NP completeness

Reading:
Chapter 10, Section 10.3 (skim the rest)

Exercises:
10.3: 1, 4, 6, 7, 8, 10

Our old list of problems

- Sorting
- Searching
- Shortest paths in a graph
- Minimum spanning tree
- Primality testing
- Traveling salesman problem
- Knapsack problem
- Chess
- Towers of Hanoi
- Program termination

Classifying a problem’s complexity

Is there a polynomial-time algorithm that solves the problem?

Possible answers:

- yes
- no
  - because it can be proved that all algorithms take exponential time
  - because it can be proved that no algorithm exists at all to solve this problem
- don’t know
- don’t know, but if such algorithm were to be found, then it would provide a means of solving many other problems in polynomial time
Types of problems

- **Optimization problem**: construct a solution that maximizes or minimizes some objective function

- **Decision problem**: answer yes/no to a question

Many problems will have decision and optimization versions.

*Example*: Traveling salesman problem

- **Optimization**: find Hamiltonian cycle of minimum weight
- **Decision**: find Hamiltonian cycle of weight < \( k \)

Some more problems

- **Partition**: Given \( n \) positive integers, determine whether it is possible to partition them into two disjoint subsets with the same sum

- **Bin packing**: Given \( n \) items whose sizes are positive rational numbers not larger than 1, put them into the smallest number of bins of size 1

- **Graph coloring**: For a given graph, find its chromatic number, i.e., the smallest number of colors that need to be assigned to the graph's vertices so that no two adjacent vertices are assigned the same color

- **CNF satisfiability**: Given a boolean expression in conjunctive normal form (conjunction of disjunctions of literals), is there a truth assignment to the variables that makes the expression true?

The class \( P \)

\( P \) is the class of decision problems that are solvable in \( O(p(n)) \), where \( p(n) \) is a polynomial on \( n \)

- Why polynomial?
- If not, very inefficient
- Nice closure properties
- Machine independent in a strong sense
The class $NP$

- $NP$: the class of decision problems that are solvable in polynomial time on a nondeterministic machine.
- A deterministic computer is what we know.
- A nondeterministic computer is one that can “guess” the right answer or solution.
- Thus $NP$ can also be thought of as the class of problems whose solutions can be verified in polynomial time; or that can be solved in polynomial time on a machine that can pursue infinitely many paths of the computation in parallel.
- Note that $NP$ stands for “Nondeterministic Polynomial-time.”

Example: CNF satisfiability

- This problem is in $NP$. Nondeterministic algorithm:
  - Guess truth assignment
  - Check assignment to see if it satisfies CNF formula.
- Example:
  
  $$(\neg A \lor \neg B \lor C) \land (\neg A \lor B \lor D \lor E) \land (F \lor \neg D)$$

- Truth assignments:
  
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<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>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>... (how many more?)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Checking phase: $O(n)$

Where are we now?

- Exhibited nondeterministic poly-time algorithm for CNF-satisfiability
- CNF-sat is in $NP$
- Similar algorithms can be found for TPS, HC, Partition, etc proving that these problems are also in $NP$
- All the problems in $P$ can also be solved in this manner (but no guessing is necessary), so we have:
  
  $P \subseteq NP$

- Big question: $P = NP$?
A decision problem $D$ is $\text{NP}$-complete if:

1. $D \in \text{NP}$
2. Every problem in $\text{NP}$ is polynomial-time reducible to $D$

Cook's theorem (1971): $\text{CNF-sat}$ is $\text{NP}$-complete

Other $\text{NP}$-complete problems obtained through polynomial-time reductions of known $\text{NP}$-complete problems

The class of $\text{NP}$-complete problems is denoted $\text{NPC}$.

Example: Polynomial-time reduction of directed $\text{HC}$ to undirected $\text{HC}$

What does this prove?

• $\text{HC}$ is harder or easier for directed graphs?

• If $\text{HC}$ is $\text{NPC}$ for directed graphs, is it also $\text{NPC}$ for undirected graphs?

OR

• If $\text{HC}$ is $\text{NPC}$ for undirected graphs, is it also $\text{NPC}$ for directed graphs?