \( a, b, c \in \Gamma \)):
      \[
      \begin{array}{ccc}
        a & b & c \\
        a & b & c \\
      \end{array}
      \]
    + Legal windows for \((q_2, a, L) \in \delta(q_1, b)\) (for any \( a, b, c, d \in \Gamma \)):
      \[
      \begin{array}{ccc}
        d & c & a \\
        d & c & a \\
      \end{array}
      \]
  + Similar to what we did with PCP

- Size of formula will be polynomial in size of input: \(O(n^{2k})\)
  - Has a very repetitive structure, so we can generate it in time \(O(n^{2k})\)

PCP versus SAT

- Used accepting computation histories of a TM on \(w\) in a reduction to prove
  - PCP is undecidable
  - SAT is NP-hard
  - Seems contradictory

- Showed acceptance of a TM on \(w\) can be encoded in PCP
  - Tiles encode legal moves
  - There is an accepting configuration of TM on \(w\) iff there is a solution to corresponding PCP problem
  - Since \(A_{TM}\) is not decidable, PCP is not decidable

- Here: Showed acceptance of a TM on \(w\) can be encoded in SAT
  - Does this mean that SAT is undecidable?
3SAT

Corollary 7.42: 3SAT is NP-Complete

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

Overview

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