Comments about NP-completeness

NP-completeness says that some instances of the problem are really hard.

But, it could be the case that the typical instance is not hard at all!

The source of the difficulty of these problems is NOT the existence of an exponential number of possible solutions.

There are more difficult problems than NP-complete problems.

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 a polynomial-time algorithm.

Must sacrifice one (or more) of three desired features:

- Solve problem in polynomial time.
- Solve arbitrary instances of the problem.
- Solve problem to optimality.

If we make assumptions about the input, there may be a polynomial time algorithm.

Example: assuming the graph is a DAG or a tree often makes the problem easier – e.g. maximum weighted independent set (chapter 10).

Approximation Algorithms (Chapter 11)

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

Must sacrifice one (or more) of three desired features:

- Solve problem in polynomial time.
- Solve arbitrary instances of the problem.
- Solve problem to optimality.

\( \rho \)-approximation algorithms.

- Run in polynomial time.
- Solve arbitrary instances of the problem
- Guaranteed to find solution within ratio \( \rho \) of true optimum.
- Often implement a greedy strategy.

Randomized Algorithms (Chapter 13)

Algorithmic design patterns:

- Greedy.
- Divide-and-conquer.
- Dynamic programming.
- Reductions (e.g. to network flow).
- Randomization.

Randomization: Use random numbers to make decisions.

Why randomize? Can lead to simplest, fastest, or only known algorithm for a particular problem.

Examples: Graph algorithms, quicksort, hashing, Monte Carlo integration, cryptography.
Exploring the solution space of a problem using local search (chapter 12)

**VERTEX COVER.** Given a graph $G = (V, E)$, find a subset of nodes $S$ of minimal cardinality such that for each $(u, v)$ in $E$, $u$ or $v$ are in $S$.

The cost associated with a vertex cover:

$$E(S) = \text{cardinality of } S$$

**Objective:**

$$S^* = \arg\min E(S)$$

**Neighbor relation:** $S - S'$ if $S'$ can be obtained from $S$ by adding or deleting a single node.

How many neighbors does a vertex cover have?

---

**Local Search**

Strategy:

- Start with a solution $S_0$.
- Move to a neighboring solution.
- Repeat.

**Steepest Descent**

Start with a solution $S_0$.

- Move to a neighboring solution with the lowest value of $E(S)$.
- Repeat until convergence.

---

Vertex cover: start with $S_0 = V$.

How many steps until convergence in the worst case?

How is this different from a greedy algorithm?

In some energy landscapes steepest descent gets stuck in a local minimum.
Steepest Descent

Steepest descent:
- Start with a solution $S_0$.
- Move to a neighboring solution with the lowest value of $E(S)$.
- Repeat until convergence.

This is guaranteed to happen if the problem is NP-hard!

Steepest Descent: Vertex Cover

Local optimum. No neighbor is strictly better.

Local optimum: all nodes as left side
Optimum: all nodes as right side

Optimum: center node only
Local optimum: all other nodes

Optimum: even nodes
Local optimum: omit every third node

Steepest Ascent

In the case where you want to maximize a function, the algorithm would be called steepest ascent:
- Start with a solution $S_0$.
- Move to a neighboring solution with the highest value of $E(S)$.
- Repeat until convergence.

Hill-climbing is a variant where you go to a neighboring solution that has a higher value of $E(S)$ - not necessarily the highest.

Local search for TSP

The neighborhood relationship needs to respect the constraints of the problem.

Example: TSP - need a rule for modifying a tour

How would we find an initial solution for TSP?

Maximum Cut

Maximum cut: Given an undirected graph $G = (V, E)$ with positive integer edge weights, find a partition $(A, B)$ of $V$ such that the total weight of edges crossing the cut is maximized.

Find $A, B$ such that $w(A, B) = \sum_{u \in A, v \in B} w_{uv}$ is maximized.

The decision version of the problem is NP-complete, even when the weights are all equal.

Maximum Cut

Single-flip neighborhood. Given a partition $(A, B)$, move one node from $A$ to $B$, or one from $B$ to $A$.

Steepest ascent algorithm:

```
Maximum-cut-steepest-ascent(G, w) {
  pick a random partition of nodes (A, B)
  while (there exists an improving node v) {
    if (v is in A) move v to B
    else move v to A
  }
  return (A, B)
}
```
Maximum Cut: Local Search Analysis

**Theorem.** Let \((A, B)\) be a locally optimal partition and let \((A^*, B^*)\) be optimal partition.
Then \(w(A, B) \geq \frac{1}{2} w(A^*, B^*)\).

Conclusion: Single flip algorithm gives a good approximation.

Choosing a neighbor relation

The quality of solution will depend on your choice of a neighborhood
- The neighborhood should be rich enough such that you don’t get stuck in bad local optima
- It should be small enough so that you can efficiently search the neighbors for the best local move

The Metropolis Algorithm

**Gibbs-Boltzmann distribution.** The probability of finding a physical system in a state with energy \(E\) is proportional to \(e^{-E / (kT)}\)

- System more likely to be in a lower energy state than higher one.
- \(T\) large: high and low energy states have roughly same probability
- \(T\) small: low energy states are much more probable

**Theorem.** Let \(f(t)\) be fraction of the first \(t\) steps in which simulation is in state \(S\), and assuming some technical conditions, then with probability 1:

\[
\lim_{t \to \infty} f(t) = \frac{1}{Z} e^{-E(S)/ (kT)},
\]

where \(Z = \sum_S e^{-E(S)/ (kT)}\).
Simulation spends roughly the right amount of time in each state, according to Gibbs-Boltzmann equation.

The Metropolis Algorithm

1. **initialization:** \(S_0\), \(k\), \(T\)
2. **while** (not done)
   - randomly choose solution \(S’\) among the neighbors of \(S\)
   - if \(E(S’) < E(S)\)
     - \(S \leftarrow S’\)
   - else
     - with probability \(\exp(- (E(S’)-E(S))/(kT))\)
     - \(S \leftarrow S’\)
Simulated Annealing

- T large → probability of accepting an uphill move is large.
- T small → uphill moves are almost never accepted.
- Idea: turn knob to control T.
- Cooling schedule: $T = T(i)$ at iteration $i$.

Physical analog:
- Take a molten solid and freeze it very abruptly, we do not expect to get a perfect crystal.
- Annealing: cool material gradually from high temperature, allowing it to reach equilibrium at succession of intermediate lower temperatures.
- Algorithm will find optimal solution with high probability if using a sufficiently slow cooling schedule.

Genetic algorithms

Approach: Evolve a population of solutions according to evolutionary operations (recombination and mutation). Survival of the "fittest".
Example: TSP - take two tours and create a new, possibly better one out of the combination.