Introduction to Algorithms December 6, 2003
Massachusetts Institute of Technology 6.046J/18.410J
Professors Shafi Goldwasser and Silvio Micali Handout 30

Problem Set 9

This problem set is not to be handed in.

Reading: Chapters §34, §35.

Problem 9-1. The challenges of industrial research

We say that a graph $G = (V, E)$ has a *spanning graph of degree* d if there exists a subset of edges $E' \subseteq E$ such that (1) $G' = (V, E')$ is a connected graph and (2) the degree of each vertex in G' is at most d. Such a $G' = (V, E')$ is called a degree d spanning graph of G. This problem has practical applications for telecommunications networks where nodes have bounded degrees.

You are working as an algorithms designer for a major telecommunications company. A co-worker of yours has been working on developing an efficient algorithm for the following search problem, SEARCH-SG: given a graph $G = (V, E)$ and an integer d, find a subset of edges $E' \subseteq E$ such that $G' = (V, E')$ is a degree d spanning graph of G, if one exists.

One day this co-worker leaves the company for a start-up, and the management asks you to take over his project where he has left off. It turns out that all he has been able to do so far is to design a magic box DECISION-SG$(< G, d >)$ that solves the following decision problem: on input a graph $G = (V, E)$ and an integer d, the magic box outputs 1 if G has a degree d spanning graph, and 0 otherwise.

(a) Assuming that the magic box that your co-worker designed runs in polynomial time, design a polynomial-time algorithm that solves SEARCH-SG. Prove that your algorithm is correct and runs in polynomial time.

(b) Suppose that the magic box that your co-worker designed has a bug and does not do anything meaningful at all. But another co-worker has designed a magic box for the 3-SAT decision problem. Assuming that her magic box runs in polynomial time, prove that there exists a polynomial-time algorithm that solves the smallest degree spanning graph problem.

Problem 9-2. Approximating knapsack

Recall from practice quiz 2 the problem of picking out, from a large pile of treasure, the best things you can fit in your knapsack. Specifically, the treasure consists of n objects $o_1, o_2, ..., o_n$. Object o_i has size s_i and is worth w_i, for each $i = 1, 2, ..., n$. Your backpack has size B, that is, can hold a number of objects whose total size is at most B. You want to carry home as much total worth as possible.

You might remember that on practice quiz 2 we asked you to give an efficient algorithm for this problem, assuming that the s_i, w_i and B were integers. It turns out that that method only works well when the values are small integers— recall that the running time of your algorithm was polynomial in B, not in the size of the encoding of B, which is $\lg B$, —in general the problem is NP-hard!

(a) Show that the knapsack problem is NP-hard by a reduction from the SUBSET-SUM problem in the book.

Probably you're not going to leave all the treasure sitting there just because you can't easily figure out how to get the absolute best total worth. So let's look for an approximation algorithm, that is, one guaranteed to get most of the total worth you can hope for.

(b) One obvious algorithm to try is a greedy algorithm: sort the objects in decreasing order of w_i/s_i and go through the list taking an object if it still fits. Show that there are inputs on which this algorithm might give you an arbitrarily small fraction of the optimal total worth.

(c) Let's try a slight improvement to part (a): run the greedy algorithm, and find out what total worth it would give you. Compare this value to the single most valuable object that fits in your knapsack. If the single object is better, take only the single object. Otherwise take the greedy solution. Show that this is a 2-approximation algorithm. (Hint: Reason about the fractional solution—i.e. a solution you would get if you were allowed to split an object—at the time the greedy algorithm first rejects an object.)

Problem 9-3. Reductions

Suppose that L_1 and L_2 are languages such that $L_1 <_p L_2$. For each of the following statements, either prove that it is true, or prove that it is false, or give a convincing justification for why it is an open question.

(a) Let $L_1, L_2 \in NP$. If $L_1 \in P$, then $L_2 \in P$.

(b) If $L_2 \in P$, then $L_1 \in P$.

(c) Let $L_2 \in NP$. If L_2 is not NP-complete, then neither is L_1.

(d) Let $L_1, L_2 \in NP$. If $L_2 <_p L_1$, then both L_1 and L_2 are NP-complete.

(e) If L_1 and L_2 are both NP-complete, then $L_2 <_p L_1$.

Problem 9-4. Easy vs. hard cases of the same problem

The **directed Hamiltonian path** problem is as follows: given a directed graph $G = (V, E)$ and two distinct vertices $u, v \in V$, determine whether G contains a path that starts at u, ends at v, and visits every vertex of the graph exactly once.

(a) Prove that the directed Hamiltonian path problem is NP-complete. (Hint: consider the Hamiltonian cycle problem described in the book.)

(b) Design a polynomial-time algorithm that solves the directed Hamiltonian path problem for directed acyclic graphs.