

CSE 417: Algorithms and Computational Complexity

Winter 2001
Lecture 25
Instructor: Paul Beame

1

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 Clique problem is easy on "interval graphs".
 - search the literature to see if special cases already solved

2

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

3

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

- Try to find an approximation algorithm
 - 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

4

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

5

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

- Try to search the space of possible hints in a more efficient way and hope it is quick enough
 - e.g. **back-tracking search**
 - For **Satisfiability** 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 $x_1 \leftarrow 1, x_2 \leftarrow 0$ we don't even need to set x_3 or x_4 to know that it won't satisfy $(\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (x_4 \vee \neg x_3) \wedge (x_1 \vee \neg x_4)$
 - For **Satisfiability** this seems to run in times like $2^{n/20}$ on typical hard instances.
 - Related technique: **branch-and-bound**

6

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

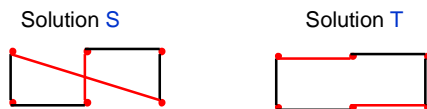
7

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

8

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

9

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)

10

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)
 - 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).
- little evidence that they work well and they are usually very slow
 - as much religion as science

11

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

12

Other fun 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 # 6×10^{23} is large to get enough strings to fit in a test-tube.
- Error-prone & so far only problem sizes less than 15!

13

Other fun 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**
- Need totally new kinds of algorithms to work well. Theoretically able to factor efficiently but huge practical problems: errors, decoherence.

14