

CS 222A Fall 2007 HW. 5, Due Friday Nov. 16

1. Recall the recurrences for the non-crossing matching problem from the midterm.

Then, for $i > 0$ and $j > 0$, $M(i, j) = \text{Max} [M(i - 1, j - 1) + 1 \text{ if edge } (i, j) \text{ exists}; M(i - 1, j - 1); M(i, j - 1); M(i - 1, j)]$. The base cases are $M(0, j) = M(i, 0) = 0$.

Many people justified these recurrences by something like the following argument:

There are four options in calculating $M(i, j)$.

1) Don't use node i of A, but do use node j of B, and the best matching of that type has $M(i - 1, j)$ edges.

2) Don't use node j of B, but do use node i of A, and the best matching of that type has $M(i, j - 1)$ edges.

3) Don't use either node i of A or node j of B, and the best matching of that type has $M(i - 1, j - 1)$ edges.

4) If (i, j) is an edge, then use it in the matching and the best matching of that type has $M(i - 1, j - 1) + 1$ edges.

But in this argument, there is no mention of the non-crossing condition. If we want to establish that these recurrences correctly compute the largest non-crossing matching, the correctness argument must use the non-crossing condition somewhere, otherwise the arguments would establish that the recurrences compute the largest matching, even allowing crossing edges.

Problem 1a) Do these recurrences correctly compute the largest matching, even allowing crossing edges? If not, why not?

Problem 1b) Complete or correct the above justification to show that the recurrences correctly compute the size of the largest non-crossing matching.

2. The book, on Page 559, develops a greedy algorithm for the (unweighted) independent set problem when the graph is restricted to be a tree. Is there a greedy algorithm for the unweighted node cover problem? If so, describe it as briefly as possible, given whatever you know from the class.

3. In class we stated that the Satisfiability problem is NP-complete, that the Independent Set problem is NP-complete and the Node-Cover Problem is NP-complete. So in this problem you may assume those problems, but only those problems, are known to be NP-complete.

3a. In problem ZZZ, the input is a number k , and bipartite graph G , where the two node sets on the two sides of G are denoted A and B. The

answer is yes if and only if there is a subset S of size at most k of the nodes in A , such that every node in B is adjacent to at least one node in S . Prove that problem ZZZ is NP-complete.

3b. In problem QQQ, the input is an undirected graph $G = (V, E)$ and an undirected graph G_1 . There are no node or edge labels. The answer to problem QQQ is yes if and only if there is an “induced” subgraph $G' = (V', E')$ of G which is isomorphic (identical in this context) to G_1 . In an induced subgraph containing the set of nodes V' , the edge set E' consists of *every* edge whose two endpoints are both in V' . Prove that Problem QQQ is NP-complete.

4. Weighted Node Cover: Approximation Algorithms Based on Network Flow

In this problem we consider a common approach that is taken to problems which are known to be NP-hard. We will illustrate the approach with the node cover problem, and a polynomial-time approximation algorithm for it based on network flow.

Recall the node cover problem

Let G be an undirected graph with each node i given weight $w(i) > 0$. A set of nodes S is a *node cover* of G if every edge of G is incident to at least one node of S . The *weight* of a node cover S is the summation of the weights, denoted $w(S)$, of the nodes in S ; the weighted node cover problem is to select a node cover with minimum weight.

The node cover problem (even when all weights are one) is known to be NP-hard, and hence we do not expect to find a deterministic, polynomial-time (in terms of worst case) algorithm that is always correct. Therefore, we relax somewhat the insistence that the method be both correct and efficient for all problem instances. There are many types of relaxations that have been developed for NP-hard problems. The most common is the constant-factor, polynomial-time *approximation* algorithm.

For a graph G with node weights, let $S^*(G)$ denote the minimum weight node cover. Let A be a polynomial time algorithm that always finds a node cover, but one that is not necessarily minimum; let $S(G)$ denote the node cover of G that A finds. Then A is called a *constant-error polynomial-time approximation algorithm* (or approximation algorithm for short) if for any graph G , $S(G)/S^*(G) \leq c$ for some fixed constant c .

For the node cover problem we will give an approximation algorithm, based on network flow, with $c = 2$. First, recall that the weighted node cover problem has a polynomial-time solution when the graph G is bipartite. This was shown in a prior homework. You may take it as a black box at this point.

The approximation algorithm for general graphs

Given G (not necessarily bipartite), create bipartite graph $B = (N, N', E)$ as follows: for each node i in G , create two nodes i and i' , placing i on the N side, and i' on the N' side of B ; give both of these nodes the weight $w(i)$ of the original node i in G . If (i, j) is an edge in G , create an edge in B from i to j' and one from j to i' . Now find a minimum cost node cover $S^*(B)$ of graph B . From $S^*(B)$, create a node cover $S(G)$ in G as follows: for any node i in G , if either i or i' is in $S^*(B)$, then put i in $S(G)$.

It is easy to find examples where $S(G)$ is not a minimum node cover of G , and where $S(G)/S^*(G) = 2$.

4a. Show such an example.

However, no worse error ever happens.

Theorem $S(G)/S^*(G) \leq 2$ for any G and any choice of node weights for G .

4b. Prove (that is, explain completely) the theorem.