Nalongsone Danddank Student ID : 14958950 StarID: jf3893pd

Email: [nalongsone.danddank@my.metrostate.edu](mailto:nalongsone.danddank@my.metrostate.edu)

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 is is not, possibly by counterexample.

Answer: This is not a 2-approximation of the optimal vertex cover. Because, First before we repeatedly add the vertex to the cover that is incident on the greatest number of uncovered edges, we have to sort vertex by number of uncovered edges, that we have to runs in n\*log n time. Then we repeatedly add the vertex to the cover by running n time and total time runnig is O( n\*n\*log n), p(n) >2. In this way will not be the best way for optimal vertex cover.

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-0)

Answer: I would try to assigned each job jk by random takes a time tk to a random processor pk  that is not duplicate,until all tasks are completed.

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

Answer: when all tasks is assigned by random, each time we process the job scheduling, because the random selection, some time we could get lucky or not. their average time complexity is same.

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

Answer: The worst-case running time of this algorithm is, when the random selection select the longest time job to process and let the second shortest time job wait for the same processor, that mean (m+(m-1)+(m-2)+…+1)/n = O(m2/n).

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-0)