CSE421: Design and Analysis of Algorithms

Lecturer: Shayan Oveis Gharan Lecture 16 Approximation Algorithms for Set Cover

## Set Cover 1

We now design an approximation algorithm for the set cover problem.

Recall  $[n] = \{1, \ldots, n\}$ . You are given a collection of sets  $S_1, \ldots, S_m \subseteq [n]$ , such that  $\bigcup_i S_i = [n]$ . The goal is to find the smallest subcollection that includes all the elements. The set cover problem is a generalization of the vertex cover problem. You can think of each vertex as a set of its connecting edges.

The problem has many applications in practice. For example, think of the a startup who needs a number skills including marketing, software developing, accounting, data science, design, UI, etc. Each applicant may have a number of these skills. The startup wants to hire a minimum number of these applicants to include all the critical skills that it needs. There is also a natural weighted variant of the problem where each set has a weight and we want to choose a subcollection of the sets with the smallest weight.

Consider the following greedy algorithm. We show that its approximation ratio is at most  $\ln n$ .

**Input:** A collection of sets  $S_1, \ldots, S_m \subseteq [n]$ , such that  $\cup_i S_i = [n]$ **Result:** A small collection of sets whose union covers [n]. Let  $T = \emptyset$ ; while  $\cup_{i \in T} S_i \neq [n]$  do  $| \text{ If } S_j \text{ maximizes } S_j \cap ([n] - \bigcup_{i \in T} S_i), \text{ add } j \text{ to } T;$ end Output T. Algorithm 1: Greedy Set Cover algorithm

**Claim 1.** If the smallest cover has k sets, then the algorithm finds a cover with at most  $k \ln n$  sets.

**Proof** Suppose the OPT has k sets. Consider an iteration i of the while loop. Let  $R = [n] - \bigcup_{i \in T} S_i$ be the set of remaining elements. Note that  $R \subseteq [n]$ . Since OPT covers [n] it also covers R with k sets. Therefore, there must be a set in OPT that covers at least 1/k fraction of elements of R. Since Greedy chooses the set that covers the largest fraction of elements of R, the set that Greedy chooses also covers at least 1/k fraction of elements of R.

Now, let us calculate how the number of remaining elements changes over the iterations of the algorithm. At the beginning we have n. After 1 iteration (at least) n/k elements are covered so we have at most n(1-1/k) elements. In the second iteration (at least)  $\frac{n(1-1/k)}{k}$  elements are covered so we will have (at most)

$$n(1-1/k) - \frac{n(1-1/k)}{k} = n(1-1/k)(1-1/k) = n(1-1/k)^2.$$

16 Approximation Algorithms for Set Cover-1

Similarly, after the *i*-th iteration of the while loop at most  $n(1 - 1/k)^i$  elements are remained. Observe that we will definitely stop (and cover everything) when  $n(1 - 1/k)^i < 1$  or equivalently, when  $(1 - 1/k)^i < 1/n$ .

So, the question is how large *i* should be such that  $(1-1/k)^i < 1/n$ . Here we use the following inequality without proof: For all  $x \ge 0$ ,

$$1 - x \le e^{-x}.$$

This can be proven by writing down the taylor series expansion of the exponential function. It follows that

$$(1-1/k)^i \le e^{-i/k}.$$

So, for  $i = k \ln n$  we have

$$(1 - k)^i \le e^{-k \ln n/k} = e^{-\ln n} = 1/n$$

as desired.  $\blacksquare$ 

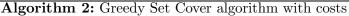
The above analysis for the algorithm is in fact tight. To see this, suppose the *n* elements are party of *k* disjoint sets  $S_1, \ldots, S_k$ , where the *i*'th set has exactly  $2^i$  elements. Thus  $n = 2 + 4 + \ldots + 2^k = 2^{k+1} - 2$ . Now add two more sets A, B which are disjoint. A contains half of the elements of every  $S_i$ , and *B* contains the other half. So  $|A| = |B| = 2^k - 1$ . The algorithm will pick the *k* sets  $S_1, \ldots, S_k$  as the set cover, even though A, B are also a set cover.

No better efficient algorithm is known for this problem. In fact, it is proven to be impossible to break the  $\Theta(\log n)$  approximation ratio assuming NP  $\neq$  P.

## 2 Weighted Set Cover

One can easily modify the above algorithm to handle the situation where each set has a cost. We pick the set that has the lowest per element cost in each step:

Input: A collection of sets  $S_1, \ldots, S_m \subseteq [n]$ , such that  $\bigcup_i S_i = [n]$ , for each set a cost  $c(S_i) \ge 0$ . Result: A collection of sets whose union covers [n] of small total cost. Let  $T = \emptyset$ ; while  $\bigcup_{i \in T} S_i \neq [n]$  do  $| \text{ If } S_j \text{ minimizes } \frac{c(S_j)}{|S_j \cap ([n] - \bigcup_{i \in T} S_i)|}$ , add j to T; end Output T.



**Claim 2.** If the smallest cover has cost C then the algorithm finds a cover of cost at most  $O(C \log n)$ .

**Proof** For every element *i*, define  $e_i$  to be the price of covering *i*, as follows. At the point  $e_i$  is covered, let *L* be the set of elements that have already been covered, and let  $S_j$  be the set containing *i* that is about to be added to the cover. Set  $e_i = c(S_j)/|S_j - L|$ .

16 Approximation Algorithms for Set Cover-2

Then we see that the total cost of the solution computed by the algorithm is exactly  $\sum_{i=1}^{n} e_i$ . Now observe that if *i* is covered in the *j*'th step of the algorithm, then at this point at least j-1 elements have been covered. Moreover, since there is cover of cost *C*, there must be some set that covers elements at a price of C/(n-j+1), and the algorithm picks the set that covers elements at the lowest price. Thus the total cost of the solution found by the algorithm is at most  $\sum_{i=1}^{n} e_i \leq C(1+1/2+1/3+\ldots+1/n) \leq O(C \log n)$ .