NP-Complete Proofs

Analysis of Algorithms
Indirect Proofs

CIRCUIT-SAT was a direct proof. Since it was the first candidate to be shown to be NP-complete, a direct proof was necessary. However, we do not have to use a direct proof every time. Once we have a single NP-complete problem, we can use that one to show that other problems are also NP-complete.

Lemma:
If L is a language such that \( L' \leq_p L \) for some \( L' \in \text{NPC} \), then L is NP-hard. Moreover, if \( L \in \text{NP} \), then \( L \in \text{NPC} \).
Indirect Proofs

Keep in mind:

\[ \text{PROBLEM A} \leq_p \text{PROBLEM B} \] means:

Problem A is not any harder to solve than Problem B. Thus, B is \textit{at least as hard to solve as} A (maybe harder).

If we could solve B efficiently, then we would use the algorithm for B to get the solution for A. But if we know that A cannot be solved efficiently, then the algorithm for B cannot be efficient either.
Formulas Satisfiability (SAT)

An instance of SAT is a Boolean formula $\phi$ composed of:

1. N Boolean variables: $x_1, x_2, \ldots, x_N$
2. Boolean connectives such as $\land$ (AND), $\lor$ (OR), $\neg$ (NOT), $\rightarrow$ (implication), $\leftrightarrow$ (biconditional)
3. Parentheses

The satisfiability problem asks whether a given Boolean formula is satisfiable. For example, $\phi = ((x_1 \rightarrow x_2) \lor \neg((\neg x_1 \leftrightarrow x_3) \lor x_4)) \land \neg x_2$ has the satisfying assignment $(x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 1)$, since

\[
\phi = ((0 \rightarrow 0) \lor \neg((\neg0 \leftrightarrow 1) \lor 1)) \land \neg 0 \\
= (1 \lor \neg(1 \lor 1)) \land 1 \\
= (1 \lor 0) \land 1 \\
= 1,
\]
Formula Satisfiability (SAT)

Prove that SAT is NP-Complete:

1. SAT ∈ NP
2. CIRCUIT-SAT ≤^p SAT

1. Given a certificate (“witness”) consisting of a satisfying assignment, check if it is a solution by replacing each variable in the formula with the corresponding value from the certificate, and then evaluate the expression. If the output is true, then the formula is satisfiable, otherwise it is not.

2. Any instance of CIRCUIT-SAT can be reduced in polynomial time to a specific instance of SAT by converting the circuit C to a formula (How? See the next slide). C is satisfiable exactly when the formula is satisfiable. Why? The formula is the result of the assignment of wires in C to variables in the formula. If each clause in the formula is true, then the output is true.
Conclusion: If we were able to get satisfying solutions to Boolean formulas easily, then we could use that algorithm to easily assign truth values to digital circuits such that the output of the circuit is 1.
3-CNF-Satisfiability

- A Boolean expression in Conjunctive Normal Form (CNF) is a series of clauses combined with the AND operator
  \[(x_1 \lor \neg x_1 \lor \neg x_2) \land (x_3 \lor x_2 \lor x_4) \land (\neg x_1 \lor \neg x_3 \lor \neg x_4)\]
- Each clause has Boolean variables combined with the OR operator
- 3-CNF has exactly 3 variables in each clause
- The decision problem asks if there is a set of values for the variables that will result in the entire expression being true
3-CNF-Satisfiability

Prove that 3-CNF-SAT is NP-Complete:

1. $3\text{-CNF-SAT} \in \text{NP}$
2. $\text{SAT} \leq_p 3\text{-CNF-SAT}$

1. Same as for SAT.
2. Three steps (see the next slide for the necessary details of each step):
   a. Construct a binary parse tree from the formula, introducing a variable $y_i$ for each internal node; rewrite the formula $\phi$ as $\phi'$
   b. Convert to CNF: $\phi'$ becomes $\phi''$ (with at most 3 literals) using DeMorgan’s Law.
   c. Create exactly 3 literals using extra Boolean variables $p$ and $q$. $\phi''$ becomes $\phi'''$

Note: all three steps can be done in polynomial time!

Conclusion: If we were able to easily find a satisfying solution to a 3-CNF-SAT formula, we could convert any other (SAT) formula to 3-CNF-SAT using the 3-step procedure given above, and easily get a satisfying solution to the SAT formula.
3-CNF-Satisfiability

Step a: Construct a binary parse tree from the formula; introducing a variable $y_i$ for each internal node; rewrite $\phi$ as $\phi'$:

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

$$\phi' = y_1 \land (y_1 \leftrightarrow (y_2 \land \neg x_2))$$
$$\land (y_2 \leftrightarrow (y_3 \lor y_4))$$
$$\land (y_3 \leftrightarrow (x_1 \rightarrow x_2))$$
$$\land (y_4 \leftrightarrow \neg y_5)$$
$$\land (y_5 \leftrightarrow (y_6 \lor x_4))$$
$$\land (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3)).$$
3-CNF-Satisfiability

Step b: Convert to CNF: \( \phi' \) becomes \( \phi'' \) (each clause has at most 3 literals).

Below is the truth table for clause \( \phi'_1 = (y_1 \leftrightarrow (y_2 \land \neg x_2)) \):

<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>

The formula \( \neg \phi'_1 \) corresponds to the rows in the table where \( \phi'_1 \) is 0:

\[
(y_1 \land y_2 \land x_2) \lor (y_1 \land \neg y_2 \land x_2) \lor (y_1 \land \neg y_2 \land \neg x_2) \lor (\neg y_1 \land y_2 \land \neg x_2)
\]

Next, apply DeMorgan’s Law to \( \neg \phi'_1 \) to get \( \neg \phi''_1 \) :

\[
\phi''_1 = (\neg y_1 \lor \neg y_2 \lor \neg x_2) \land (\neg y_1 \lor y_2 \lor \neg x_2)
\land (\neg y_1 \lor y_2 \lor x_2) \land (y_1 \lor \neg y_2 \lor x_2),
\]

This is equivalent to the original clause \( \phi'_1 \)
3-CNF-Satisfiability

Step c: Exactly 3 literals using extra Boolean variables $p$ and $q$. $\phi''$ becomes $\phi'''$:

The third and final step of the reduction further transforms the formula so that each clause has \textit{exactly} 3 distinct literals. We construct the final 3-CNF formula $\phi'''$ from the clauses of the CNF formula $\phi''$. The formula $\phi'''$ also uses two auxiliary variables that we shall call $p$ and $q$. For each clause $C_i$ of $\phi''$, we include the following clauses in $\phi'''$:

- If $C_i$ has 3 distinct literals, then simply include $C_i$ as a clause of $\phi'''$.
- If $C_i$ has 2 distinct literals, that is, if $C_i = (l_1 \lor l_2)$, where $l_1$ and $l_2$ are literals, then include $(l_1 \lor l_2 \lor p) \land (l_1 \lor l_2 \lor \neg p)$ as clauses of $\phi'''$. The literals $p$ and $\neg p$ merely fulfill the syntactic requirement that each clause of $\phi'''$ has exactly 3 distinct literals. Whether $p = 0$ or $p = 1$, one of the clauses is equivalent to $l_1 \lor l_2$, and the other evaluates to 1, which is the identity for AND.
- If $C_i$ has just 1 distinct literal $l$, then include $(l \lor p \lor q) \land (l \lor p \lor \neg q) \land (l \lor \neg p \lor q) \land (l \lor \neg p \lor \neg q)$ as clauses of $\phi'''$. Regardless of the values of $p$ and $q$, one of the four clauses is equivalent to $l$, and the other 3 evaluate to 1.
Clique

• A clique is a complete subgraph of G
• The size of a clique is the number of vertices it contains
• Clique problem:
  – Does a clique of a given size, k, exist within a graph, G? (decision problem)
  – Find a clique of maximum size in a graph (optimization problem)
Clique

Prove that CLIQUE is NP-Complete:
1. CLIQUE $\in$ NP
2. 3-CNF-SAT $\leq_p$ graphs where cliques occur in triples

1. The set of vertices in the clique is the qualifying certificate. We can check whether this set is a clique in polynomial time by making sure that for each pair of vertices $u$ and $v$ in the clique, the edge $(u, v)$ belongs to the edge set of the graph.
2. Let $\phi$ be an instance of 3-CNF-SAT with $k$ clauses, where each clause has exactly 3 distinct literals. Construct a graph $G$ such that $\phi$ is satisfiable if and only if $G$ has a clique of size $k$ (How to construct the graph? See the following slide).
Clique

Construct $G$ from the 3-CNF formula $\phi = C_1 \land C_2 \land C_3$ where $C_1 = (x_1 \lor \neg x_2 \lor \neg x_3)$, $C_2 = (\neg x_1 \lor x_2 \lor x_3)$, and $C_3 = (x_1 \lor x_2 \lor x_3)$: For each clause, create three vertices, $x_1$, $x_2$, and $x_3$ in $G$. Place an edge between two vertices if (1) they are in different clauses, and (2) a vertex is not the negation of the other vertex. If we can find a clique of size $k = 3$, then we have a satisfying assignment of $\phi$: $(x_1 = 0$ or $1$, $x_2 = 0$, $x_3 = 1)$ corresponding to the lightly shaded vertices in the graph shown here.
Vertex Cover

• A vertex cover is a set of vertices that “covers” all of the edges in the graph
  – Each vertex “covers” its incident edge
  – A clique has a vertex cover of size N-1
  – A star graph has a vertex cover of size 1

• Vertex cover problem:
  – Find a vertex cover of minimum size (optimization)
  – Does a vertex cover of size \( k \) exist in a graph? (decision problem)
Prove that VERTEX-COVER is NP-Complete:

1. VERTEX-COVER $\in$ NP
2. CLIQUE $\leq_p$ VERTEX-COVER

1. For every vertex in the vertex cover, mark its incident edges. When done, make sure all edges in G have been marked.

2. Given graph G that has a clique in it, create the complement graph G', which contains all of the edges not in G. Then find the vertex cover of G'. The vertex cover of G' consists of exactly the vertices that are not in the clique of G. Therefore, graph G has a clique of size $k$ if and only if G' has a vertex cover of size $|V| - k$, where $|V|$ is the number of vertices in G.
Reducing CLIQUE to VERTEX-COVER. (a) An undirected graph \( G = (V, E) \) with clique \( V' = \{u, v, x, y\} \). (b) The graph \( G' \) produced by the reduction algorithm that has vertex cover \( V - V' = \{w, z\} \).
Some Well-Known NP-Complete Problems

- 3-CNF
- Circuit-Sat
- Formula-Sat
- Clique
- Vertex Cover
- Hamiltonian Cycle
- Traveling Salesman
- Subset Sum
- Graph Coloring
- Crossword Puzzles
- Longest Common Subsequence (LCS) for more than 2 strings
- 0-1 Knapsack
- Exam Scheduling and CPU Register Assignment (Graph Coloring)
- 3D matching
- Chess
- Minesweeper ($1,000,000 Grand Challenge to solve it in p-time!)
- Tetris
- Rubic’s Cube
- Longest Path