Today:
− NP-Completeness (con’t.)
Reading Assignments

• Today’s class:
  – Chapter 34 (con’t.)
Recall: NP-Completeness

- A problem $B$ is **NP-complete** if:
  1. $B \in \text{NP}$
  2. $A \leq_p B$ for all $A \in \text{NP}$

- If $B$ satisfies only property (2) we say that $B$ is **NP-hard**

- No polynomial time algorithm has been discovered for an **NP-Complete** problem

- No one has ever proven that no polynomial time algorithm can exist for any **NP-Complete** problem

- **Significance:** If one NP-Complete problem can be solved in poly-time, then all NP problems can be solved in poly-time
Recall: We always cast NP-Complete problems as decision problems

• Decision problems
  – Given an input and a question regarding a problem, determine if the answer is yes or no

• Optimization problems
  – Find a solution with the “best” value

• Interesting question:
  – Let’s presume that someone (amazingly) proves that P=NP.
  – But, all the NP-complete (NPC) problems are expressed as decision problems.
  – So, if P=NP, how can we make use of the poly-time algorithm that solves an NPC decision problem to also solve the optimization version of the same problem in poly-time?
Example: Using Poly-time alg. for decision problem to solve optimization problem in poly-time

Example: Show that if P = NP, then there is a polynomial time algorithm that, given a Boolean formula $\phi$, actually produces a satisfying assignment for $\phi$ (assuming $\phi$ is satisfiable).
Recall: Polynomial Reductions

- Reduction is a way of saying that one problem is **no harder** than another.
- We say that problem A is no harder than problem B, (i.e., we write “$A \leq_p B$”) if we can solve A using the algorithm that solves B.
- **Idea:** transform the inputs of A to inputs of B in poly time
Recall: Polynomial Reductions

Given two problems A, B, we say that A is polynomially reducible to B \((A \leq_p B)\) if:

1. There exists a function \(f\) that converts the input of A to inputs of B in polynomial time

2. \(A(\alpha) = YES \iff B(f(\alpha)) = YES\)
Proving NP-Completeness In Practice

1) Prove that the new problem $B \in \text{NP}$

2) Show that one known NP-Complete problem, $A$, can be transformed to $B$ in polynomial time (i.e., $A \leq_p B$)

• Conclude that $B$ is NP-Complete
Once one problem (SAT) shown to be NP-Complete, can show many others...

Example reductions (From CLRS, Ch. 34):
NP-Complete Problem: 
Circuit-SAT

- **Circuit-SAT problem**: Given a boolean combinational circuit, C, determine if there is a satisfying assignment to inputs such that the circuit’s output is 1.
- CLRS proves this is NP-Complete (more next week)

Example circuit with satisfying assignment
NP-Complete Problem: Satisfiability (SAT)

- **Satisfiability problem**: Given a logical expression $\phi$, find an assignment of values (F, T) to variables $x_i$ that causes $\phi$ to evaluate to T:

$$\phi = x_1 \lor \neg x_2 \land x_3 \lor \neg x_4$$

- SAT was the historically first problem shown to be NP-complete

- Here, we’ll presume CIRCUIT-SAT is known NP-Complete (per CLRS), and prove SAT is NP-Complete by reduction
Prove SAT is NP-complete

• **Step 1:** SAT $\in$ NP
  - Argue that, given a certificate, you can verify that the certificate provides a solution in polynomial time

• **Step 2:** Show that some known NP-Complete problem is reducible in poly-time to SAT (i.e., $A \leq_p SAT$)
  - What known NP-Complete problem do we choose?
Show \( \text{Circuit-SAT} \leq_p \text{SAT} \)

- What do we have to do?
  1) Given an instance \(<C>\) of Circuit-SAT, define poly-time function \(f\) that converts \(<C>\) to instance \(<\phi>\) of SAT
  2) Argue that \(f\) is poly-time
  3) Argue that \(f\) is correct (i.e., \(<C>\) of Circuit-SAT is satisfiable iff \(<\phi>\) of SAT is satisfiable)

- Here’s a proposed poly-time reduction, \(f\):
  - For every wire \(x_i\) of \(C\), define a variable \(x_i\) in the formula.
  - Every gate can be expressed as:
    \[ x_i \leftrightarrow \text{(boolean operations consistent with gate)} \]
  - The final formula \(\phi\) is the conjunction (AND) of the circuit output variable and conjunction of all clauses describing the operation of each gate.
Example of reduction of Circuit-SAT to SAT

Here’s an input instance \(<C>\) if Circuit-SAT:

$$\phi = x_{10} \land (x_{10} \leftrightarrow (x_7 \land x_8 \land x_9)) \land (x_9 \leftrightarrow (x_6 \lor x_7)) \land (x_8 \leftrightarrow (x_5 \lor x_6)) \land (x_7 \leftrightarrow (x_1 \land x_2 \land x_4)) \land (x_6 \leftrightarrow \neg x_4)) \land (x_5 \leftrightarrow (x_1 \lor x_2)) \land (x_4 \leftrightarrow \neg x_3)$$

Here’s the result of \(f(C)\), which gives instance \(<\phi>\) of SAT:
Next, prove properties of $f$

• Argue that $f$ is poly-time:
  – Obvious -- Clearly the reduction can be done in poly time

• Argue that $f$ is correct:
  – C is satisfiable if and only if $\phi$ is satisfiable:
    • $\Rightarrow$ If $C$ is satisfiable, then there is a satisfying assignment. This means that each wire of $C$ has a well-defined value and the output of $C$ is 1. Thus, the assignment of wire values to variables in $\phi$ makes each clause in $\phi$ evaluate to 1. So $\phi$ is 1 when $C$ is satisfiable.
    • $\Leftarrow$ The reverse proof should also be done (i.e., if $\phi$ evaluates to 1, then $C$ must be satisfiable); proof mirrors the argument above.

• Since (1) SAT $\in$ NP, and (2) Circuit-SAT $\leq_p$ SAT, we conclude that SAT is NP-Complete.
Important note about reductions...

- Note that we never make use of the solution to a problem in creating reduction function $f$

- In the proof of correctness, we mention the solution of one problem helping us to get the solution of the other (if such a solution were known), based on our construction

- But, this solution is not used for the construction defined by $f$. Why?
  - We don’t know the solution, because finding it is an NP-complete problem.
  - Thus, our reduction function could not be polynomial-time if it required solving an NP-complete problem to create the construction.
  - The reduction function $f$ must therefore work for any possible instance of the known NP-complete problem, but without knowledge of the solution.
Another NP-Complete Problem: CNF Satisfiability

- CNF is a special case of SAT
- \( \phi \) is in “Conjuctive Normal Form” (CNF)
  - “AND” of expressions (i.e., clauses)
  - Each clause contains only “OR”s of the variables and their complements

\[ E.g.: \phi = (x_1 \lor x_2) \land (x_1 \lor \neg x_2) \land (\neg x_1 \lor \neg x_2) \]
Another NP-Complete Problem: 3-CNFM Satisfiability

3-CNFM is a subcase of CNFM problem:
- A literal in a boolean formula is an occurrence of a variable or its negation.
- CNFM (Conjunctive Normal Form) is a boolean formula expressed as AND of clauses, each of which is the OR of one or more literals.
- 3-CNFM is a CNFM in which each clause has exactly 3 distinct literals (a literal and its negation are distinct)

• Example:

\[ \Phi = (x_1 \lor \neg x_1 \lor \neg x_2) \land (x_3 \lor x_2 \lor x_4) \land (\neg x_1 \lor \neg x_3 \lor \neg x_4) \]

• Let us prove that 3-CNFM is NP-Complete
Prove 3-CNF-SAT is NP-complete

• **Step 1:** 3-CNF-SAT $\in$ NP
  
  – Argue that, given a certificate, you can verify that the certificate provides a solution in polynomial time

• **Step 2:** Show that some known NP-Complete problem is reducible in poly-time to 3-CNF-SAT (i.e., $A \leq_p 3$-CNF-SAT)
  
  – What known NP-Complete problem do we choose?
SAT ≤ₚ 3-CNF-SAT

• What do we have to do?
  1) Given an instance < ϕ > of SAT, define poly-time function f that converts < ϕ > to instance < ϕ'''' > of SAT
  2) Argue that f is poly-time
  3) Argue that f is correct (i.e., < ϕ > of SAT is satisfiable iff < ϕ'''' > of 3-CNF-SAT is satisfiable)
SAT $\leq_p$ 3-CNF-SAT

• Proposed definition of $f$:
  • Suppose $\phi$ is any boolean formula, Construct a binary parse tree, with literals as leaves and connectives as internal nodes.
  • Introduce a variable $y_i$ for the output of each internal nodes.
  • Rewrite the formula to $\phi'$ as the AND of the root variable and a conjunction of clauses describing the operation of each node.
  • The result is that in $\phi'$, each clause has at most three literals.
  • Change each clause into conjunctive normal form as follows:
    – Construct a truth table
    – Write the disjunctive normal form for all truth-table items evaluating to 0
    – Using DeMorgans law to change to CNF.
  • The resulting $\phi''$ is in CNF but each clause has 3 or fewer literals.
  • Change 1 or 2-literal clauses into a 3-literal clause $\phi'''$ as follows:
    – If a clause has one literal $l$, change it to $(l \lor p \lor q) \land (l \lor p \lor \neg q) \land (l \lor \neg p \lor q) \land (l \lor \neg p \lor \neg q)$.
    – If a clause has two literals $(l_1 \lor l_2)$, change it to $(l_1 \lor l_2 \lor p) \land (l_1 \lor l_2 \lor \neg p)$. 
Example: Binary parse tree for:

\[ \phi = ((x_1 \rightarrow x_2) \lor \neg((\neg x_1 \leftrightarrow x_3) \lor x_4)) \land \neg x_2 \]

\[ \phi' = y_1 \land (y_1 \leftrightarrow (y_2 \land \neg x_2)) \]
\[ \land (y_2 \leftrightarrow (y_3 \lor y_4)) \]
\[ \land (y_4 \leftrightarrow \neg y_5) \]
\[ \land (y_3 \leftrightarrow (x_1 \rightarrow x_2)) \]
\[ \land (y_5 \leftrightarrow (y_6 \lor x_4)) \]
\[ \land (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3)) \]

Figure 34.11 The tree corresponding to the formula \( \phi = ((x_1 \rightarrow x_2) \lor \neg((\neg x_1 \leftrightarrow x_3) \lor x_4)) \land \neg x_2 \).
Example of Converting a 3-literal clause into CNF format

<table>
<thead>
<tr>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$x_2$</th>
<th>$(y_1 \leftrightarrow (y_2 \land \neg x_2))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Disjunctive Normal Form:
$$\phi'_i = (y_1 \land y_2 \land x_2) \lor (y_1 \land \neg y_2 \land x_2) \lor (y_1 \land \neg y_2 \land \neg x_2) \lor (\neg y_1 \land y_2 \land \neg x_2)$$

Conjunctive Normal Form:
$$\phi''_i = (\neg y_1 \lor \neg y_2 \lor \neg x_2) \land (\neg y_1 \lor y_2 \lor \neg x_2) \land (\neg y_1 \lor y_2 \lor x_2) \land (y_1 \lor \neg y_2 \lor x_2)$$

**Figure 34.12** The truth table for the clause $(y_1 \leftrightarrow (y_2 \land \neg x_2))$. 
3-CNF-SAT is NP-complete

Now, prove correctness of $f$:

- First, prove reduction is poly time:
  - From $\phi$ to $\phi'$, we introduce at most 1 variable and 1 clause per connective in $\phi$.
  - From $\phi'$ to $\phi''$, we introduce at most 8 clauses for each clause in $\phi'$.
  - From $\phi''$ to final 3-CNF, we introduce at most 4 clauses for each clause in $\phi''$.

- Then, prove reduction is correct – i.e., $\phi$ and resulting 3-CNF formula are equivalent:
  - From $\phi$ to $\phi'$, keep equivalence by construction.
  - From $\phi'$ to $\phi''$, keep equivalence by construction.
  - From $\phi''$ to final 3-CNF, keep equivalence by construction.

Since: (1) $3\text{-CNF-SAT} \in \text{NP}$, and (2) $\text{SAT} \leq_p 3\text{-CNF-SAT}$, we conclude that 3-CNF-SAT is NP-Complete.
Another NP-Complete Problem: Clique

Clique Problem:

– Given: undirected graph $G = (V, E)$
– Clique: a subset of vertices in $V' \subseteq V$, each pair of which is connected by an edge in $E$, i.e., a clique is a complete subgraph of $G$.
– Size of a clique: number of vertices it contains

Optimization problem:

– Find a clique of maximum size

Decision problem:

– Does $G$ have a clique of size $k$?
– Input instance = $<G, k>$
Prove Clique is NP-complete

- **Step 1:** Clique $\in$ NP
  - Argue that, given a certificate, you can verify that the certificate provides a solution in polynomial time

- **Step 2:** Show that some known NP-Complete problem is reducible in poly-time to Clique (i.e., $A \leq_p$ Clique)
  - What known NP-Complete problem do we choose?
3-CNF-SAT $\leq_p$ Clique

• What do we have to do?
  1) Given an instance $< \phi >$ of 3-CNF-SAT, define poly-time function $f$ that converts $< \phi >$ to instance $< G, k >$ of Clique
  2) Argue that $f$ is poly-time
  3) Argue that $f$ is correct (i.e., $< \phi >$ of 3-CNF-SAT is satisfiable iff $G$ has a Clique of size $k$)
3-CNF-SAT $\leq_p$ Clique

- Reduction function $f$ from 3-CNF-SAT to Clique:
  - Suppose $\phi = C_1 \land C_2 \land \ldots \land C_k$ is a boolean formula in 3-CNF form with $k$ clauses.
  - We construct a graph $G=(V,E)$ as follows:
    - For each clause $C_r = (l_1^r \lor l_2^r \lor l_3^r)$, place a triple of $v_1^r, v_2^r, v_3^r$ into $V$
    - Place an edge between two vertices $v_i^r$ and $v_j^s$ when:
      - $r \neq s$, that is $v_i^r$ and $v_j^s$ are in different triples, and
      - Their corresponding literals are consistent, i.e., $l_i^r$ is not negation of $l_j^s$.
  - Our resulting instance of Clique is $<G, k>$

- Then we argue that $\phi$ is satisfiable if and only if $G$ has a clique of size $k$. 
Example reduction from $\langle \phi \rangle$ to $\langle G, k \rangle$:

$$
\phi = (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_2 \lor x_3)
$$

$G$: $k = 3$
Clique is NP-Complete

Now, prove correctness of $f$:

- First, prove reduction is poly time:
  - Should be apparent – only create 3 variables per clause; do this $k$ times

- Then, prove reduction is correct – i.e., $\phi$ is satisfiable if and only if $G$ has a clique of size $k$:
  - $\Rightarrow$ If $\phi$ is satisfiable, then there exists a satisfying assignment that makes at least one literal in each clause evaluate to True. Pick one of these literals from each clause. Then consider the subgraph $V'$ consisting of the corresponding vertex of each such literal. For each pair, $v_i^r, v_j^s \in V'$, where $r \neq s$, since $l_i^r, l_j^s$ both evaluate to 1, and $l_i^r$ is not negation of $l_j^s$, then there must be an edge between $v_i^r$ and $v_j^s$. So $V'$ is a clique of size $k$.
  - $\Leftarrow$ If $G$ has a clique $V'$ of size $k$, then $V'$ contains exactly one vertex from each triple. Assign all the literals corresponding to the vertices in $V'$ to True, and other literals to either True or False (i.e., they don’t matter). Then each clause will evaluate to True. So $\phi$ is satisfiable.

Since: (1) Clique $\in$ NP, and (2) 3-CNF-SAT $\leq_p$ Clique, we conclude that Clique is NP-Complete.
Another NP-Complete Problem: Vertex Cover

Vertex Cover Problem:
- Undirected graph $G = (V, E)$
- **Vertex cover**: a subset $V' \subseteq V$ such that each edge in the graph is covered by some vertex in $V'$ (i.e., if $(u, v) \in E$, then $u \in V'$ or $v \in V'$ or both.)
- **Size of a VC**: number of vertices it contains

Optimization problem:
- Find a VC of maximum size

Decision problem:
- Does $G$ have a VC of size $k$?
- Instance of VC = $<G, k>$
Prove Vertex-Cover is NP-complete

• **Step 1:** Vertex-Cover $\in$ NP
  
  – Argue that, given a certificate, you can verify that the certificate provides a solution in polynomial time

• **Step 2:** Show that some known NP-Complete problem is reducible in poly-time to Vertex-Cover (i.e., $A \leq_p$ Vertex-Cover)
  
  – What known NP-Complete problem do we choose?
Clique $\leq_p$ Vertex-Cover

• What do we have to do?
  1) Given an instance $< G, k >$ of Clique, define poly-time function $f$ that converts $< G, k >$ to instance $< G', k' >$ of Vertex-Cover
  2) Argue that $f$ is poly-time
  3) Argue that $f$ is correct (i.e., $G$ has a clique of size $k$ iff $G'$ has a vertex cover of size $k$)
Clique \( \leq_p \) Vertex Cover

• Reduction \( f \), from Clique to Vertex Cover:
  – Convert \( G(V, E) \) to complement graph \( G'(V,E') \):
    • The edges \( E' \) of \( G' \) contain only those edges \textit{not} in \( E \)
  – Output vertex cover instance \(<G', |V|- k>\)

• Then we argue that \( G \) has a clique of size \( k \) iff \( G' \) has a vertex cover of size \(|V| - k\)
Proof of correctness of $f$

Now, prove correctness of $f$:

- First, prove reduction is poly time; straight-forward

Next, prove reduction is correct – i.e., $G$ has a clique of size $k$ iff $G'$ has a vertex cover of size $|V| - k$

- $\Rightarrow$ If $G$ has a clique of size $k$, $G'$ has a vertex cover of size $|V| - k$
  - Let $V'$ be the $k$-clique
  - Then $V - V'$ is a vertex cover in $G'$
    - Let $(u,v)$ be any edge in $G'$. Then $u$ and $v$ cannot both be in $V'$ (Why?)
    - Thus at least one of $u$ or $v$ is in $V - V'$ (why?), so edge $(u, v)$ is covered by $V - V'$
    - Since this is true for any edge in $G'$, $V - V'$ is a vertex cover.
Vertex-Cover is NP-Complete (con’t)

- \( \Longleftarrow \) If \( G' \) has a vertex cover \( V' \subseteq V \), with \( |V'| = |V| - k \), then \( G \) has a clique of size \( k \)
  - For all \( u,v \in V \), if \( (u,v) \in G' \) then \( u \in V' \) or \( v \in V' \) or both (\( why? \))
  - Contrapositive: if \( u \not\in V' \) and \( v \not\in V' \), then \( (u,v) \in E \)
  - In other words, all vertices in \( V-V' \) are connected by an edge, thus \( V-V' \) is a clique
  - Since \( |V| - |V'| = k \), the size of the clique is \( k \)
- Thus we conclude that \( G \) has a clique of size \( k \) iff \( G' \) has a vertex cover of size \( |V| - k \)

Since: (1) Vertex-Cover \( \in \) NP, and (2) Clique \( \leq_p \) Vertex-Cover, we conclude that Vertex-Cover is NP-Complete.
Reading Assignments

• Next class:
  – Chapter 34.3
    • Looking more deeply at reductions