CS 125 Algorithms & Complexity — Fall 2016

Problem Set 11

Due: 11:59pm, Wednesday, November 30th

See homework submission instructions at `http://seas.harvard.edu/~cs125/fall16/schedule.htm`

Problem 1

If we restrict the problems we look at, sometimes hard problems like counting the number of independent sets are in a graph become solvable.

(a) (3 points) Consider a graph that is a path on n vertices. (That is, the vertices are labelled 1 to n, and there is an edge from 1 to 2, 2 to 3, etc.) How many independent sets are there as a function of n? We want to express your answer in terms of a family of numbers – like "For n vertices the number of independent sets is the nth prime." (note: that is not the answer).

(b) (3 points) How many independent sets are there on a cycle of n vertices?

(c) (4 points) How many independent sets are there on a complete binary tree with 127 nodes? Describe how you arrived at this number.

Problem 2

Consider the problem MAX-k-CUT, which is like the MAX CUT algorithm, except that we divide the vertices into k disjoint sets, and we want to maximize the number of edges between sets. Explain how to generalize both the randomized and the local search algorithms for MAX CUT to MAX-k-CUT and prove bounds on their performance.

Problem 3

We know that that all of NP-complete reduce to each other. It would be nice if this meant that an approximation for one NP-hard problem would lead to another. But this is not the case. Consider the case of Vertex Cover, for which we have a polynomial-time 2-approximation algorithm.

Another NP-complete optimization problem is Independent Set: given a graph $G = (V, E)$, find as large a set $S \subset V$ as possible such that no two vertices $u, v \in S$ share an edge. In particular, $S \subset V$ is a vertex cover iff its complement $V - S$ is an independent set.

(a) (4 points) Explain why applying the above equivalence to the 2-approximation for Vertex Cover does not yield a constant-factor approximation algorithm for Independent Set. That is, show that for every constant $c \in (0,1)$, there exists a family of graphs (growing so that the number of vertices/edges grows to infinity) for which even if we obtain a 2-approximation of the minimum vertex cover, the corresponding independent set is not within a factor of c of the maximum independent set.

(b) (6 points) Using the PCP Theorem and a variant of the standard NP-completeness reductions from 3-SAT, it can be shown that both Independent Set and Vertex Cover are NP-hard to approximate to within factors $1 - \epsilon_1$ and $1 + \epsilon_2$, respectively, for some constants $\epsilon_1, \epsilon_2 > 0$. Deduce from this that Independent Set is NP-hard to approximate to within any constant factor $\alpha \in (0,1)$ by, given a graph G, considering the graph G_k with vertex set $V_k = V^k$ and edge set $E_k = \{((u_1, \ldots, u_k), (v_1, \ldots, v_k)) : \exists i \ (u_i, v_i) \in E\}$. How does the size of the maximum independent set in G_k relate to that in G? Why doesn't the same reduction apply to Vertex Cover?

Problem 4

Sometimes it is worth running an approximation algorithm to solve a problem not because the problem is **NP**-hard, but because the fastest known polynomial time algorithm we have to solve isn't as fast as we would like!

Consider for example Problem 4 from Problem Set 3 of this class (optimal layout of the nodes of a trie on blocks on disk, with block size B). You saw in that problem set that if there is a probability distribution over queries to the leaves of an n-node trie, there is a dynamic programming algorithm running in time $O(nB^2)$ which finds a layout that minimizes the expected cost of a query. (In problem set 3 it was assumed that queries could to be any node, but only querying leaves is a special case since that just corresponds to instances for which internal nodes have probability 0 of being queried.)

Suppose the optimal layout achieves an expected query cost of OPT. Give an algorithm with improved running time $O(nB)$ which finds a layout guaranteed to achieve expected query cost at most $\mathsf{OPT} + 1$. That is, on average we only have to touch one more disk block than the optimal solution. **Hint:** reduce to the case that the input trie has at most n/B leaves, suffering at most one unnecessary block transfer in your reduction.