Approximation Algorithms Lecture 3

Contents

1 Recap

2 Content

Definition 1

SET-COVER: Given a universe $U = \{e_i\}_{i=1}^{n}$, and sets $S_1, S_2, \cdots, S_n$ called the set system, where $S_i \subseteq U$ (and each element is in at least one S_i). Find a minimum collection of sets which cover all elements, i.e., the union of all such sets is U.

This is a NP-hard problem. So we look for an approximation algorithm (more specifically, a greedy algorithm) as follows:

1. Pick the set which covers the most elements in U.
2. Remove covered elements from U.
3. Repeat until U is empty.

Another way of saying the same thing: we pick the set which cover the most number of uncovered elements.

An example where this doesn't give us the optimum solution:

Let $U = \{i\}_{i=1}^{16}$, and sets $\{1, 2, 3, 4, 5, 6, 7\}$, $\{9, 10, 11, 12, 13, 14, 15\}$, $\{1, 2, 3, 4, 9, 10, 11, 12\}$, $\{5, 6, 13, 14\}$, $\{7, 15\}$, $\{8, 16\}$. (Example done in class had $8, 16$ in the first two sets as well, respectively)

Analyzing the greedy algorithm

We will associate a value in $[0, 1]$ with each elemebt (v_i is the value of element e_i) such that the sum of values of elements in any set is at most 1.

Claim 0.1

The sum of the values in any feasible assignment is at most the value of the optimum solution. Equivalently, the sum of values on elements is a lower bound on the value of the optimum solution.

Proof. Consider the optimum solution. Then the sum of values in each set is ≤ 1. The sum of elements is upper bounded by the sum of "values" of each set, which is at most the number of sets, whence we are done. □

Note that we did this since while analyzing approximation algorithms, we need to compare our solution and the optimum solution. Finding the optimum value in itself is NP-hard in most cases. So for a minimization problem, we compare with a lower bound on **OPT**. Ideally we want a good lower bound so that our bounding gives us a good bound on the competitive ratio.

So we need an assignment that tries to maximize the sum of elements. So we try to assign the values to help in the analysis (not a part of the algorithm).

1. If at step i, we pick a set which covers u_i uncovered elements, assign each of these elements a value of $\frac{1}{u_i}$. This is infeasible, but we will fix this later.

From here, we get to know that the sum of values of all elements equals the number of sets picked by the greedy algorithm.

Question 1

How infeasible is this assignment? Note that the property should hold for all sets, not just this set.

Say $S = \{e_1, e_2, \ldots, e_k\}$ is some set. Suppose the elements of S are covered by the greedy algorithm in the order $e_1, e_2, \ldots, e_k$ (else rename the elements).

Suppose e_1 was picked. Then the value of e_1 is at most $\frac{1}{k}$, since if it was more, we could have picked S at that step rather than the one picked by the greedy algorithm. Then suppose e_2, e_3 were picked, in that case the value of each is at most $\frac{1}{k-1}$. Doing this, we get that the value of e_i is at most $\frac{1}{k-i+1}$. So the sum of values of elements in S is at most $\sum_{i=1}^{k} \frac{1}{i} \leq 1 + \ln k$.

So we can scale down all values by $1 + \ln k$ to get a feasible assignment (actually scale by the maximum size of any set of S).

Hence $n_i = \frac{v_i}{1+\ln k}$ is a feasible assignment.

$\sum_{i=1}^{m} n_i = \frac{\text{number of sets picked by the greedy algorithm}}{1+\ln k} \leq \mathbf{OPT}$.

To show that we can't get a better factor using the greedy algorithm, we can construct an example similar to the original example to obtain an instance where $\mathbf{OPT} = 2$ and greedy solution gives us $\log_2 n$ sets.