CSE 521: Design & Analysis of Algorithms I

Dealing with NP-completeness

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

- You might have phrased your problem too generally
  - e.g., in practice, the graphs that actually arise are far from arbitrary
    - maybe they have some special characteristic that allows you to solve the problem in your special case
      - for example the Independent-Set problem is easy on “interval graphs”
        - Exactly the case for interval scheduling!
  - search the literature to see if special cases already solved
What to do if the problem you want to solve is NP-hard

- Try to find an **approximation algorithm**
  - Maybe you can’t get the size of the best Vertex Cover but you can find one within a factor of 2 of the best
    - Given graph $G=(V,E)$, start with an empty cover
    - **While** there are still edges in $E$ left
      - **Choose** an edge $e=\{u,v\}$ in $E$ and add both $u$ and $v$ to the cover
      - Remove all edges from $E$ that touch either $u$ or $v$.
    - Edges chosen don’t share any vertices so optimal cover size must be at least # of edges chosen
What to do if the problem you want to solve is NP-hard

- Polynomial-time approximation algorithms for NP-hard problems can sometimes be ruled out unless P=NP
  - E.g. **Coloring Problem**: Given a graph $G=(V,E)$ find the smallest $k$ such that $G$ has a $k$-coloring.
    - No approximation ratio better than $4/3$ is possible unless P=NP
      - The graph in our NP-completeness reduction is always 4-colorable. This would let us figure out if it is 3-colorable.
Travelling Salesperson Problem

- **TSP**
  - Given a weighted graph $G$ find of a smallest weight tour that visits all vertices in $G$

- **NP-hard**

- Notoriously easy to obtain close to optimal solutions
Minimum Spanning Tree Approximation
Minimum Spanning Tree Approximation: Factor of 2

Any tour contains a spanning tree

\[ \text{MST}(G) \leq \text{TOUR}_{\text{OPT}}(G) \leq 2 \text{MST}(G) \leq 2 \text{TOUR}_{\text{OPT}}(G) \]
Why did this work?

- We found an Euler tour on a graph that used the edges of the original graph (possibly repeated).
- The weight of the tour was the total weight of the new graph.
- Suppose now
  - All edges possible
  - Weights satisfy triangle inequality
    - $c(u,w) \leq c(u,v) + c(v,w)$
Minimum Spanning Tree Approximation: Triangle Inequality

Can shortcut edges
• Go to next new vertex on the Euler tour
Minimum Spanning Tree Approximation: Factor of 2

\[ \text{TOUR}_{\text{OPT}}(G) \leq 2 \text{MST}(G) \leq 2 \text{TOUR}_{\text{OPT}}(G) \]
Christofides Algorithm: A factor 3/2 approximation

- Any Eulerian subgraph of the weighted complete graph will do
  - Eulerian graphs require that all vertices have even degree so

- Christofides Algorithm
  - Compute an MST $T$
  - Find the set $O$ of odd-degree vertices in $T$
  - Add a minimum-weight perfect matching $M$ on the vertices in $O$ to $T$ to make every vertex have even degree
Christofides Approximation
Christofides Approximation

Any tour costs at least the cost of two matchings on $O$

Claim: $2 \text{Cost}(M) \leq \text{TOUR}_{\text{OPT}}$
Knapsack Problem

- For any $\varepsilon > 0$ can get an algorithm that gets a solution within $(1+\varepsilon)$ factor of optimal with running time $O(n^2(1/\varepsilon)^2)$
  - “Polynomial-Time Approximation Scheme” or PTAS
  - Based on maintaining just the high order bits in the dynamic programming solution.
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 $P \neq NP$
    - **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
PCP Theorem and Hardness of Approximation

- **PCP (Probabilistically Checkable Proofs) Theorem**: Every $A \in \text{NP}$ has a polytime verifier $V$ that looks at only 3 random bits of its certificate $c$ such that
  - $x \in A \Rightarrow$ There is a certificate $c$ such that $V(x,c)$ always outputs YES
  - $x \not\in A \Rightarrow$ For every certificate $c$, $V(x,c)$ outputs YES with probability $< 0.99999$

- Implies that there is a polytime reduction $f$ such that
  - $F \in 3\text{SAT} \Rightarrow f(F) \in 3\text{SAT}$
  - $F \not\in \#\text{SAT} \Rightarrow$ any truth assignment to $f(F)$ satisfies at most $88\% (< 7/8+\varepsilon)$ of clauses of $F$
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 
        $$\neg x_1 \lor x_2 \land \neg x_2 \lor x_3 \land x_4 \lor \neg x_3 \land \neg x_1 \lor \neg x_4$$
  - 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 – more later
What to do if the problem you want to solve is NP-hard

- 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

Solution $S$

Solution $T$

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 for NP-hard problems

- **genetic algorithms**
  - view each solution as a string (analogy with DNA)
  - maintain a **population of good solutions**
  - allow **random mutations** of single characters of individual solutions
  - **combine two solutions** by taking part of one and part of another (analogy with crossover in sexual reproduction)
  - **go with the winners**: get rid of solutions that have the worst values and make multiple copies of solutions that have the best values (analogy with natural selection -- survival of the fittest).

- Not a lot of evidence that they work well
  - Often “brittle” when they do – small changes in constraints lead to big changes in solution
  - Usually very slow

- However, the “go with the winners” part of the strategy can be combined with local search and works well in that context
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

- DNA computing
  - Each possible hint for an NP problem is represented as a string of DNA
    - fill a test tube with all possible hints
  - View verification algorithm as a series of tests
    - e.g. checking each clause is satisfied in case of Satisfiability
  - For each test in turn
    - use lab operations to filter out all DNA strings that fail the test (*works in parallel* on all strings; uses PCR)
  - If any string remains the answer is a YES.
  - Relies on fact that Avogadro’s # $6 \times 10^{23}$ is large to get enough strings to fit in a test-tube.
  - Error-prone & problem sizes typically very small!
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
  - **Can reduce brute force search from $2^n$ to $2^{n/2}$ time**

- Strong evidence that they won’t solve NP-complete problems efficiently

- Theoretically able to factor efficiently.

- Large practical problems: errors, decoherence that need to be overcome.