Lecture 29: NP-complete problems
• Optimization problem: find the best solution from among a large space of possibilities.

The format for an optimization problem is

Instance: what does the input look like?
Solution format: what does an output look like?
Constraints: what properties must a solution have?
Objective function: what makes a solution better or worse?
OPTIMIZATION PROBLEMS WE’VE SEEN

1. Max flow
2. Shortest path
3. Cookie monster problem
4. Event scheduling
1. Shortest path - Dijkstra’s algorithm, fast
2. Minimum spanning tree - Kruskal’s algorithm, fast
3. Optimal event scheduling - greedy alg, fast
4. Maximum independent set - backtracking, slow
5. Longest increasing subsequence - dp, fast (relatively)
6. Maximum flow - hill-climbing, fast (relatively)
Search problem: find a solution from among a large space of possibilities that meets the constraints, or say none exists.

The format for a search problem is

Instance: what does the input look like?
Solution format: what does an output look like?
Constraints: what properties must a solution have?
Objective: Find any solution meeting constraints, or certify none exists
Decision version: Is there a solution meeting the constraints? Yes/No
SEARCH/DECISION PROBLEMS WE’VE SEEN

1. 
2. 
3. 
4. 
5. 
6. 
7. 
8.
DECISION PROBLEMS WE’VE SEEN

1. Is there a path? DFS/BFS, fast
2. Is there a topological sort? I.e., is a graph a DAG? (greedyish)
3. Is a graph 3-colorable? (bt, slow)
4. Is there an arbitrage? (dp, somewhat fast)
5. Does a bipartite graph have a perfect matching? hc, somewhat fast
OTHER SEARCH / OPTIMIZATION PROBLEMS

Finding the lowest energy protein folding
Finding the minimum area arrangement for components in a chip
Solving a Sudoku puzzle
Packing your car trunk
Decrypting a message
Proving a theorem
Put 8 queens on a chessboard such that no two are attacking.

Solution format:

An arrangement of 8 queens on a chessboard

Constraint:
no two are attacking.
Given a graph with nodes representing people, and there is an edge between A and B if A and B are enemies, find the largest set of people such that no two are enemies. In other words, given an undirected graph, find the largest set of vertices such that no two are connected with an edge.

**Instance:**

**Solution format:** subset of vertices

**Constraint:** Between every two vertices there is no edge.

**Objective:** maximize size of subset.
How can we view n queens as a special case of independent set?
We have a number of approaches that give fast algorithms for a wide variety of search and optimization problems.

But other search and optimization problems seem to defeat our standard algorithmic methods, e.g., Factoring, independent set, 3-coloring.

Are ALL reasonable search and optimization problems easy? If not, what makes some hard? Can we identify the hard ones?
When is it hopeless to expect to find a polynomial-time algorithm for a search problem?

Instance: what does the input look like?  
Solution format: what does an output look like?  
Constraints: what properties must a solution have?  
Objective: Find any solution meeting constraints, or certify none exists
REASONABLE SEARCH PROBLEMS

NP = class of decision versions of reasonable search problems

Instance: what does the input look like?
  x, n bits
Solution format: what does an output look like?
  y, poly(n) bits to describe
Constraints: what properties must a solution have?
  R(x,y) can be checked in poly time.
Objective: Given x, does there exist a y so that R(x,y)?
Any polynomial time decidable yes/no problem is in NP, making $y$ trivial (0 or 1).
So P is a subset of NP

$P = NP$: Every decision version of a reasonable search problem is solvable in polynomial time

Implications: Using binary search, can reduce search/optimization to decision. So $P = NP$ implies every reasonable search and optimization problem can be solved in polynomial time.

Totally open after many years, one of the Clay Institute list of most important open problems in mathematics.
The famous logician Kurt Godel asked the famous computer scientist, mathematician, and economist John von Neumann the P vs. NP question in a private letter, written shortly before von Neumann’s death.
S.V. Yablonski invents the term "perebor" or "brute force search" to describe the combinatorial explosion limiting algorithms, especially for circuit design problems (1959)
In 1965, Jack Edmonds gives the first polynomial time algorithm for perfect matching on general graphs. To explain the significance to referees, he introduces a section defining P, NP and posing the P vs. NP question.
In 1971, Steve Cook defines NP-completeness and proves that several problems from logic and combinatorics are NP-complete, meaning that P=NP if and only if any of these problems are polynomial time solvable.
Following Cook’s work, Richard Karp showed that a large number of the most Important optimization problems from all sub-areas (scheduling, graph theory, Number theory, logic, Puzzles and games, packing, Coding, …) are NP-complete
Leonid Levin, a student of Kolmogorov’s, publishes similar results to Cook and Karp’s in his thesis, but needs to be careful to disguise what he’s claiming, since it might be interpreted as Questioning earlier work on perebor.
Garey and Johnson’s classic Textbook (1979) includes an Appendix listing hundreds of NP-complete problems.
Since then, thousands of NP-complete Problems have been identified in pretty much any area with computational problems - physics, biology, chemistry, economics, sociology, linguistics, games, engineering, …..
TETRIS
CANDY CRUSH
We can view a decision problem as the set of all instances for which the answer is yes.

A reduction from decision problem A to decision problem B is a polynomial time computable function F so that

\[ x \in A \equiv F(x) \in B \]

Then if we have an algorithm Alg_B solving B, we can get an Alg_A solving A by using Alg_A (x) := Alg_B(F(x)). If A reduces to B, we write, \( A \leq_{\text{mp}} B \) (mp: polytime map reduction)
Existential problem: ``Given input x, does there exist a y so that R(x,y)?'' The decision versions of search and optimization problems have this format.

If A and B are both existential problems, defined by $x_A, y_A, R_A$ and $x_B, y_B, R_B$, to show equivalence, we actually need two more maps:

- $F(x_A) = x_B$ is the actual reduction
- $G(y_B) = y_A$ so that $R_B(x_B, y_B)$ implies $R_A(x_A, G(y_B))$
- $H(y_A) = y_B$ so that $R_A(x_A, y_A)$ implies $R_B(x_B, H(y_A))$
We’ve used reductions many times to design algorithms.

Reducing max bandwidth path to connectivity
Reducing LIS to longest path in DAG
Reducing arbitrage to negative cycles
Reducing perfect matching to maximum flow
If $A \leq_{mp} B$ and $B \in P$ then $A \in P$

So if $A$ reduces to $B$ and $B$ is easy, then $A$ is easy.

Contrapositive:

If $A$ reduces to $B$ and $A$ is hard, then $B$ is hard.

Caution: If $A$ reduces to $B$, and $B$ is hard, then nothing happens

If $A$ reduces to $B$, and $A$ is easy, then nothing happens
3-coloring reduces to Maximum Independent Set

Given an undirected graph \( G=(V,E) \), we make another graph \( H=F(G)=(V',E') \)

As follows.

For each \( u \in V \), create three vertices in \( V' \): \( u_g, u_r, u_b \) and connect them in a triangle.

For each edge \( \{u,v\} \) in \( E \), add the edges \( \{u_g, v_g\}, \{u_r, v_r\}, \{u_g, v_g\} \)

In \( E' \).
Claim: H has an independent set of size \(n = |V| = |V'|/3\) if and only if G is 3-colorable.

If H has an independent set of size \(n\), it must select exactly one of \(\{u_r, u_g, u_b\}\) for each \(u\). Color \(u\) by the vertex in the independent set. This must be a valid coloring, because say, two adjacent vertices can’t both be red, since \(\{u_r, v_r\}\) is an edge, and the set is independent.

On the other hand, if there’s a coloring of G, use that color to pick the vertex among \(\{u_g, u_r, u_b\}\).
If we can solve independent set quickly, we can solve 3-coloring quickly. If we believe 3-coloring is hard, we believe independent set is hard.
A problem $A$ is NP-complete if $A \in NP$ and
For EVERY $B \in NP$,
$B \leq_{mp} A$

It follows that, if $A$ is NP-complete, then
$A \in P \equiv P = NP$

So each NP-complete problem determines the fate of all search and Optimization problems. In particular, they are all equivalent to each other.
<table>
<thead>
<tr>
<th>NP-COMPLETE</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>(SAT), (3SAT)</td>
<td>(2SAT)</td>
</tr>
<tr>
<td>(TSP)</td>
<td>Shortest path</td>
</tr>
<tr>
<td>(HAMILTONIAN PATH)</td>
<td>(MST)</td>
</tr>
<tr>
<td>(KNAPSACK)</td>
<td>(EULER PATH)</td>
</tr>
<tr>
<td>(INDEP.SET)</td>
<td>(UNARY.KNAPSACK)</td>
</tr>
<tr>
<td></td>
<td>(INDEP.SET.TREES)</td>
</tr>
</tbody>
</table>
If an optimization problem is not in polynomial time, that means there is no algorithm that runs quickly on every input, and gives the exact optimal answer on every input.

**WHAT DOES NP-COMPLETENESS MEAN?**
Generic problem: Is there a $y$ that has a certain property, $R(x,y)$?

First step: Build a Boolean circuit to compute $R(x,y)$, $C$. $C$ has $m$ inputs, $y_1,..y_m$, then $T$ gates, $g_1,..g_T$

Each gate is a Boolean operation of previous gates or inputs, e.g., $g_i = g_j$ or $g_k$.

We want to know: is there some input $y$ that causes $C$ to output True, $g_T=True$?
Have two vertices per gate, True\textsubscript{i}, False\textsubscript{i},
And four intermediate vertices per gate, \( gi = gj \) or \( gk \)
For each input, we have two vertices, one for true, one for false.
For each Boolean operation gate, we have two vertices (one for true and one for false) and a four vertex middle layer.
Output gate has only `true` vertex, eliminating false.

Total: $2n + 6m - 1$ vertices
Is the maximum independent set of size $n + 2m$? (Exactly one value per gate and input, and exactly one of each four in the gadget for a gate)
Equivalence

If there is a solution $y_1 \ldots y_n$ that causes the circuit to accept:

Look at the values of the gates in the circuit on input $y$, and use them to pick the corresponding vertex for each input and gate.

Use the pair of input values to pick one vertex in the gadget for each gate.

Output gate is true, so can pick that one as well. Defines independent set, since each gate is correctly computed from inputs.

Total size: $n+2m$
If $S$ is an independent set of size $n+2m$, must have exactly one value per input, one value per gate, and one node in the middle layer of each gadget.

Use input values to define answer $y$ to search problem.

Because $S$ is an independent set, each gate value must be the value of the gate on input $y$.
Output gate must have value true.
Therefore, $y$ is a solution to the original search problem.
Change problem
Restrict instances. Are there some restrictions on the Instances you have to solve that you didn’t include in the problem Definition?
Approximation algorithms: Is coming close to optimal good enough?
Multi=parameter analysis: Can you be exponential in a typically small value?
Hope that an exponential time algorithm runs faster on your instance: SAT solvers, ILP
Average-case analysis
Search heuristics