CSE 589 -- Lecture 5

If you try to optimize everything, you will always be unhappy.

-- Don Knuth

Plan for Today

- Review of NP-completeness
  - Cormen, Leiserson, Rivest, Chapter 36
- Coping with NP-completeness
  - exact solutions
  - approximation algorithms: CLR, Chapter 37
  - homeopathic remedies

Easy vs. Hard Problems

- The standard definition of a tractable problem is one that can be solved in time that is polynomial O(P(n)) where P(n) is a polynomial in the size of the input n (size of input = # bits required to represent that input)
- This class of problems called P.
- NP -- class of problems whose solution can be **verified** in polynomial time.
- Formulated as **decision problems:** problems whose answer is either yes or no.

Polynomial Time Reductions

- Let R and Q be two problems. We say that R is **polynomially reducible** to Q if there is a polynomial time algorithm that converts each input r to R to another input q to Q such that r is a yes-instance of P if and only if q is a yes-instance of Q.

Theorem: If R is polynomially reducible to Q and there is a polynomial time algorithm for Q, then there is a polynomial time algorithm for R.

Other Factoid: Polynomial reducibility is transitive.

Essence of NP-completeness

- We haven't a clue, so when we're given a problem we can't solve efficiently, we try to find whether it is equivalent to other problems we can't solve efficiently.
- Hundreds (thousands?) of equivalently hard NP-complete problems of immense practical importance. (scheduling, resource allocation, hardware design and test,……..)

Definitions, etc.

- **NP** : class of problems whose solutions can be verified in polynomial time.
- A problem P is **NP-hard** if every problem in NP is polynomially reducible to P.
- A problem P is **NP-complete** if
  - P belongs to NP
  - P is NP-hard

Proving a Problem is NP-complete

- Cook proved that there exist NP-complete problems (satisfiability).
- Once we have one, can start blazing.

To prove a problem Q is NP-complete

- show Q is in NP
- show R is polynomially reducible to Q, for some NP-complete problem R.

Remarkable Theorem of Steve Cook (1971)
Proved that there exist NP-complete problems

- The Satisfiability problem is NP-complete.
- Satisfiability: Given a boolean formula in conjunctive normal form (and of ors), find an assignment of the variables to 0's and 1's so that the value of the formula is 1.

- Example:

Idea of Proof

- In NP: easy.
- If a problem is in NP, there is a nondeterministic Turing machine that recognizes yes-instances.
- A Turing machine and all its operations on a given input can be described by a Boolean expression such that:
- Expression is satisfiable if and only if the Turing machine will terminate at an accepting state for given input.

  => any NP algorithm can be described by an instance of SAT

Traveling Salesman Problem

- Input description: A weighted graph G, L
- Output description: Is there a tour of length at most L that visits each of the vertices exactly once.
- Optimization version: minimize the length of the tour.

Clique

- Input description: A graph G=(V,E), k
- Problem description: Is there a subset S of V of size at least k such that for all x,y in S, (x,y) in E.

Optimization Version: Find maximum sized subset S.

Vertex Coloring

- Input description: A graph G=(V,E), k
- Problem description: Is it possible to color the vertices of the graph using at most k colors such that for each edge (i,j) in E, vertices i and j have different colors
- Optimization version: minimize the number of colors used.

Integer Linear Programming

- Input description: A linear functional **cx**, a set of linear constraints **Ax >= b**, a set of non-negative variables **x** that can take on only integer values, say 0 or 1, a value V.
- Problem description: Are there 0/1 integer values for the variables **x** satisfying the linear constraints such that the linear functional **cx** <= V.
- Optimization version: minimize V.

Techniques for Dealing with NP-complete Problems

- Exactly
  - backtracking , branch and bound
- Approximately
  - approximation algorithms with performance guarantees
  - using LP (e.g., randomized rounding)
  - local search, tabu search
  - simulated annealing
  - ….
- Change the problem.

Obtaining an Exact Solution:

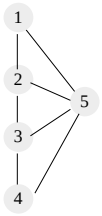
backtracking and branch-and-bound

Backtracking

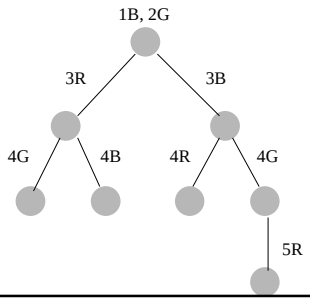
Example: Finding a 3-coloring of a graph

- Explore possibilities; backtrack when doesn't work.
- Start by assigning an arbitrary color to one of the vertices.
- Continue coloring while maintaining the constraints imposed by the edges
- If reach a vertex that can't be colored, backtrack -- go back up the recursion tree and explore other children

Graph



Recursion Tree



Branch-and-Bound

- Variation for case where finding minimum (or maximum) of objective function
- Example: finding the minimum number of colors needed to color graph.
- Idea: improve performance of algorithm by pruning search when know that can't possibly be going down the correct path.

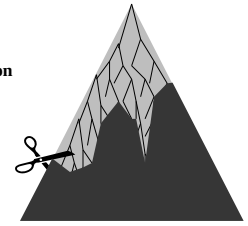
Example: Minimum number of colors in graph coloring

- Suppose traverse tree to leaf and find valid coloring with k colors
- Suppose later, after backtracking, reach a vertex that requires a (k+1)st color => can backtrack.
- In example, k serves as a bound for backtracking.

Branch-and-Bound

Pruning technique

Lower bound computation



How might you obtain such a bound?

- Example: looking for a maximum clique.
- Formulate problem as integer linear programming problem.
- N variables x_1,…. x_n corresponding to vertices
  - x_i = 1 if node v_i is in maximum clique, 0 otherwise
- Constraints: $x_i + x_j <= 1$ for each pair of vertices such that (v_i , v_j) an edge; x_i in {0,1} for every i
- Objective Function: maximize

Bounds for clique problem

- Use solution to corresponding linear program:
  - $0 <= x_i <= 1$
- What can you say about the solution v* of linear program?

Two things you can do with linear program

- Use it to get upper bounds on solution:
  - Example: at a node in tree with corresponds to including v,w in clique but excluding x, y.
  - If at that point solution of LP gives bound < size of already known clique, can backtrack.
- Can use it to help choose good traversal order => good solutions found fast => can backtrack earlier.
  - Example: if x_i=1 in linear program, might guess that v_i is in integer solution as well.

Approximation Algorithms With Provable

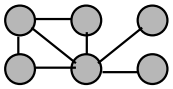
Performance Guarantees

Approximation Algorithms

- The fact that a problem is NP-complete doesn't mean that we can't find an approximate solution efficiently.
- Would really like guarantee on performance.

Example 1: Vertex Cover

- Given G=(V,E), find a minimum sized subset W of the vertices V such that for every (v,w) in E, at least one of v or w is in W.
- Approximation Algorithm:
  - Find maximal matching in graph.
  - Include both endpoints of every edge in the matching in the vertex cover



The Vertex Cover Produced Is At Most Twice The Size Of The Optimal VC

Proof:

Example 2: Approximation Algorithm for Euclidean Traveling Salesman Problem

The Problem: Given n points in the plane (corresponding to the locations of n cities) find a minimum-distance traveling salesman tour among them.

Distances in the plane satisfy the triangle inequality:

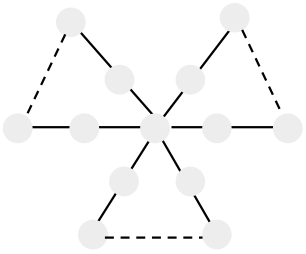
dist(a,b) <= dist(a,c) + dist(c,b)

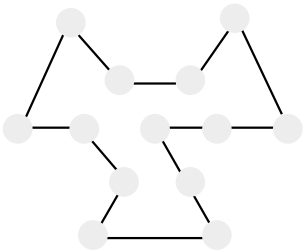
Approximation Algorithm For Euclidean TSP

- Find a minimum spanning tree of points
- Convert to tour by following DFS and including edge in opposite direction when tour backtracks.
- Convert to tour by taking direct routes instead of backtracking.

Better version of algorithm

- Find minimum length matching of odd-degree vertices.
  - There's an even number of them.
- Find Eulerian tour of MST plus edges in matching.
  - Eulerian tour: circuit in graph that contains each edge exactly once. (draw the graph without lifting the pencil.)
  - Can be found very easily if all vertices have even degree.



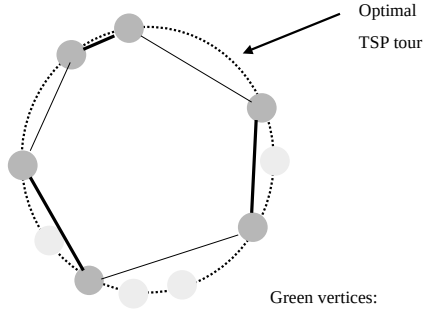


This algorithm has provably performance guarantee

Theorem: The approximation algorithm for Euclidean TSP finds a tour of length at most 3/2 optimal.

Proof:

- weight of MST <= weight of optimal tour
- weight of matching <= (weight of optimal tour)/2
- shortcuts don't cost



Randomized Approximation Algorithms:

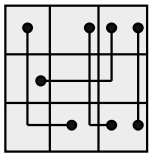
Using Linear Programming

Randomized Rounding

- Randomized algorithm: makes random choices during its execution.
- Randomized rounding: technique for using linear programming and randomization to produce approximate solution for integer programming problem.

Example: Global Wiring In A Gate Array

- 2D array of gates
- implement logic circuit by connecting some of gates using wires.
- **Net**: set of gates to be connected by wires running parallel to axes.



Wiring problem: given set of nets, specify physical path for each to minimize max # wires crossing any boundary.

Simplifications

- Each net = 2 gates
- Global route for each net contains at most one 90^o turn.
  - First horizontal, then vertical
  - first vertical, then horizontal
- Problem remains NP-complete with these simplifications.

Cast as Integer Program

- $h_i = 1$ if route for i-th net goes horizontally first, then vertically. 0 otherwise.
- $v_i = 1$ if route for i-th net goes vertically first, then horizontally.

For each boundary b in array:

- $B_0 = \{i \mid$ net i passes through b if $h_i = 1\}$
- $B_1 = \{i \mid$ net i passes through b if $v_i = 1\}$
- Constraints:

Randomized Rounding

- Solve linear programming relaxation:
  - gives you optimum: h_i^* ,v_i^* for each i and w^*
- What can you say about w^*?

- Set $h_i = 1$ with probability h_i^*
- Set $v_i = 1$ with probability v_i^*
- Value of resulting w provably close to w^*

Local Search

Local Search Algorithms

General Idea:

- Start with a solution (not necessarily good one)
- Repeatedly try to perform modifications to the current solution to improve it

Use simple local changes.

Local Search Procedure for TSP

Example: Start with TSP tour, repeatedly perform **Swap**, if it improves solution (Swap sometimes called 2-opt)

Call this Greedy Local Search

Does Greedy Local Search Lead Eventually To Optimal Tour?

No.

Solution Spaces

- Solution Space: set of all solutions to a search process and ways one can move from one solution to another.
- Represent process using a graph: a vertex for each possible solution, an edge from solution to solution if a local move can take you from one to other.
- Key question: how to choose moves. Art.
- Tradeoff between small neighborhoods and large neighborhoods.

Other Types of Local Moves For TSP Used

- 3-Opt

Tabu Search

Tabu Search

- Problem with local search procedure we just saw
  => can get stuck in a local optimum.
- To avoid this, perhaps should sometimes allow an operation that takes you to a worst solution.
- Hope is to escape the local optima and find global optimum.
- Tabu search paradigm: forbid certain moves.
- Example: backtrack, prohibit following same path again.

Augment Greedy Local Search on TSP with Ideas From Tabu Search Paradigm

- Maintain set L of all points visited.
- At each step, move to neighboring point of lower cost that is not in L, if one exists.
- Otherwise move to neighbor of higher cost that is in L.
- If get to situation where all neighbors of current position in L, start afresh from random solution.

Problems:

Limit Acceptance of Moves that Worsen Cost of Current Tour

- Let t be the number of local search steps so far.
- When t is small, be more willing to increase cost of tour.
- Get more and more reluctant as t increases.

An implementation:

- Let f(t) be decreasing function. Example: $f(t) = ae^{-bt}$
- c(t) cost of best tour found up to time t
- Move from node to neighbor of higher cost if:
  - not in L
  - cost increase at most f(t)c(t)

Simulated Annealing

Simulated Annealing

Analogy with thermodynamics:

Best crystals grown by “annealing” out their defects.

- First heat or melt material
- Then very very slowly cool to allow system to find its state of lowest energy.

Notation

- Solution Space X, x is a solution in X
- Energy (x) -- measure of how good a solution x is.
- Each x in X has a neighborhood.
- T -- temperature

- Cooling schedule: function for updating T. Typically,
  - power law: $a(1+bt)^{c}$
  - exponential decay: ae^{-bt}

Metropolis Algorithm

initialize T to hot, choose starting state
do
  generate a random move
  evaluate dE (change in energy)
  if (dE < 0) then accept the move
  else accept the move with probability
    proportional to $e^{-dE / kT}$
  update T
until T is frozen.

What's going on?

- For fixed T, probability of being in state x converges to $e^{-E(x)/T}$
- For small T, probability of being in low energy state:

- Very little known theoretically
- Widely used.

Some Details

- a -- initial tolerance parameter
- b -- scaling parameter, typically << 1
- parameter choices chosen by experimentation.

Termination Criteria:
- Limit the total number of steps.
- Step when there has been no improvement in cost of best tour in last m iterations.

Your turn to do an NP-completeness proof

- Subgraph isomorphism.

Your turn to analyze an approximation algorithm for an NP-complete problem.

Bin Packing: Let x_1 x_2 ,…, x_n be real numbers between 0 and 1. Partition them into as few subsets (bins) as possible such that the sum of numbers in each subset is at most 1.

Heuristic Solution: First Fit

- put x_1 in first bin
- then, for each i, put x_i in the first bin that has room for it, or start a new bin if there is no room in any of the used bins.

- Prove that First Fit requires at most twice the optimal number of bins.