Lecture 8

In which we show how to round a linear programming relaxation in order to approximate the set cover problem, and we show how to reason about the dual of the relaxation to derive a simple combinatorial approximation algorithm for the weighted case.

Recall that in Set Cover we are given a finite set U and a collection $S_1, \ldots, S_n$ of subsets of U. We want to find the fewest sets whose union is U, that is, the smallest $I \subseteq \{1, \ldots, n\}$ such that $\bigcup_{i \in I} S_i = U$.

We described an algorithm whose approximation guarantee is $\ln |U| + O(1)$. The lower bound technique that we used in our analysis was to realize that if we have a subset $D \subseteq U$ of elements, such that every set S_i contains at most t elements of D, then $opt \geq |D|/t$.

Today we will describe a linear programming relaxation of set cover, and we will show a way to round it, in order to achieve an $O(\log |U|)$ approximation.

We will also show that the lower bounds to the optimum that we used in the analysis of the greedy algorithm can also be expressed as feasible solutions for the dual of our linear programming relaxation.

1 A Linear Programming Relaxation of Set Cover

We begin by formulating the set cover problem as an Integer Linear Programming problem. Given an input $(U, S_1, \ldots, S_n)$ of the set cover problem, we introduce a variable x_i for every set S_i, with the intended meaning that $x_i = 1$ when S_i is selected, and $x_i = 0$ otherwise. We can express the set cover problem as the following integer linear program:

$$\begin{array}{lll} \text{minimize} & \sum_{i=1}^{n} x_i & \\ \text{subject to} & & \\ & \sum_{i: v \in S_i} x_i \geq 1 & \forall v \in U \\ & x_i \leq 1 & \forall i \in \{1, \ldots, n\} \\ & x_i \in \mathbb{N} & \forall i \in \{1, \ldots, n\} \end{array} \tag{1}$$

From which we derive the linear programming relaxation

$$\begin{array}{lll} \text{minimize} & \sum_{i=1}^{n} x_i & \\ \text{subject to} & & \\ & \sum_{i:v\in S_i} x_i \geq 1 & \forall v \in U \\ & x_i \leq 1 & \forall i \in \{1,\ldots,n\} \\ & x_i \geq 0 & \forall i \in \{1,\ldots,n\} \end{array} \tag{2}$$

Remark 1 *In an earlier lecture, we noted that every instance G of the vertex cover problem can be translated into an instance $(U, S_1, \ldots, S_n)$ of the set cover problem in which every element of U belongs precisely to two sets. The reader should verify that the linear programming relaxation (3) of the resulting instance of set cover is identical to the linear programming relaxation of the vertex cover problem on the graph G.*

More generally, it is interesting to consider a *weighted* version of set cover, in which we are given the set U, the collection of sets $S_1, \ldots, S_n$, and also a *weight* w_i for every set. We want to find a sub-collection of *minimal total weight* whose union is U, that is, we want to find I such that $\bigcup_{i\in I} S_i = U$, and such that $\sum_{i\in I} w_i$ is minimized. The unweighted problem corresponds to the case in which all weights w_i equal 1.

The ILP and LP formulation of the unweighted problem can easily be generalized to the weighted case: just change the objective function from $\sum_i x_i$ to $\sum_i w_i x_i$.

$$\begin{array}{lll} \text{minimize} & \sum_{i=1}^{n} w_i x_i & \\ \text{subject to} & & \\ & \sum_{i:v\in S_i} x_i \geq 1 & \forall v \in U \\ & x_i \leq 1 & \forall i \in \{1,\ldots,n\} \\ & x_i \geq 0 & \forall i \in \{1,\ldots,n\} \end{array} \tag{3}$$

Suppose now that we solve the linear programming relaxation (3), and we find an optimal fractional solution $\mathbf{x}^*$ to the relaxation, that is, we are given a number x_i^* in the range $[0, 1]$ for every set S_i. Unlike the case of vertex cover, we cannot round the x_i^* to the nearest integer, because if an element u belongs to 100 sets, it could be that $x_i^* = 1/100$ for each of those sets, and we would be rounding all those numbers to zero, leaving the element u not covered. If we knew that every element u belongs to at most k sets, then we could round the numbers $\geq 1/k$ to 1, and the numbers $< 1/k$ to zero. This would give a feasible cover, and we could prove that we achieve a k-approximation. Unfortunately, k could be very large, even $n/2$, while we are trying to prove an approximation guarantee that is never worse than $O(\log n)$.

Maybe the next most natural approach after rounding the x_i^* to the nearest integer is to think of each x_i^* as a *probability*, and we can think of the solution $\mathbf{x}^*$ as describing a probability distribution over ways of choosing some of the subsets $S_1, \ldots, S_n$, in which we choose S_1 with probability x_1^*, S_2 with probability x_2^*, and so on.

Algorithm *RandomPick*

- Input: values $(x_1, \ldots, x_n)$ feasible for (3)
- $I := \emptyset$
- for $i = 1$ to n
  - with probability x_i, assign $I := I \cup \{i\}$, otherwise do nothing
- return I

Using this probabilistic process, the expected cost of the sets that we pick is $\sum_i w_i x_i^*$, which is the same as the cost of $\mathbf{x}^*$ in the linear programming problem, and is at most the optimum of the set cover problem. Unfortunately, the solution that we construct could have a high probability of missing some of the elements.

Indeed, consider the probabilistic process "from the perspective" of an element u. The element u belongs to some of the subsets, let's say for simplicity the first k sets $S_1, \ldots, S_k$. As long as we select at least one of $S_1, \ldots, S_k$, then we are going to cover u. We select S_i with probability x_i^* and we know that $x_1^* + \cdots x_k^* \geq 1$; what is the probability that u is covered? It is definitely not 1, as we can see by thinking of the case that u belongs to only two sets, and that each set has an associated x_i^* equal to $1/2$; in such a case u is covered with probability $3/4$. This is not too bad, but maybe there are $n/10$ elements like that, each having probability $1/4$ of remaining uncovered, so that we would expect $n/40$ uncovered elements on average. In some examples, $\Omega(n)$ elements would remain uncovered with probability $1 - 2^{-\Omega(n)}$. How do we deal with the uncovered elements?

First of all, let us see that every element has a reasonably good probability of being covered.

Lemma 2 *Consider a sequence of k independent experiments, in which the i-th experiment has probability p_i of being successful, and suppose that $p_1 + \cdots + p_k \geq 1$. Then there is a probability $\geq 1 - 1/e$ that at least one experiment is successful.*

PROOF: The probability that no experiment is successful is

$$(1 - p_1) \cdot \cdots \cdot (1 - p_k) \tag{4}$$

This is the kind of expression for which the following trick is useful: $1 - x$ is approximately e^{-x} for small values of x.

More precisely, using the Taylor expansion around 0 of e^x, we can see that, for $0 \leq x \leq 1$

$$e^{-x} = 1 - x + \frac{x^2}{2} - \frac{x^3}{3!} + \cdots$$

and so

$$1 - x \leq e^{-x} \tag{5}$$

because

$$e^{-x} - (1 - x) = \left(\frac{x^2}{2} - \frac{x^3}{3!}\right) + \left(\frac{x^4}{4!} - \frac{x^5}{5!}\right) \cdots \geq 0$$

where the last inequality follows from the fact that we have an infinite sum of non-negative terms.

Combining (4) and (5) we have

$$(1 - p_1) \cdot \cdots \cdot (1 - p_k) \leq e^{-p_1 - \ldots - p_k} \leq e^{-1}$$

□

Our randomized rounding process will be as follows: we repeat the procedure *RandomPick* until we have covered all the elements.

Algorithm *RandomizedRound*

1. Input: $x_1, \ldots, x_n$ feasible for (3)
2. $I := \emptyset$
3. while there are elements u such that $u \notin \bigcup_{i \in I} S_i$
   - for $i := 1$ to n
     - with probability x_i, assign $I := I \cup \{i\}$, otherwise do nothing
4. return I

How do we analyze such a procedure? We describe a very simple way to get a loose estimate on the quality of the cost of the solution found by the algorithm.

Fact 3 *There is a probability at most e^{-100} that the while loop is executed for more than $\ln |U| + 100$ times. In general, there is a probability at most e^{-k} that the while loop is executed form more than $\ln |U| + k$ times.*

PROOF: The probability that we need to run the *while* loop for more than $\ln|U| + k$ times is the same as the probability that, if we run the body of the *while* loop (that is, the *RandomPick* procedure) for exactly $\ln|U| + k$ steps, we end up with some uncovered elements.

Consider an element $v \in U$. For each iteration of the *while* loop, there is a probability at most $1/e$ that v is not covered by the sets added to I in that iteration. The probability that v is not covered after $\ln|U| + k$ iterations is then at most

$$\left(\frac{1}{e}\right)^{\ln|U|+k} = \frac{1}{|U|} \cdot e^{-k}$$

The probability that, after $\ln|U|+k$ iterations, there is an element that is not covered, is at most the sum over all v of the probability that v is not covered, which is at most e^{-k}. □

Fact 4 *Fix any positive integer parameter t and any feasible solution $(x_1, \ldots, x_n)$ for (3). Then the expected size of I in Algorithm RandomizedRound on input $(x_1, \ldots, x_n)$ after t iterations (or at the end of the algorithm, if it ends in fewer than t iterations) is at most*

$$t \cdot \sum_{i=1}^{n} w_i x_i$$

PROOF: Let $A_1, \ldots, A_t$ be the total cost of the elements assigned to I by the algorithm during the first, second, ..., t-th iteration, respectively. Let W denote the total weight $\sum_{i \in I} w_i$ after t iterations (or at the end of the algorithm if it terminates in fewer than t iterations). Note that these are random variables determined by the random choices made by the algorithm. Clearly, we have, with probability 1,

$$W \leq A_1 + \cdots + A_t$$

where we do not have equality because certain elements might be assigned to I in more than one iteration.

If the algorithm stops before the j-th iteration, then $A_j = 0$, because there is no j-th iteration and so no assignment happens during it. Conditioned on the j-th iteration occurring, the expectation of A_j is

$$\mathbb{E}[A_j | j\text{th iteration happens }] = \sum_{i=1}^{n} w_i x_i$$

and so we have

$$\mathbb{E}[A_j] \leq \sum_{i=1}^{n} w_i x_i$$

□

Recall that Markov's inequality says that if X is a non-negative random variable (for example, the size of a set), then, for every $c > \mathbb{E}\, X$

$$\mathbb{P}[X \geq c] \leq \frac{\mathbb{E}\, X}{c}$$

For example, $\mathbb{P}[X \geq 2\,\mathbb{E}\, X] \leq 1/2$.

We can combine these observations to get a rather loose bound on the size of the set output by RandomizedRound

Fact 5 *Given an optimal solution* $(x_1^*, \ldots, x_n^*)$ *to* (3)*, algorithm RandomizedRound outputs, with probability* $\geq .45$*, a feasible solution to the set cover problem that contains at most* $(2 \ln |U| + 6) \cdot opt$ *sets.*

Proof: Let $t := \ln |U| + 3$. There is a probability at most $e^{-3} < .05$ that the algorithm runs the *while* loop for more than t steps. After the first t steps, the expected total weight of the solution I is at most $t \sum_i w_i x_i^*$, which is at most $t \cdot opt$. Thus, the probability that the solution I, after the first t steps, contains sets of total weight more than $2t \cdot opt$ is at most $1/2$. Taking the union of the two events, there is a probability at most .55 that either the algorithm runs for more than t iterations, or that it adds to its solution sets of total cost more than $2t \cdot opt$ in the first t steps.

Equivalently, there is a probability at least .45 that the algorithm halts within t iterations and that, within that time, it constructs a solution of total cost at most $2t \cdot opt$. □

2 The Dual of the Relaxation

In a past lecture, we analyzed a simple greedy algorithm for unweighted set cover, that repeatedly picks a set that covers the largest number of uncovered elements, until all elements are covered, and we proved that the algorithm uses at most $(\ln |U| + O(1)) \cdot opt$ sets.

As in all analyses of approximation algorithms, we had to come up with ways to prove lower bounds to the optimum. In the unweighted case, we noted that if we have a subset $D \subseteq U$ of elements (D for "difficult" to cover), and every S_i contains

at most t elements of D, then $opt \geq |D|/t$. In the weighted case, neither this lower bound technique nor the approximation analysis works. It is possible to modify the algorithm so that, at every step, it picks the set that is most "cost-effective" at covering uncovered elements, and we can modify the analysis to take weights into accounts. This modification will be easier to figure out if we first think about the analysis of our previous algorithm in terms of the dual of the LP relaxation of set cover.

To form the dual of (3), we first note that the constraints $x_i \leq 1$ do not change the optimum, because a solution in which some x_is are bigger than 1 can be converted to a solution in which all variables are ≤ 1 while decreasing the objective function, and so no variable is larger than 1 in an optimal solution, even if we do not have the constraint $x_i \leq 1$.

If we work with this simplified version of the LP relaxation of set cover

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{n} w_i x_i \\ \text{subject to} & \\ & \sum_{i:v \in S_i} x_i \geq 1 \quad \forall v \in U \\ & x_i \geq 0 \qquad\quad \forall i \in \{1, \ldots, n\} \end{array} \tag{6}$$

we see that its dual is an LP with one variable y_v for every element $v \in U$, and it is defined as follows:

$$\begin{array}{ll} \text{maximize} & \sum_{v \in U} y_v \\ \text{subject to} & \\ & \sum_{v:v \in S_i} y_v \leq w_i \quad \forall i \in \{1, \ldots, n\} \\ & y_v \geq 0 \qquad\quad \forall v \in U \end{array} \tag{7}$$

That is, we assign a "charge" to every element, subject to the constraint that, for every set, the sum of the charges of the elements of the set are at most 1. The cost of the dual solution (and a lower bound to the optimum of the set cover problem) is the sum of the charges.

In the unweighted case, $w_i = 1$ for every i. Suppose that we are in the unweighted case, and that we notice a set $D \subseteq U$ of elements such that every S_i contains at most t elements of D. Then we can form the feasible dual solution

$$y_v := \begin{cases} \frac{1}{t} & \text{if } v \in D \\ 0 & \text{otherwise} \end{cases}$$

This is a feasible dual solution of cost $|D|/t$, and so it is a way to formulate our old lower bound argument in the language of dual solutions. The simplest extension of this example to the weighted setting is that: if we have a set D of elements such that for every set S_i of weight w_i we have that S_i contains at most $t \cdot w_i$ elements of D;

then the assignment $y_v := 1/t$ for $v \in D$ and $y_v := 0$ for $v \notin D$ is a feasible dual solution of cost $|D|/t$, and so the optimum is at most $|D|/t$.

These observations are already enough to derive a version of the greedy algorithm for weighted instances.

Algorithm: SetCoverGreedy

- Input: set U, subsets $S_1, \ldots, S_n$, weights $w_1, \ldots, w_n$
- $I := \emptyset$
- while there are uncovered elements $v \in U$ such that $v \notin S_i$ for every $i \in I$
  - let D be the set of uncovered elements
  - for every set S_i, let $e_i := |D \cap S_i|/w_i$ be the *cost effectiveness* of S_i
  - let S_{i^*} be a set with the best cost-effectiveness
  - $I := I \cup \{i^*\}$
- return I

We adapt the analysis of the unweighted case.

Let $v_1, \ldots, v_m$ be an enumeration of the elements of U in the order in which they are covered by the algorithm. Let c_j be the cost-effectiveness of the set that was picked at the iteration in which v_j was covered for the first time. Then, in that iteration, there is a set D of at least $m - j + 1$ elements, and every set S_i of weight w_i contains at most $w_i c_j$ elements of D. This means that setting $y_{v_1} = \cdots = y_{v_{j-1}} = 0$ and $y_{v_j} = \cdots = y_{v_m} = 1/c_j$ is a feasible dual solution of cost $(m - j + 1)/c_j$, and so

$$opt \geq (m - j + 1)\frac{1}{c_j}$$

If we denote by apx the cost of the solution found by our algorithm, we see that

$$apx = \sum_{j=1}^{m} \frac{1}{c_j}$$

because, at each iteration, if we pick a set S_{i^*} of weight w_{i^*} that covers t new elements, then each of those t elements has a parameter c_j equal to t/w_{i^*}, and so when we sum $1/c_j$ over all elements v_j that are covered for the first time at that iteration, we get exactly the weight w_{i^*}.

Rearranging the equations, we get again

$$apx \leq \sum_{j=1}^{m} \frac{1}{m-j+1} \cdot opt \leq (\ln m + O(1)) \cdot opt$$