

Approximation Algorithms Practice Problems

October 12, 2004

Here are practice problems on approximation algorithms. The solutions can also be found on the course web page.

1. An **independent set** of a graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices such that each edge in E is incident on at most one vertex in V' . The **independent set problem** is to find a maximum-size independent set in G . Recall that a **clique** of a graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices, each pair of which is connected by an edge in E . The **clique problem** is to find a maximum-size clique in G .

Suppose that you were given an approximation algorithm A for the independent set problem that has a ratio bound of two. Describe how you can use A to obtain a 2-approximation algorithm for the clique problem.

2. In proving that $\text{CLIQUE} \leq_p \text{VERTEX-COVER}$ for a given the input $\langle G = (V, E), k \rangle$ for CLIQUE we converted it to the input $\langle \overline{G}, |V| - k \rangle$ for VERTEX-COVER. We then proved that G has a clique of k vertices if and only if \overline{G} has a vertex cover of $|V| - k$ vertices.

Recall that we have a 2-approximation algorithm, call it A , for the optimization version of the vertex cover problem. We now propose a scheme to use A to create an approximation algorithm for clique (that is, given input G , the goal is to find a set of vertices that form a maximum-sized clique). First compute \overline{G} . Next run A , the vertex cover approximation algorithm, on \overline{G} to obtain the approximate vertex cover $V' \subseteq V$. Then output $V - V'$ (i.e. all vertices not in V') as the clique approximation.

Let $C_*(G)$ be the size of the maximum clique in G and let $C(G)$ be the size of the clique output by the above approximation algorithm. Do ONE of the following two tasks:

- Give the smallest constant c for which you can prove that for all inputs G to CLIQUE that $\frac{C_*(G)}{C(G)} \leq c$. That is, prove that the resulting approximation has an approximation ratio of c .
 - Prove that for any constant c there exists an input G for which $\frac{C_*(G)}{C(G)} > c$.
3. In this problem we consider the optimization version of the set covering problem where your goal is to find a subset F of \mathcal{F} of minimum cardinality for which $\cup_{f \in F} f = X$. In this problem you are to design an approximation algorithm that has a ratio bound of 3 for a special case of the set covering problem where each object is contained in at most 3 sets. Be sure to clearly state your algorithm (code not required), argue that it runs in polynomial time, and prove that it has a ratio bound of 3.

Hint: Think about the relationship between the set cover problem and the vertex cover problem discussed in class along with the approximation algorithm for the optimization version of the vertex cover problem.

4. Consider the following problem. Each job consists of a set of *operations* whose processing cannot overlap in time. The set of all operations is denoted $\{O_1, \dots, O_k\}$; each operation O_k belongs to

some job J_j and must be processed on a specific machine M_i . However, the operations of a job can be processed in any order. The goal is to minimize the time at which all operations have been completed.

You are to give a 2-approximation algorithm for this problem. (Be sure to clearly describe your algorithm and then prove that it has an approximation ratio of 2.)

Hint: The algorithm need not be very sophisticated. Here are some definitions that you might find useful. Let P_{max} be the maximum processing time required by any job and let Π_{max} be the maximum processing time required on any machine. Finally, let M_i be the machine that finishes last in the approximate solution and let J_j be the job whose operation O_k was the last to run on machine M_i .

5. Consider the following scheduling problem. You are given n jobs where job i is specified by an earliest start time s_i and a processing time p_i . In homework 1, we considered a preemptive version of this problem and gave a greedy algorithm to give an optimal preemptive schedule.

In this problem we consider the non-preemptive version of this scheduling problem. Here a job CANNOT be suspended but rather must be performed in a contiguous time interval. Consider the following heuristic for the non-preemptive problem: schedule the jobs in the order in which they complete in an optimal preemptive schedule starting each job as soon as the one before it completes. You are to prove that this algorithm is a 2-approximation algorithm.