CMSC 451: Lecture 7
Greedy Approximation: Set Cover

Set Cover: An important class of optimization problems involves covering a certain domain with minimum number of sets. Covering problems arise in many applications of science and engineering:

Surveilance: You want to place cameras in an environment to monitor a given region. Each possible camera position can view a certain region. Cameras are expensive to place and operate. What is the minimum number of camera placements to monitor the entire environment?

Wireless Coverage: You want to place wireless routers around a college campus so that every location is within range of some transmitter. Coverage regions vary due to the presence of obstacles. What is the minimum number of transmitters are needed to cover the entire campus?

Workforce scheduling: You are the manager of a large service center. Workers have various constraints on where/when they can work. Your job is to assign a minimum number of workers to cover all the stations/hours needed.

Many of these problems can be expressed abstractly as the *set cover problem*. We are given a pair $\Sigma = (X, S)$, called a *set system*, where $X = \{x_1, \ldots, x_n\}$ is a finite set of objects, called the *universe*. This is the domain to be covered. $S = \{s_1, \ldots, s_m\}$ is a collection of subsets of X. We assume that every element of X belongs to at least one set of S. Throughout, let $n = |X|$ and and $m = |S|$.

Given such a set system (X, S), the set cover problem is to determine the smallest number of sets of S needed to cover X. For example, in Fig. 1(a), the elements of X are the 12 black circles, and the sets $s_1, \ldots, s_6$ are indicated by the blue regions. In this case there exists a cover of size 3, consisting of s_3, s_4, and s_5 (see Fig. 1(b)). (In this case, the sets of this cover do not overlap. This is called an *exact cover*. In general, the sets of the cover are allowed to overlap.)

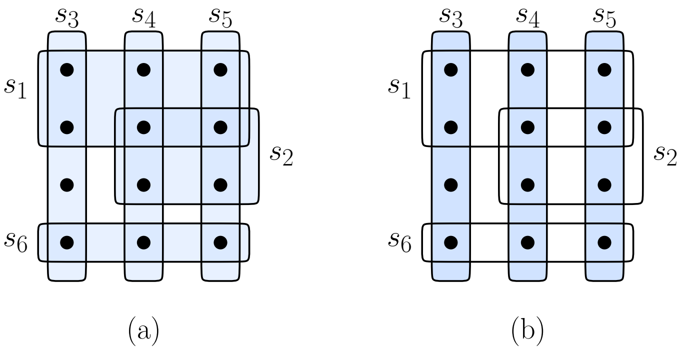

Fig. 1: (a) A set system consisting of 12 elements and 6 sets, and (b) an optimum cover consisting of the three sets $\{s_3, s_4, s_5\}$.

Notice that the output of set cover is not a set, but rather a set of sets. If we think of the sets of S as being indexed by the integers from 1 to m, then we can think of a cover C more conveniently as a subset of $\{1, \ldots, m\}$. This suggests the following definition.

Set Cover Problem: Given a set system $\Sigma = (X, S)$, where $S = \{s_1, \ldots, s_m\}$, compute a set $C \subseteq \{1, \ldots, m\}$ of minimum cardinality such that

$$X = \bigcup_{i \in C} s_i$$

A more general formulation is a weighted variant, in which each set s_i is associated with a positive weight w_i, which can be thought of as the "cost" of including set s_i. The problem is to compute the set cover of *minimum total weight*. Our simpler version is equivalent to setting $w_i = 1$ for all i.

Set Cover Approximation: Set Cover is a very useful optimization problem, but it is known to be *NP-hard*. We will present a simple *greedy heuristic* for this problem, and we will show that this heuristic leads to an approximation. As we shall see, the approximation factor is rather weak, but there are reasons to believe that significantly better approximations are not easy to compute. Recall that $n = |X|$. We will show that the size of the greedy heurisitc exceeds the size of an optimum cover by a factor of at most $\ln n$, the natural logarithm of n.

It is worth noting that, in a number of practical instances (including the geometric covering examples mentioned above), the greedy heuristic performs reasonably well. Unfortunately, we will show that it is possible to construct set systems where greedy's performance is $\Omega(\log n)$.

Greedy Set Cover: A simple greedy approach to set cover works by at each stage selecting the set that covers the greatest number of uncovered elements. The algorithm is presented in the code block below. The set C contains the indices of the sets of the cover, and the set U stores the elements of X that are still uncovered. Initially, C is empty and $U \leftarrow X$. We repeatedly select the set of S that covers the greatest number of elements of U and add it to the cover.

Greedy Set Cover

```
greedy-Set-Cover(X, S) {
    U = X                               // U stores the uncovered items
    C = empty-set                       // C stores the sets of the cover
    while (U is nonempty) {
        select s[i] in S that covers the most elements of U
        add i to C
        remove the elements of s[i] from U
    }
    return C
}
```

An example is shown in Fig. 2. Set s_1 has the most elements, six, and is added first (see Fig. 2(a)). Next, s_6 has the most uncovered elements, three (see Fig. 2(b)). Finally, s_2 has the most uncovered elements, two (see Fig. 2(c)). Finally, s_3 covers the only remaining element (see Fig. 2(d)). Thus, it would return a set cover of size 4, whereas the optimal set cover has size 3.

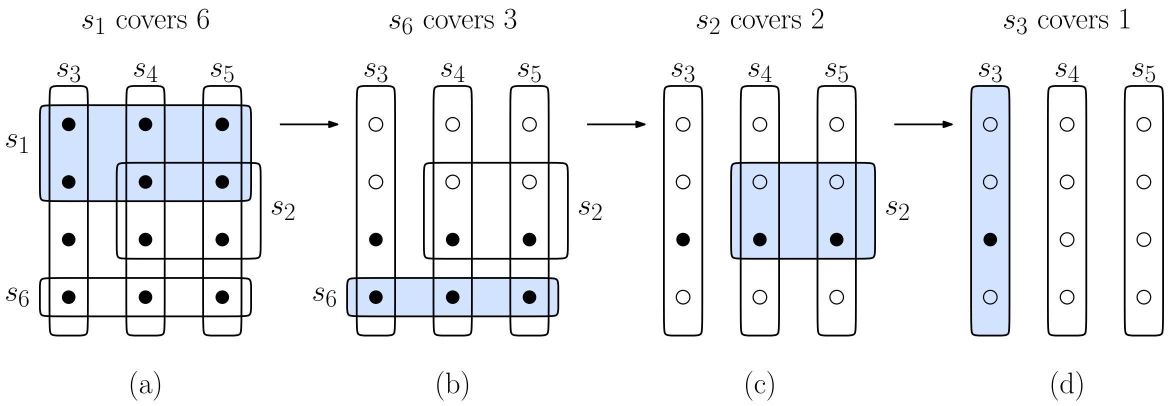

Fig. 2: Example of the greedy set-cover heuristic, returning $\{s_1, s_6, s_2, s_3\}$.

Running time: We will not worry about implementing this algorithm in the most efficient manner. Recall that $n = |X|$, and $m = |S|$ (the number of sets). It is possible implement this algorithm in time $O(nm)$. We will leave this as an exercise, but one way to visualize it is to imagine a bipartite graph, where the elements X are on one side and the sets S are on the other side. For each pair s and x where set s contains element x, we add the edge (s, x). From this perspective, the greedy algorithm repeatedly removes the highest-degree vertex from the S side, and delete all of its edges.

A bad case for greedy: The problem with the greedy heuristic is that it can be fooled into picking the wrong set, over and over again. Consider the example shown in Fig. 3 involving a universe of 32 elements. The optimal set cover consists of the two sets s_7 and s_8, each of size 16. Initially all three sets s_1, s_7, and s_8 have 16 elements. If ties are broken in the worst possible way, the greedy algorithm will first select the set s_1. We remove all the covered elements. Now s_2, s_7 and s_8 all cover eight of the remaining elements. Again, if we choose poorly, s_2 is chosen. The pattern repeats, choosing s_3 (covering four of the remainder), s_4 (covering two) and finally s_5 and s_6 (each covering one). Although there are ties for the greedy choice in this example, it is easy to modify the example so that the greedy choice is unique.

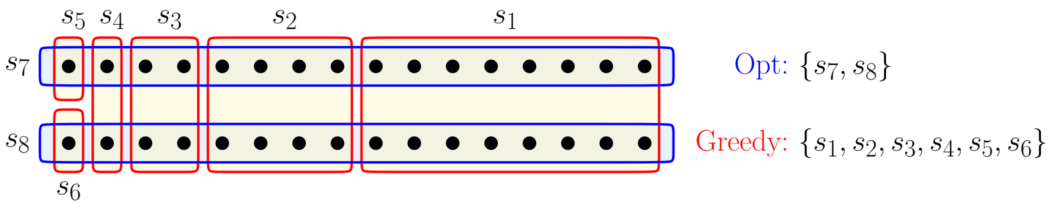

Fig. 3: Repeatedly fooling the greedy heuristic.

From the pattern, you can see that we can generalize this to any number of elements that is a power of 2. While there is a optimal solution with 2 sets, the greedy algorithm will select roughly $\lg n$ sets, where $n = |X|$. (Recall that "lg" denotes logarithm base 2.) Thus, on this example the greedy heuristic achieves an approximation factor of roughly $(\lg n)/2$. There were many cases where ties were broken badly here, but it is possible to redesign the example such that there are no ties, and yet the algorithm has essentially the same ratio bound.

Greedy's approximation factor: Next, we will show that this bad case is close to the worst case. In particular, we'll show that the number of sets generated by the greedy heuristic exceeds the optimum number by a factor of at most $\ln n$.

Theorem: Given any set system $\Sigma = (X, S)$, let G be the output of the greedy heuristic and let O be an optimum cover. Then $|G| \leq |O| \cdot \ln n$, where $n = |X|$.

Before giving the proof, we need one useful technical inequality.

Lemma: For all $c > 0$,

$$\left(1 - \frac{1}{c}\right)^c \leq \frac{1}{e}.$$

where e is the base of the natural logarithm.

Proof: We use the fact that for any real z (positive, zero, or negative), $1 + z \leq e^z$. (This follows from the Taylor's expansion $e^z = 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \ldots \geq 1 + z$.) Now, if we substitute $-1/c$ for z we have $(1 - 1/c) \leq e^{-1/c}$. By raising both sides to the cth power, we have the desired result.

We now prove the main theorem.

Proof: We will cheat a bit. Let o denote the size of the optimum set cover, and let g denote the size of the greedy set cover minus 1. We will show that $g \leq o \cdot \ln n$. (Note that we should really show that $g + 1 \leq o \cdot \ln n$, but this is close enough and saves us some messy details.)

Let's consider how many new elements we cover with each round of the algorithm. Let X_i denote the subset of X that remains to be cover after the ith iteration of the algorithm, and let $n_i = |X_i|$. Initially, $X_0 = X$ and $n_0 = n$. At the start of the ith iteration, there are n_{i-1} elements that remain to be covered.

We know that there is a cover of size o for the entire set X, and therefore there is a cover of size o for X_{i-1}. (You might think it should be smaller since we have covered many elements so far. But there is no guarantee that we have used any of the optimum sets in doing so.). Since $n_{i-1} = |X_{i-1}|$, by the pigeonhole principal there exists some set that covers at least n_{i-1}/o elements. Since the greedy algorithm selects the set covering the largest number of remaining elements, it must select a set that covers at least this many elements. The number of elements that remain to be covered is at most

$$n_i \;\leq\; n_{i-1} - \frac{n_{i-1}}{o} \;=\; n_{i-1}\left(1 - \frac{1}{o}\right).$$

Since this applies to every iteration, we see that with each iteration the number of remaining elements decreases by a factor of at least $(1 - 1/o)$. If we repeat this i times, we have

$$n_i \;\leq\; n_0\left(1 - \frac{1}{o}\right)^i \;=\; n\left(1 - \frac{1}{o}\right)^i.$$

How long can this go on? Since the greedy heuristic ran for $g+1$ iterations, we know that just prior to the last iteration we must have had at least one remaining uncovered element, and so we have

$$1 \;\leq\; n_g \;\leq\; n\left(1-\frac{1}{o}\right)^{g} \;=\; n\left(\left(1-\frac{1}{o}\right)^{o}\right)^{g/o}.$$

(In the last step, we just rewrote the expression in a manner that makes it easier to apply the above technical lemma.) By the above lemma, with $c = o$, we have

$$1 \leq n\left(\frac{1}{e}\right)^{g/o}.$$

Now, if we multiply by $e^{g/o}$ on both sides and take natural logs we find that g satisfies:

$$e^{g/o} \;\leq\; n \qquad \Rightarrow \qquad \frac{g}{o} \;\leq\; \ln n \qquad \Rightarrow \qquad g \;\leq\; o \cdot \ln n,$$

as desired. Therefore (ignoring the missing "+1" as mentioned above) the greedy set cover is larger than the optimum set cover by a factor of at most $\ln n$.

Summary: To summarize. In this short lecture we introduced the set-cover problem. (In a later lecture, we will show that set cover is NP-hard.) We introduced a simple greedy algorithm for set cover, which works by repeatedly taking the set that covers the greatest number of uncovered elements. Finally, we presented a proof that the greedy heuristic is within a factor of $\ln n$ of the size of the optimal cover, where n is the number of elements to be covered.

While greedy often works well in practice, there is no hope in trying to improve this result. It is known that (under fairly well-accepted assumptions about NP-hardness) it is not possible to approximate set cover to any factor smaller than $(1 - o(1)) \cdot \ln n$. (The factor $1 - o(1)$ means that the multiplicative factor is smaller than but arbitrarily close to 1.)