Homework D (15 pts)

1. [5 pts]. Consider the “alternate” approximation algorithm for the Vertex Cover problem: repeatedly add the vertex to the cover that is incident on (that is, covers) the greatest number of uncovered edges.

Either argue that like the approximation algorithm from the text and notes, this is a 2-approximation of the optimal vertex cover, or show that it is not, possibly by counterexample.

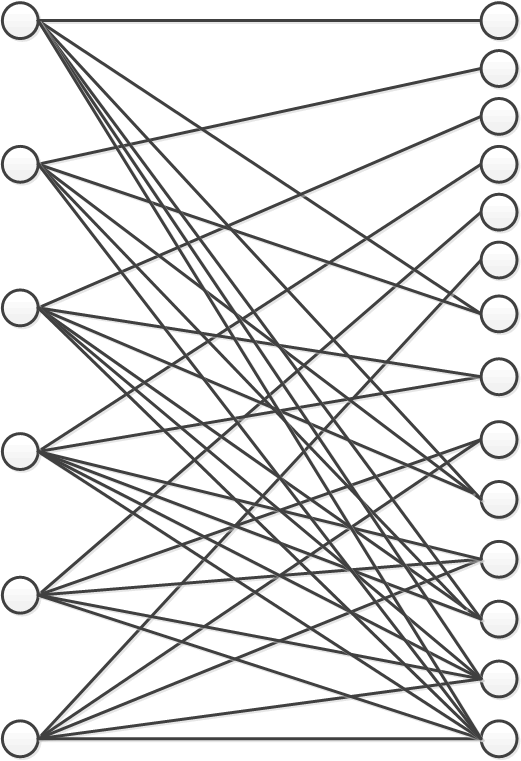
***Comment:***

I’m giving the comment before the solution. This is a fun problem, one that I remember doing. The answer is that this is not a 2-approximation, and there is a counterexample with bipartite graphs.

I remembered the counterexample as being fairly small, but I couldn’t come up with it! ☹ (I spent a couple hours trying.) I finally gave up and checked on the internet. I found it in about 5 minutes. It’s a larger counterexample than I had remembered. One such instance of the counterexample is on the next page. Unfortunately, the picture is the smallest one for which the counterexample holds. The explanation follows the picture.

You can find this a number of places on the internet. For instance, <https://cgi.csc.liv.ac.uk/~michele/TEACHING/COMP309/2005/Lec10.4.4.pdf> calls it the “Clever Greedy Algorithm.” Or <https://courses.engr.illinois.edu/cs573/fa2013/lec/lec/06_approx_I.pdf>

The internet is a wonderful resource.



***Solution:***

The lower right node, and some of the left nodes, have 6 edges. If you chose to put the lowest right node into the cover, and erase the covered edges, then the 2nd lowest right node and some of the left nodes have 5 uncovered edges. If you chose to put the 2nd lowest right node into the cover, and erase the covered edges, then the 3nd lowest right node and some of the left nodes have 4 uncovered edges. Keep going up the right side. When you get to the top 6 nodes on the right side, all of them connect to a single different left side node, so all of them go into the cover. So one possible vertex cover according to this algorithm puts every right side node into the cover. That’s 14 nodes. Clearly, you could create a cover out of all the left side nodes, for a cover of size 6. 14 > 2\*6, so this is not a 2-approximation.

1. [10 pts] Suppose you have *n* processors, call then p1, p2, …, pn. And suppose you have *m* jobs to be assigned to these *n* processors, call them j1, j2, …, jm. Job jk takes time tk. Your goal is to minimize the time until all tasks are completed. This job scheduling problem is known to be NP-Complete.

Part A: [4 pts] Propose a polynomial time approximation algorithm for this problem. [[1]](#footnote-1)

***Solution:***

Assume m > n, otherwise you just assign one job to each processor (maybe leaving some free.) One way to approximate this would be assign the longest job to the first processor, the second longest job to the next processor, etc. and whenever a processor frees up, assign the longest job in the queue to it.

Part B: [3 pts] Explain in a few sentences or less why this is likely to be a good approximation.

***Solution:***

You’re keeping all of the processors fully occupied, and by assigning the longest items first, you are making sure that you don’t assign two very long jobs to the same processor.

Part C: [3 pts] Analyze the worst-case running time of your algorithm in terms of *n* and *m*.

***Solution:***

This is a polynomial time algorithm, because you sort the jobs first in time O(m lg m), then you just assign the jobs to processors in the sorted order. Since there are m jobs, this is O(m). So the algorithm is an O(m lg m) algorithm.

1. To be clear, in this case “polynomial time” means that the time to run the approximation algorithm to figure out the approximately optimal schedule is polynomial in n and m. [↑](#footnote-ref-1)