CSE 589 Part VI

Reading

Skiena, Sections 5.5 and 6.8

CLR, chapter 37

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

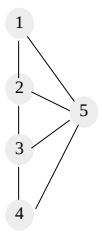
Explore possibilities; backtrack when doesn't work.

Start by assigning an arbitrary color to one of the vertices.

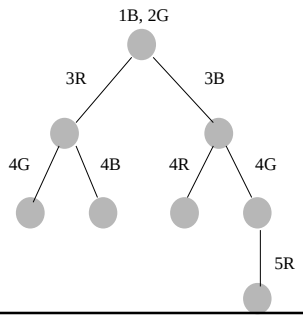
Continue coloring while maintaining the constraints imposed by the edges

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

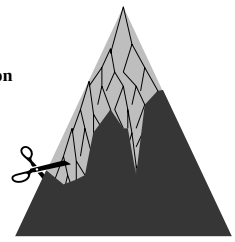
Suppose later, after backtracking, reach a vertex that requires a (k+1)st color => can backtrack.

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 $x_1, \ldots x_n$ 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) NOT 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:

- while edges remain
  - select an arbitrary edge (u,v)
  - add both u and v to the cover
  - delete all edges incident on either u or v

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

Proof:

consider the edges selected.

No two of these edges share a vertex

therefore, just considering these edges, any cover must include at least one vertex per edge.

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 shortest traveling salesman tour

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.

Construct shortcuts by taking direct routes instead of backtracking.

✓

Better version of algorithm

Uses basic graph algorithms as subroutines:
Matching (Skiena, Section 8.4.6)

- a matching in a graph G=(V,E): a set of edges S from E such that each vertex in V is incident to at most one edge of S.
- a maximum matching in G: a matching of maximum cardinality
- a minimum weight matching in a weighted graph: a maximum matching of minimum total weight.

Better version of algorithm

Euler tours (Skiena, Section 8.4.6)

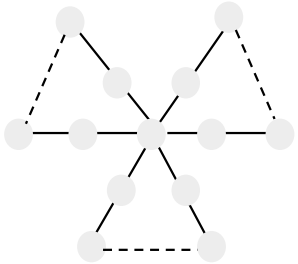
- An Euler tour in a graph is a tour of the graph that visits each edge exactly once.
- Well known that an undirected graph contains an Euler tour (or cycle) iff (1) it is connected and (2) each vertex has even degree.
- Easy to construct Euler tours efficiently.

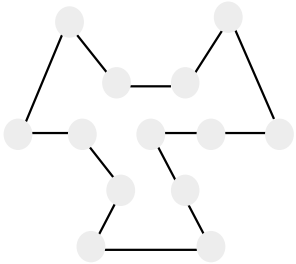
Better version of algorithm

Find minimum length matching of odd-degree vertices.

- There's an even number of them.

Find Eulerian tour of MST plus edges in matching.





✔

This algorithm has provably performance guarantee

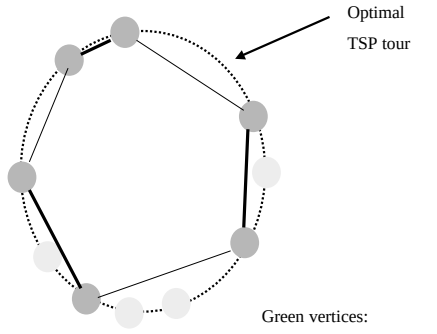
Theorem: The approximation algorithm for Euclidean TSP finds a tour of length at most 3/2 optimal.

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.

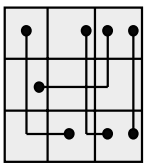
Example: Global Wiring In A Gate Array

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° 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 | net i passes through b if h_i = 1}

B_1 = {i | 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*

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.