CMSC 451: Lecture 9
Greedy Approximation: Set Cover
Thursday, Sep 28, 2017

Reading: Chapt 11 of KT and Sect 5.4 of DPV.

Set Cover: An important class of optimization problems involves covering a certain domain, with sets of a certain characteristics. 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 $X = \{x_1, \ldots, x_m\}$ is a finite set of objects, called the *universe*, and $S = \{s_1, \ldots, s_n\}$ is a collection of subsets of X, such that every element of X belongs to at least one set of S. Set systems arise in many applications of science and engineering.

Undirected Graph: An undirected graph $G = (V, E)$ is a set system where V constitutes the universe, and the edges E are subsets of cardinality two.

Geometric set systems: The universe consists of n points in space, and the sets are the subsets of points that are contained within some specified geometric shape (balls, cubes, rectangles, triangles, etc.)

Wireless Coverage: The universe consists all the locations on a college campus, and for each possible location of a wireless transmitter there is an associated region of the campus that is covered by placing a wireless transmitter at this location.

A fundamental question involving set systems is determining the smallest number of sets needed to cover the entire universe. A *cover* of S is defined to be a subcollection of sets whose union covers X. For example, in Fig. 1(a), we elements of X are the black circles, and the sets $s_1, \ldots, s_6$ are indicated by rectangles. In this case there exists a cover of size three, consisting of s_3, s_4, and s_5 (see Fig. 1(b)). (The sets of this cover do not overlap, which is sometimes called an *exact cover*. In general, the sets of the cover are allowed to overlap.)

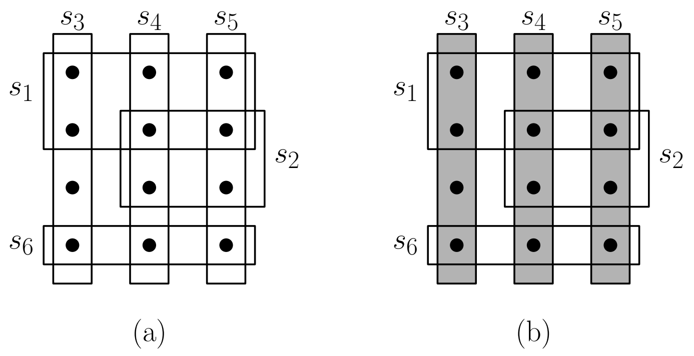

Fig. 1: Set cover. The optimum cover consists of the three sets $\{s_3, s_4, s_5\}$.

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 n, then we can think of a cover C more conveniently as a subset of $\{1, \ldots, n\}$. This suggests the following definition.

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

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

The set cover problem is a very important and powerful optimization problem. It arises in a vast number of applications. Determining the fewest locations to place wireless transmitters to cover the entire campus is an example. Unfortunately, the set cover problem is known to be NP-hard. We will present a simple *greedy heuristic* for this problem.

How do we determine how good our approximation is? Given an input instance $\Sigma = (X, S)$, let $O(\Sigma)$ denote an optimum cover (of minimum cardinality) and let $G(\Sigma)$ denote the cover produced by our greedy heuristic. Clearly, greedy cannot have fewer sets than the optimum, and so we have $|G(\Sigma)| \geq |O(\Sigma)|$. We say that G achieves an *approximation ratio* of ρ if $|G(\Sigma)| \leq \rho|O(\Sigma)|$, for any input Σ. Ideally, we would like ρ to be as small as possible, say, a small constant. Unfortunately, the best that we can show for set cover is that ρ is a slowly growing function of $m = |X|$, and in particular $\rho = \ln m$. (This might strike you as being rather weak, but there are compelling reasons from the theory of computational complexity that logarithmic approximation ratio is the best that we might hope for assuming that P $\neq$ NP. With a bit more work, it is possible to improve this slightly to an approximation ratio of $\rho = (\ln m')$, where m' is the maximum cardinality of any set of S.)

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                      // 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
}
```

We will not worry about implementing this algorithm in the most efficient manner. If we assume that U and the sets s_i are each represented as a simple list of elements of X (each of length at most m), then we can perform each iteration of the main while loop in time $O(mn)$, for a total running time of $O(mn^2)$.

For the example given earlier the greedy-set cover algorithm would select s_1 (see Fig. 2(a)), then s_6 (see Fig. 2(b)), then s_2 (see Fig. 2(c)) and finally s_3 (see Fig. 2(d)). Thus, it would return a set cover of size four, whereas the optimal set cover has size three.

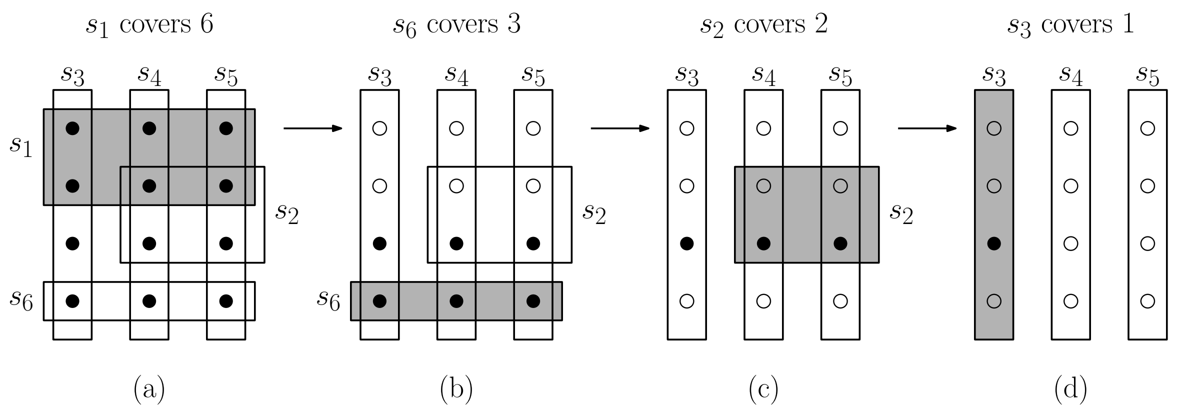

Fig. 2: The greedy heuristic. Final cover is $\{s_1, s_6, s_2, s_3\}$.

What is the approximation factor? 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 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.

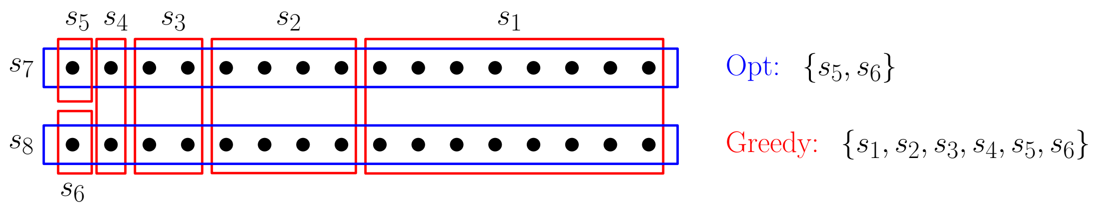

Fig. 3: Repeatedly fooling the greedy heuristic.

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 $\lg m$ sets, where $m = |X|$. (Recall that "lg" denotes logarithm base 2, and "ln" denotes the natural logarithm.) Thus, on this example the greedy heuristic achieves an approximation factor of roughly $(\lg m)/2$. 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.

We will show that the greedy set cover heuristic never performs worse than a factor of $\ln m$. 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 approximation bound.

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 $|G| \leq |O| \cdot \ln m$, where $m = |X|$.

Proof: We will cheat a bit. Let c 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 $g \leq c \cdot \ln m$. (Note that we should really show that $g + 1 \leq c \cdot \ln m$, 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. Initially, there are $m_0 = m$ elements to be covered. Let m_i denote the number of elements remaining to be covered after i iterations of the greedy algorithm. After $i - 1$ rounds there are m_{i-1} elements that remain to be covered. We know that there is a cover of size c for these elements (namely, the optimal cover), and so by the pigeonhole principal there exists some set that covers at least m_{i-1}/c 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

$$m_i \;\leq\; m_{i-1} - \frac{m_{i-1}}{c} \;=\; m_{i-1}\left(1 - \frac{1}{c}\right).$$

Thus, with each iteration the number of remaining elements decreases by a factor of at least $(1 - 1/c)$. If we repeat this i times, we have

$$m_i \;\leq\; m_0\left(1 - \frac{1}{c}\right)^i \;=\; m\left(1 - \frac{1}{c}\right)^i.$$

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

$$1 \;\leq\; m_g \;\leq\; m\left(1 - \frac{1}{c}\right)^g \;=\; m\left(\left(1 - \frac{1}{c}\right)^c\right)^{g/c}.$$

(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 we have

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

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

$$e^{g/c} \;\leq\; m \qquad \Rightarrow \qquad \frac{g}{c} \;\leq\; \ln m \qquad \Rightarrow \qquad g \;\leq\; c \cdot \ln m.$$

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 $\ln m$.