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: Give a formula Φ such that each clause has at most 2 literals, is Φ is satisfiable?

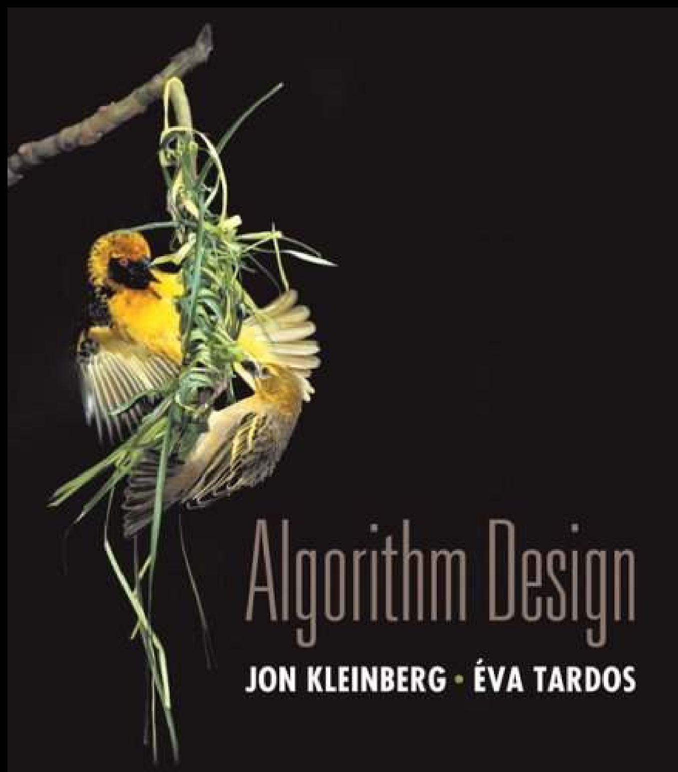
In P

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: Give an integer N. Find the factors of N.

INAPPLICABLE



Chapter 10

Extending the Limits of Tractability

Reading: 10.1-10.2

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.

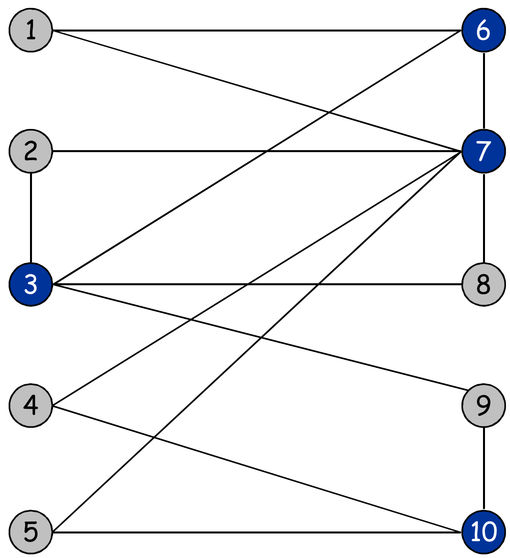
- Solve problem to optimality.
- Solve problem in polynomial time.
- Solve arbitrary instances of the problem.

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.



$k = 4$
$S = \{ 3, 6, 7, 10 \}$

5

Finding Small Vertex Covers

Q. What if k is small?

Brute force. $O(k\,n^{k+1})$.

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

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

Ex. $n = 1{,}000$, $k = 10$.
Brute. $k\,n^{k+1} = 10^{34} \Rightarrow$ infeasible.
Better. $2^k\,k\,n = 10^7 \Rightarrow$ 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

Claim. Let u-v be an edge of G. G has a vertex cover of size $\leq$ k iff at least one of $G - \{u\}$ and $G - \{v\}$ has a vertex cover of size $\leq$ k-1.

delete v and all incident edges

Pf. $\Rightarrow$

- Suppose G has a vertex cover S of size $\leq$ k.
- S contains either u or v (or both). Assume it contains u.
- $S - \{u\}$ is a vertex cover of $G - \{u\}$.

Pf. $\Leftarrow$

- Suppose S is a vertex cover of $G - \{u\}$ of size $\leq$ k-1.
- Then $S \cup \{u\}$ is a vertex cover of G. ▪

Claim. If G has a vertex cover of size k, it has $\leq$ k(n-1) edges.
Pf. Each vertex covers at most n-1 edges. ▪

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.

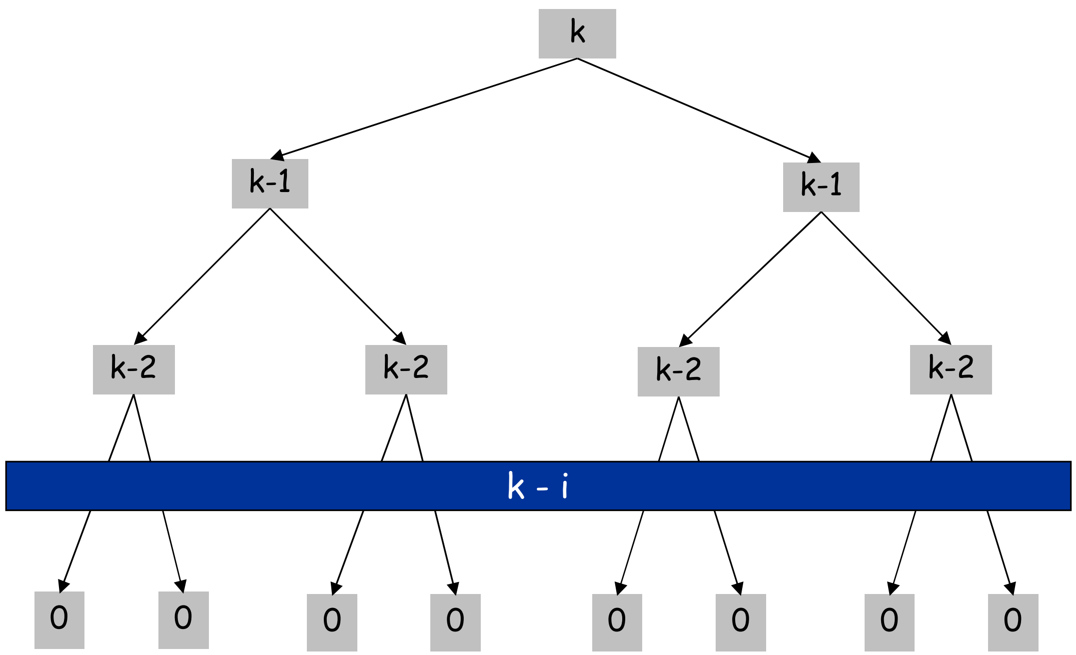
```
boolean Vertex-Cover(G, k) {
   if (G contains no edges)    return true
   if (G contains ≥ kn edges) return false

   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 $\leq 2^{k+1}$ nodes in the recursion tree; each invocation takes $O(kn)$ time. ▪

Finding Small Vertex Covers: Recursion Tree

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



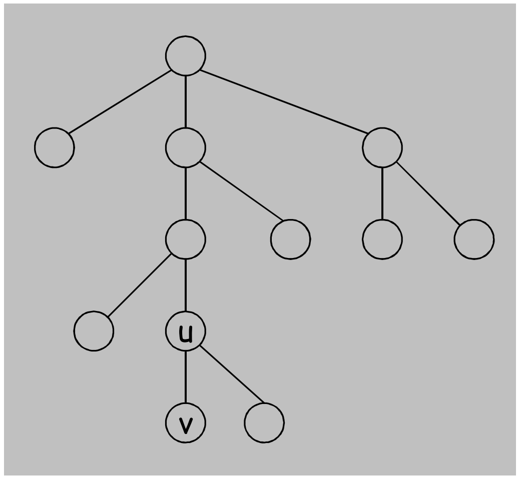
10.2 Solving NP-Hard Problems on Trees

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.

↖ degree = 1



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 $u \notin S$ and $v \notin S$, then $S \cup \{v\}$ is independent $\Rightarrow$ S not maximum.
- IF $u \in S$ and $v \notin S$, then $S \cup \{v\} - \{u\}$ is independent. ▪

11

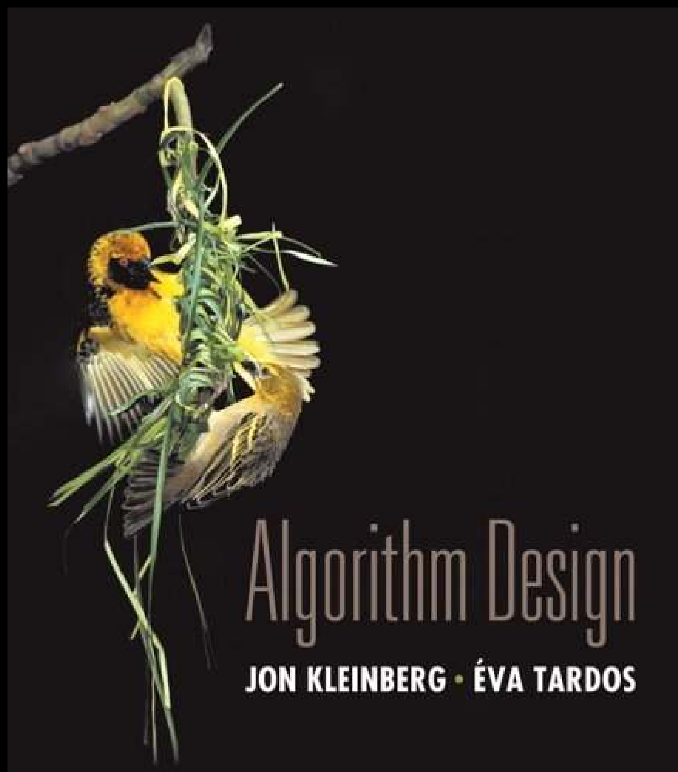
Independent Set on Trees: Greedy Algorithm

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

```
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. ▪

Remark. Can implement in O(n) time by considering nodes in postorder.



Chapter 11

Approximation Algorithms

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.

ρ-approximation algorithm.

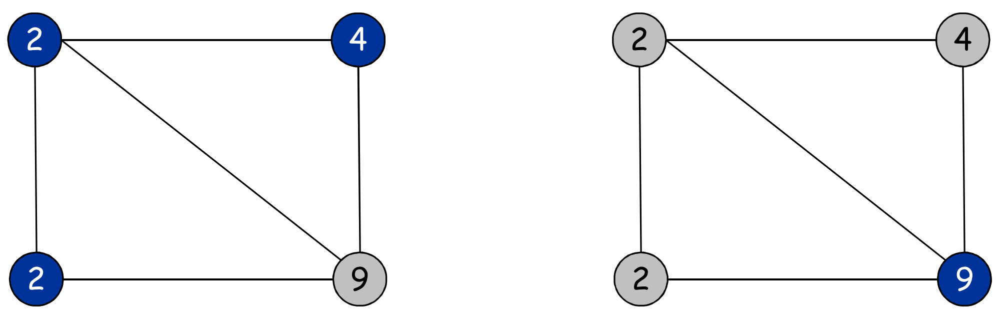
- Guaranteed to run in poly-time.
- Guaranteed to solve arbitrary instance of the problem
- Guaranteed to find solution within ratio ρ of true optimum.

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

11.4 The Pricing Method: Vertex Cover

Weighted Vertex Cover

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



weight = 2 + 2 + 4

weight = 9

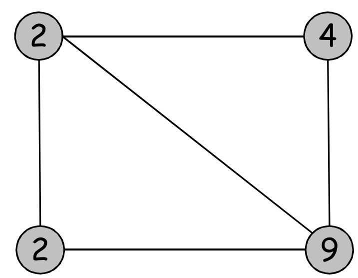
16

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.

$$\text{for each vertex } i: \sum_{e=(i,j)} p_e \leq w_i$$



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

Proof. ▪

$$\sum_{e \in E} p_e \;\leq\; \sum_{i \in S} \sum_{e=(i,j)} p_e \;\leq\; \sum_{i \in S} w_i \;=\; w(S).$$

↑ each edge e covered by at least one node in S

↑ sum fairness inequalities for each node in S

17

Pricing Method

Pricing method. Set prices and find vertex cover simultaneously.

$$\sum_{e=(i,j)} p_e = w_i$$

```
Weighted-Vertex-Cover-Approx(G, w) {
   foreach e in E
      pe = 0

   while (∃ edge i-j such that neither i nor j are tight)
      select such an edge e
      increase pe without violating fairness
   }

   S ← set of all tight nodes
   return S
}
```

Pricing Method

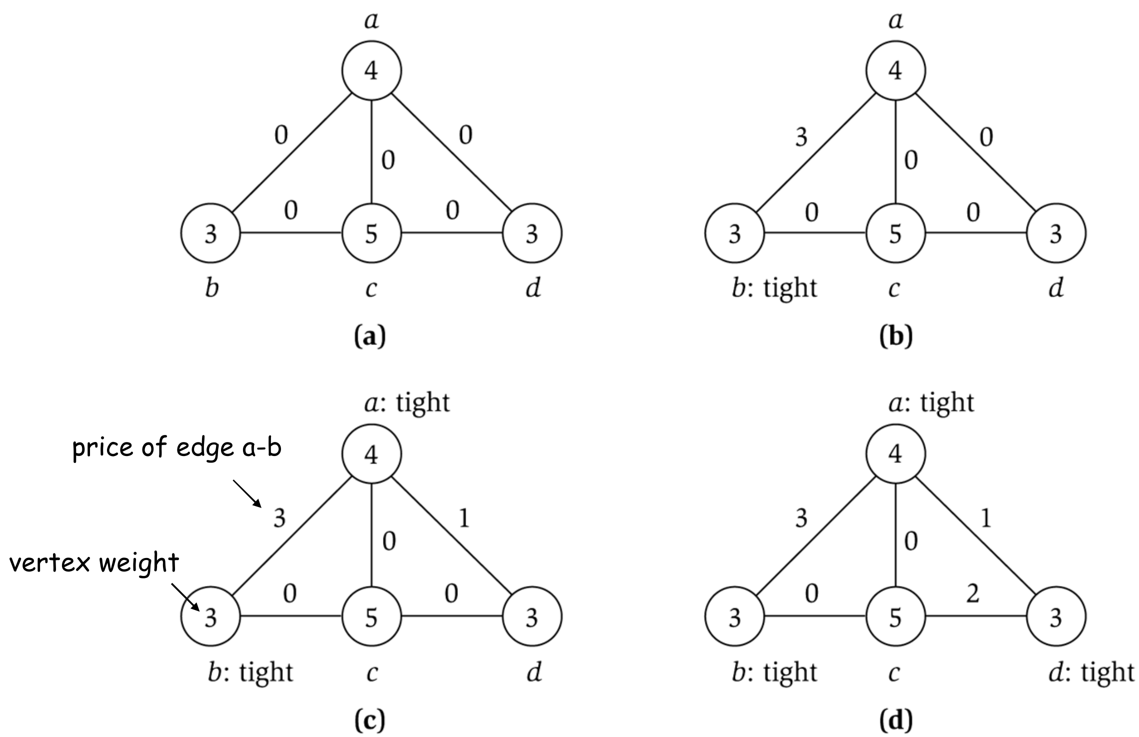


Figure 11.8

Pricing Method: Analysis

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

- Algorithm terminates since at least one new node becomes tight after each iteration of while loop.
- Let S = 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) ≤ 2w(S*).

$$w(S) = \sum_{i \in S} w_i = \sum_{i \in S} \sum_{e=(i,j)} p_e \leq \sum_{i \in V} \sum_{e=(i,j)} p_e = 2 \sum_{e \in E} p_e \leq 2w(S^*). \quad \blacksquare$$

(first =: all nodes in S are tight; first ≤: S ⊆ V, prices ≥ 0; second =: each edge counted twice; second ≤: fairness lemma)

13.4 MAX 3-SAT

Maximum 3-Satisfiability

exactly 3 distinct literals per clause

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

$$
\begin{aligned}
C_1 &= x_2 \vee \overline{x_3} \vee \overline{x_4} \\
C_2 &= x_2 \vee x_3 \vee \overline{x_4} \\
C_3 &= \overline{x_1} \vee x_2 \vee x_4 \\
C_4 &= \overline{x_1} \vee \overline{x_2} \vee x_3 \\
C_5 &= x_1 \vee \overline{x_2} \vee \overline{x_4}
\end{aligned}
$$

Remark. NP-hard search problem.

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

22

Maximum 3-Satisfiability: Analysis

Claim. Given a 3-SAT formula with k clauses, the expected number of clauses satisfied by a random assignment is 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.

$$\begin{aligned} E[Z] &= \sum_{j=1}^{k} E[Z_j] \\ &= \sum_{j=1}^{k} \Pr[\text{clause } C_j \text{ is satisfied}] \\ &= \tfrac{7}{8}k \end{aligned}$$

linearity of expectation

23

The Probabilistic Method

Corollary. For any instance of 3-SAT, there exists a truth assignment that satisfies at least a 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!

24

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 ≥ 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 ≥ 7k/8 clauses are satisfied.

$$\begin{aligned} \tfrac{7}{8}k \;=\; E[Z] &= \sum_{j \ge 0} j\,p_j \\ &= \sum_{j < 7k/8} j\,p_j \;+\; \sum_{j \ge 7k/8} j\,p_j \\ &\le \left(\tfrac{7k}{8} - \tfrac{1}{8}\right) \sum_{j < 7k/8} p_j \;+\; k \sum_{j \ge 7k/8} p_j \\ &\le \left(\tfrac{7}{8}k - \tfrac{1}{8}\right) \cdot 1 \;+\; k\,p \end{aligned}$$

Rearranging terms yields $p \ge 1 / (8k)$. ▪

25

Maximum 3-Satisfiability: Analysis

Johnson's algorithm. Repeatedly generate random truth assignments until one of them satisfies ≥ 7k/8 clauses.

Theorem. Johnson's algorithm is a 7/8-approximation algorithm.

Pf. By previous lemma, each iteration succeeds with probability at least 1/(8k). By the waiting-time bound, the expected number of trials to find the satisfying assignment is at most 8k. ▪

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?

$$E[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X=j] = \sum_{j=0}^{\infty} j\,(1-p)^{j-1}\,p = \frac{p}{1-p}\sum_{j=0}^{\infty} j\,(1-p)^{j} = \frac{p}{1-p}\cdot\frac{1-p}{p^2} = \frac{1}{p}$$

↑ j-1 tails ↑ 1 head

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 MAX-SAT.

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

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

↑ very unlikely to improve over simple randomized algorithm for MAX-3SAT

27

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: $(1+\varepsilon)$ factor for any $\varepsilon>0$.
      - 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: $\Omega(n^{1-\varepsilon})$ factor for any $\varepsilon>0$
      - Clique, Independent-Set, Coloring

Slides courtesy of Paul Beame

28

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

Slides courtesy of Paul Beame

29

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 $x_1 \leftarrow 1$, $x_2 \leftarrow 0$ we don't even need to set x_3 or x_4 to know that it won't satisfy

$$(\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (x_4 \vee \neg x_3) \wedge (x_1 \vee \neg x_4)$$

    - 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

Slides courtesy of Paul Beame

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

Slides courtesy of Paul Beame

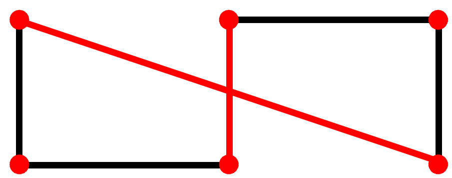
Heuristic algorithms for NP-hard problems

- **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**←**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

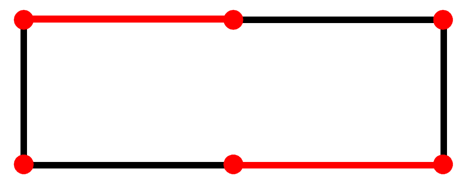
Slides courtesy of Paul Beame

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

Slides courtesy of Paul Beame

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)

Slides courtesy of Paul Beame

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**

Slides courtesy of Paul Beame

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.**

Slides courtesy of Paul Beame

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 ⊆ PSPACE

Another big question in complexity is whether P = PSPACE.