Guessing Game: NP-Complete?

1. **LONGEST-PATH**: Given a graph $G = (V, E)$, does there exists a simple path of length at least $k$ edges?
   
   **YES**

2. **SHORTEST-PATH**: Given a graph $G = (V, E)$, does there exists a simple path of length at most $k$ edges?
   
   **In $P$**

3. **2-SAT**: Given a formula $\Phi$ such that each clause has at most 2 literals, is $\Phi$ satisfiable?
   
   **In $P$**

4. **3-COLOR**: Given a graph $G = (V, E)$, can we color the nodes of $G$ with 3 colors such that no two nodes joined by an edge have the same coloring?
   
   **YES**

5. **Factoring**: Given an integer $N$. Find the factors of $N$.
   
   **INAPPLICABLE**

Coping With NP-Completeness

Q. Suppose I need to solve an NP-complete problem. What should I do?
A. Theory says you’re unlikely to find poly-time algorithm.

Must sacrifice one of three desired features.
- Solve problem to optimality.
- Solve problem in polynomial time.
- Solve arbitrary instances of the problem.

This lecture. Solve some special cases of NP-complete problems that arise in practice.

10.1 Finding Small Vertex Covers
**Vertex Cover**

**VERTEX COVER:** Given a graph $G = (V, E)$ and an integer $k$, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge $(u, v)$ either $u \in S$, or $v \in S$, or both.

**Finding Small Vertex Covers**

Q. What if $k$ is small?

Brute force. $O(kn^{k-1}).$

- Try all $\binom{n}{k} = O(n^k)$ subsets of size $k$.
- Takes $O(kn)$ time to check whether a subset is a vertex cover.

Goal. Limit exponential dependency on $k$, e.g., to $O(2^k kn)$.

Ex. $n = 1,000, k = 10.$

Brute. $kn^{k-1} = 10^{34}$ ⇒ infeasible.

Better. $2^k kn = 10^7$ ⇒ feasible.

Remark. If $k$ is a constant, algorithm is poly-time; if $k$ is a small constant, then it’s also practical.

**Finding Small Vertex Covers: Algorithm**

Claim. The following algorithm determines if $G$ has a vertex cover of size $\leq k$ in $O(2^k kn)$ time.

```java
boolean Vertex-Cover(G, k) {
    if (G contains no edges) return true
    if (G contains $\geq kn$ edges) return false
    let (u, v) be any edge of G
    a = Vertex-Cover(G - {u}, k-1)
    b = Vertex-Cover(G - {v}, k-1)
    return a or b
}
```

Pf.
- Correctness follows previous two claims.
- There are $\leq 2^{kn}$ nodes in the recursion tree; each invocation takes $O(kn)$ time. •
Finding Small Vertex Covers: Recursion Tree

\[ T(n, k) \leq \begin{cases} cn & \text{if } k = 1 \\ 2T(n, k-1) + ckn & \text{if } k > 1 \end{cases} \Rightarrow T(n, k) \leq 2^k cn 

10.2 Solving NP-Hard Problems on Trees

10.2.1 Independent Set on Trees

**Independent set on trees.** Given a tree, find a maximum cardinality subset of nodes such that no two share an edge.

**Fact.** A tree on at least two nodes has at least two leaf nodes.

**Key observation.** If \( v \) is a leaf, there exists a maximum size independent set containing \( v \).

**Pf.** (exchange argument)
- Consider a max cardinality independent set \( S \).
- If \( v \in S \), we're done.
- If \( u \in S \) and \( v \notin S \), then \( S \cup \{ v \} \) is independent \( \Rightarrow S \) not maximum.
- If \( u \notin S \) and \( v \in S \), then \( S \cup \{ v \} \setminus \{ u \} \) is independent. \( \blacksquare \)

**Theorem.** The following greedy algorithm finds a maximum cardinality independent set in forests (and hence trees).

**Algorithm.**

```
Independent-Set-In-A-Forest(F) {
    S ← φ
    while (F has at least one edge) {
        Let e = (u, v) be an edge such that v is a leaf
        Add v to S
        Delete from F nodes u and v, and all edges incident to them.
    }
    return S
}
```

**Pf.** Correctness follows from the previous key observation. \( \blacksquare \)

**Remark.** Can implement in \( O(n) \) time by considering nodes in postorder.
Chapter 11
Approximation Algorithms

11.4 The Pricing Method: Vertex Cover

Approximation Algorithms

Q. Suppose I need to solve an NP-hard problem. What should I do?
A. Theory says you’re unlikely to find a poly-time algorithm.

Must sacrifice one of three desired features:
- Solve problem to optimality.
- Solve problem in poly-time.
- Solve arbitrary instances of the problem.

\[ \rho \]-approximation algorithm.
- Guaranteed to run in poly-time.
- Guaranteed to solve arbitrary instance of the problem
- Guaranteed to find solution within ratio \( \rho \) of true optimum.

Challenge. Need to prove a solution’s value is close to optimum, without even knowing what optimum value is!

Weighted Vertex Cover

Weighted vertex cover. Given a graph \( G \) with vertex weights, find a vertex cover of minimum weight.

\[ \begin{align*}
\text{weight} = 2 + 2 + 4 &= 8 \\
\text{weight} = 9
\end{align*} \]
Weighted Vertex Cover

**Pricing method.** Each edge must be covered by some vertex $i$. Edge $e$ pays price $p_e \geq 0$ to use vertex $i$.

**Fairness.** Edges incident to vertex $i$ should pay $\leq w_i$ in total.

For each vertex $i$: $\sum_{e \in \{(i,j)\}} p_e \leq w_i$

**Claim.** For any vertex cover $S$ and any fair prices $p_e$: $\sum p_e \leq w(S)$.

**Proof.**

\[ \sum_{i \in S} p_e \leq \sum_{i \in S} \sum_{e \in \{(i,j)\}} p_e \leq \sum_{i \in S} w_i = w(S). \]

Pricing Method

**Pricing method.** Set prices and find vertex cover simultaneously.

```
Weighted-Vertex-Cover-Approx(G, w) {
    foreach e in E
        p_e = 0
    while (\exists edge i-j such that neither i nor j are tight)
        select such an edge e
        increase p_e without violating fairness
    return S
}
```

**Theorem.** Pricing method is a 2-approximation.

**Proof.**

- Algorithm terminates since at least one new node becomes tight after each iteration of while loop.
- Let $S$ be set of all tight nodes upon termination of algorithm. $S$ is a vertex cover: if some edge $i-j$ is uncovered, then neither $i$ nor $j$ is tight. But then while loop would not terminate.
- Let $S^*$ be optimal vertex cover. We show $w(S) \leq 2w(S^*)$.

\[ w(S) = \sum_{i \in S} w_i = \sum_{i \in S} \sum_{e \in \{(i,j)\}} p_e \leq \sum_{i \in S} \sum_{e \in \{(i,j)\}} p_e = 2 \sum_{e \in E} p_e \leq 2w(S^*). \]

Figure 11.8

```
(a)  3  0  0  0
     b      c      d

(b)  3  0  0  0
     b  tight  c      d

(c)  3  0  0  0
     b  tight  c  tight  d

(d)  3  0  0  1
     b  tight  c  tight  d

price of edge a-b

vertex weight
```

```
Figure 11.8
```
13.4 MAX 3-SAT

**Maximum 3-Satisfiability**

Given 3-SAT formula, find a truth assignment that satisfies as many clauses as possible.

\[ C_1 = x_2 \lor \overline{x}_3 \lor \overline{x}_4 \]
\[ C_2 = x_2 \lor x_3 \lor x_4 \]
\[ C_3 = \overline{x}_2 \lor x_3 \lor x_4 \]
\[ C_4 = x_1 \lor \overline{x}_2 \lor \overline{x}_3 \]
\[ C_5 = x_1 \lor \overline{x}_2 \lor x_4 \]

**Remark.** NP-hard search problem.

**Simple idea.** Flip a coin, and set each variable true with probability \( \frac{1}{2} \), independently for each variable.

**Claim.** Given a 3-SAT formula with \( k \) clauses, the expected number of clauses satisfied by a random assignment is \( \frac{7k}{8} \).

**Pf.** Consider random variable \( Z_j = \begin{cases} 1 & \text{if clause } C_j \text{ is satisfied} \\ 0 & \text{otherwise} \end{cases} \).

Let \( Z = \) weight of clauses satisfied by assignment \( Z_j \).

\[ E[Z] = \sum_{j=1}^{k} E[Z_j] \]

by linearity of expectation

\[ = \sum_{j=1}^{k} \Pr[\text{clause } C_j \text{ is satisfied}] \]

\[ = \frac{7k}{8} \]

**Corollary.** For any instance of 3-SAT, there exists a truth assignment that satisfies at least a \( \frac{7}{8} \) fraction of all clauses.

**Pf.** Random variable is at least its expectation some of the time. •

**The Probabilistic Method**

**Claim.** Given a 3-SAT formula with \( k \) clauses, the expected number of clauses satisfied by a random assignment is \( \frac{7k}{8} \).

**Pf.** Consider random variable \( Z_j = \begin{cases} 1 & \text{if clause } C_j \text{ is satisfied} \\ 0 & \text{otherwise} \end{cases} \).

Let \( Z = \) weight of clauses satisfied by assignment \( Z_j \).

\[ E[Z] = \sum_{j=1}^{k} E[Z_j] \]

by linearity of expectation

\[ = \sum_{j=1}^{k} \Pr[\text{clause } C_j \text{ is satisfied}] \]

\[ = \frac{7k}{8} \]

**Corollary.** For any instance of 3-SAT, there exists a truth assignment that satisfies at least a \( \frac{7}{8} \) fraction of all clauses.

**Pf.** Random variable is at least its expectation some of the time. •

**The Probabilistic Method**

**Corollary.** For any instance of 3-SAT, there exists a truth assignment that satisfies at least a \( \frac{7}{8} \) fraction of all clauses.

**Pf.** Random variable is at least its expectation some of the time. •

**Probabilistic method.** We showed the existence of a non-obvious property of 3-SAT by showing that a random construction produces it with positive probability!
Maximum 3-Satisfiability: Analysis

Q. Can we turn this idea into a 7/8-approximation algorithm? In general, a random variable can almost always be below its mean.

Lemma. The probability that a random assignment satisfies \( \geq 7k/8 \) clauses is at least \( 1/(8k) \).

Pf. Let \( p_j \) be probability that exactly \( j \) clauses are satisfied; let \( p \) be probability that \( \geq 7k/8 \) clauses are satisfied.

Rearranging terms yields \( p \geq 1/(8k) \).

\[
E[Z] = \sum_{j=0}^{\infty} j p_j = \frac{7k}{8} p + \frac{1}{8} \sum_{j=0}^{\infty} j p_j \\
\leq \left( \frac{7k}{8} - \frac{1}{8} \right) \sum_{j=0}^{\infty} j p_j + k \sum_{j=0}^{\infty} p_j \\
\leq \left( \frac{7k}{8} - \frac{1}{8} \right) \cdot 1 + k p
\]

Waiting for a first success. Coin is heads with probability \( p \) and tails with probability \( 1-p \). How many independent flips \( X \) until first heads?

\[
E[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X=j] = \frac{p}{1-p} \sum_{j=0}^{\infty} (1-p)^j = \frac{p}{1-p} \cdot \frac{1}{1-(1-p)} = \frac{1}{p}
\]

Maximum Satisfiability

Extensions.
- Allow one, two, or more literals per clause.
- Find max weighted set of satisfied clauses.

Theorem. [Asano-Williamson 2000] There exists a 0.784-approximation algorithm for \( \text{MAX-SAT} \).

Theorem. [Karloff-Zwick 1997, Zwick+computer 2002] There exists a 7/8-approximation algorithm for version of \( \text{MAX-3SAT} \) where each clause has at most 3 literals.

Theorem. [Håstad 1997] Unless \( P = \text{NP} \), no \( \rho \)-approximation algorithm for \( \text{MAX-3SAT} \) (and hence \( \text{MAX-5SAT} \)) for any \( \rho > 7/8 \).

very unlikely to improve over current randomized algorithm for \( \text{MAX-3SAT} \)

What to do if the problem you want to solve is NP-hard

- More on approximation algorithms
  - Recent research has classified problems based on what kinds of approximations are possible if \( P \neq \text{NP} \):
    - Best: \((1+\epsilon)\) factor for any \( \epsilon > 0 \):
      - packing and some scheduling problems, TSP in plane
    - Some fixed constant factor > 1, e.g. 2, 3/2, 100
      - Vertex Cover, TSP in space, other scheduling problems
    - \( \Theta(\log n) \) factor
      - Set Cover, Graph Partitioning problems
    - Worst: \( \Omega(n^{1+\epsilon}) \) factor for any \( \epsilon > 0 \)
      - Clique, Independent-Set, Coloring

Slides courtesy of Paul Beame
What to do if the problem you want to solve is NP-hard

- Try an algorithm that is provably fast “on average”.
  - To even try this one needs a model of what a typical instance is.
  - Typically, people consider “random graphs”
    - e.g. all graphs with a given # of edges are equally likely
  - Problems:
    - real data doesn’t look like the random graphs
    - distributions of real data aren’t analyzable

What to do if the problem you want to solve is NP-hard

- Try to search the space of possible hints/certificates in a more efficient way and hope it is quick enough
  - Backtracking search
    - E.g. For SAT there are $2^n$ possible truth assignments
    - If we set the truth values one-by-one we might be able to figure out whole parts of the space to avoid.
      - e.g. After setting $x_1 = 1$, $x_2 = 0$ we don’t even need to set $x_3$ or $x_4$ to know that it won’t satisfy
        $$\neg x_1 \lor x_2 \land \neg x_2 \lor x_3 \land x_4 \lor \neg x_2 \land x_1 \lor \neg x_4$$
    - Related technique: branch-and-bound
    - Backtracking search can be very effective even with exponential worst-case time
      - For example, the best SAT algorithms used in practice are all variants on backtracking search and can solve surprisingly large problems

Use heuristic algorithms and hope they give good answers

- No guarantees of quality
- Many different types of heuristic algorithms

- Many different options, especially for optimization problems, such as TSP, where we want the best solution.
- We’ll mention several on following slides

Heuristic algorithms for NP-hard problems

- local search for optimization problems
  - need a notion of two solutions being neighbors
  - Start at an arbitrary solution $S$
  - While there is a neighbor $T$ of $S$ that is better than $S$
    - $S \leftarrow T$
  - Usually fast but often gets stuck in a local optimum and misses the global optimum
  - With some notions of neighbor can take a long time in the worst case
e.g., Neighboring solutions for TSP

Two solutions are neighbors iff there is a pair of edges you can swap to transform one to the other.

Heuristic algorithms for NP-hard problems

- **randomized local search**
  - start local search several times from random starting points and take the best answer found from each point.
  - more expensive than plain local search but usually much better answers.
- **simulated annealing**
  - like local search but at each step sometimes move to a worse neighbor with some probability.
  - probability of going to a worse neighbor is set to decrease with time as, presumably, solution is closer to optimal.
  - helps avoid getting stuck in a local optimum but often slow to converge (much more expensive than randomized local search).
  - analogy with slow cooling to get to lowest energy state in a crystal (or in forging a metal).

Heuristic algorithms

- **artificial neural networks**
  - based on very elementary model of human neurons.
  - Set up a circuit of artificial neurons.
    - each artificial neuron is an analog circuit gate whose computation depends on a set of connection strengths.
  - Train the circuit.
    - Adjust the connection strengths of the neurons by giving many positive & negative training examples and seeing if it behaves correctly.
  - The network is now ready to use.
  - useful for ill-defined classification problems such as optical character recognition but not typical cut & dried problems.

Other directions

- **Quantum computing**
  - Use physical processes at the quantum level to implement “weird” kinds of circuit gates.
  - unitary transformations.
  - Quantum objects can be in a superposition of many pure states at once.
    - can have $n$ objects together in a superposition of $2^n$ states.
  - Each quantum circuit gate operates on the whole superposition of states at once.
    - inherent parallelism but classical randomized algorithms have a similar parallelism: not enough on its own.
    - Advantage over classical: parallel copies interfere with each other.
  - Need totally new kinds of algorithms to work well. Theoretically able to factor efficiently but huge practical problems: errors, decoherence.
Loose Ends

Space Complexity:
- Amount of memory used by an algorithm
- If an algorithm runs in time $T$, then it uses at most $T$ units of memory
- Every poly-time algorithm uses poly-space
- If an algorithm uses $S$ units of memory, it run in time $O(2^S)$

$PSPACE$: class of algorithms solvable by algorithms that use a polynomial amount of space.

$P \subseteq PSPACE$

Another big question in complexity is whether $P = PSPACE$. 