Lecture 6: Set cover and relaxation

Rajat Mittal

IIT Kanpur

Previously, we designed an ILP and an LP for min-cut. It turned out that both had the same value. Specifically, it happened because the constraint matrix was totally unimodular. It is known that the vertices of the feasible region of $Ax \geq b$ is integral, if matrix A is unimodular. We will not cover the proof of this fact in this course.

Exercise 1. Read about totally unimodular matrices. The proof of above fact can be found at [2].

Clearly constraint matrix need not be totally unimodular. In general, when we move from ILP to LP for a problem, they will not have the same optimal value. Even in such cases, relaxing an ILP to LP might help. The LP might be a *close approximation* to the value of ILP. We will take the example of set cover in this section. By relaxing ILP of set cover to LP, we will be able to get an approximation algorithm for it.

1 Set cover problem

The main ideas of this section are taken from Luca Trevisan's lecture note [1]. Let us describe set cover problem and the concept of approximation algorithm first.

Set cover problem: Given a finite set U and its subsets $S_1, S_2, \cdots, S_n$, sets $S_{i_1}, S_{i_2}, \cdots, S_{i_k}$ cover U if and only if every element of U is contained in at least one of these sets. We will denote a set cover of U by a set $I \subseteq [n]$, indicating the indices of the subsets of U picked. For example, $I = \{i_1, i_2 \cdots, i_k\}$ would be the solution if sets $S_{i_1}, S_{i_2}, \cdots, S_{i_k}$ cover U. The set cover problem is to find a set cover I with minimum size.

To take an example, U could be the set of all students and S_i is set of students who play game i. We need to minimize number of sports to keep in the school such that every student has a game to play.

Notice that the number of sets is denoted by n, it can be different from the number of elements in U. Similarly I, our solution set, is a subset of $[n]$. Hence, $i \in [n]$ corresponds to the set S_i and not the i-th element of U.

We can consider the weighted version too, where each subset $S_i (1 \leq i \leq n)$ is assigned a weight w_i. In this case, we need to minimize the weighted sum of sets picked by I. To continue with the example above, w_i could denote the cost of having game i at the school.

Exercise 2. Can you give an ILP for set cover?

We will denote by x_i, if set S_i is part of the cover or not. The ILP is,

$$\begin{aligned} \min \quad & w_i x_i \\ \text{s.t.} \sum_{i: u \in S_i} x_i \geq 1 \quad & \forall u \in U \\ x_i \in \{0, 1\}. & \end{aligned}$$

Exercise 3. Verify that the ILP captures set cover problem (a solution of ILP is a solution of set cover and vice versa). What will be the ILP for non-weighted version?

Approximation algorithm: An *approximation algorithm* outputs a solution which is provably not far from the optimal solution. In our case, we are interested in an algorithm which outputs a set cover I, s.t.,

$$|I| \leq c. \text{ Optimal of set-cover problem.}$$

Here, $c \geq 1$ is a constant and the algorithm will be called a c-approximation algorithm for set cover.

Exercise 4. The sign of inequality and property of c will change if we had a maximization problem. Can you define the notion of approximation, like above, for a maximization problem?

Most of the times, our approximation algorithm will be a randomized algorithm. What this means is, at some stages of the algorithm, a step/decision will depend on a random number/coin toss. In these cases, we want to correctly answer (give a good approximation in case of approximation algorithms) with high probability over the randomness of the algorithm.

Notice that the probability is not over input but the randomness of the algorithm. In other words, for every input we will answer *correctly* with high probability over the randomness of the algorithm. In our case, we will show that the expected value of the outcome (for this randomized algorithm) approximates the optimal, where the expectation is taken over randomness of the algorithm. This statement can be converted into: outcome of the algorithm will approximate the optimal with high probability (using tools like Markov inequality and Chernoff bounds).

1.1 LP relaxation and algorithm

To give an approximation algorithm, the first step is to convert the ILP into an LP, known as *relaxation.* A relaxation of an optimization program is another optimization program which, ideally, is easier to solve and every solution of original program is also a solution of the new program with the same (or related) objective value.

A relaxation of the ILP for set cover can be obtained by replacing integer constraint with linear constraint $0 \leq x_i \leq 1$.

$$\begin{aligned} \min \quad & w_i x_i \\ \text{s.t.} \sum_{i: u \in S_i} x_i \geq 1 \quad & \forall u \in U \\ x_i \in [0,1]. \end{aligned}$$

Again, we know that $val(ILP) \leq val(LP)$, because the feasible region of LP is bigger. Can it be equal like minimum cut?

Let $U = \{1,2,3\}$ and $S_1 = \{1,2\}, S_2 = \{2,3\}, S_3 = \{3,1\}$. It is clear that we need at least two sets to cover U entirely.

Exercise 5. Find a feasible solution of the LP for this example, with a value better than 2.

This was expected, set cover is an NP-hard problem. We don't expect that an LP can characterize an NP-hard problem (why?).

So, the LP value is not equal to the ILP value. Can we at least get an approximate solution of set cover using this LP? Lovász gave a *randomized algorithm* for set-cover with approximation factor of $O(\log(|U|))$. We list the main idea (steps) of the algorithm.

1. Convert the integer linear program into a linear program.
2. Solve the linear program using any of the polynomial time solvers.
3. Then, convert the solution of LP into an integer solution ($\{0,1\}$) again.

We have already done the first step, second step is taken care of by interior point/ellipsoid method.

Exercise 6. Why did we not talk about simplex method?

The last ingredient is called *rounding*, converting the fractional solution of LP ($0 \leq x_i \leq 1$) into an integer solution ($x_i \in \{0,1\}$).

This rounding step might result in lose of objective value, we will analyze the rounding and prove that our resulting solution is still close to value of LP solution. In other words, we will prove,

$$val(Algo) \leq O(\log(|U|))val(LP) \text{ and } val(LP) \leq val(Algo).$$

Since algorithm returns a set cover, second inequality is obvious (why?). Notice that the LP solution has value higher than the ILP solution, our solution will be close to ILP optimal too.

$$val(Algo) \leq O(\log(|U|))val(ILP)$$

This gives a $\log(|U|)$-factor approximation algorithm for set-cover.

1.2 Rounding of LP solution

What is the objective of rounding? Let x_i's be the optimal solution of LP. That means, we will get a value x_i for every set S_i. Our task is to decide for all S_i;s, whether S_i is an element of the set cover using these values x_i.

If $x_i \in \{0, 1\}$, then this decision was easy — if $x_i = 1$ then pick the set S_i, otherwise ignore it. Unfortunately, LP can output any value between 0 and 1 for these x_i's. We only know that $\sum_{i:u \in S_i} x_i \geq 1$ for all $u \in U$.

Exercise 7. We also know that $\sum_i w_i x_i$ is not bigger than the weight of the best set cover. Why?

Our first attempt of rounding could be, if x_i is higher than $1/2$ then we pick the set S_i, otherwise we ignore it. Actually, the threshold $1/2$ is arbitrary, we can choose any threshold between 0 and 1.

Let us look at the following example which shows that the above rounding will not work. Let U be any set with n elements and all its subsets of size 3 are possible sets for set cover. One possible solution for this problem is $x_i = \frac{1}{\binom{n}{2}}$ for every subset S_i.

Exercise 8. Why did we choose this particular value of x_i?

As we increase n, the value of x_i goes below any threshold. For big enough n, our rounding will not pick any set.

The next attempt is to view x_i's as probability, i.e., x_i denotes the probability that set S_i should be in set cover. What rounding does this viewpoint give? Simply put, we pick each set S_i in our set cover with probability x_i independently (remember, we are targeting a randomized algorithm).

Note 1. This can be done by picking a random number between 0 and 1 and comparing it with x_i.

We need to check two things: our solution should not have high weight and it should be a set cover.

Exercise 9. What is the expected value of our solution?

Since weight w_i is picked with probability x_i, the expected value of our solution is $\sum_i w_i x_i$. This is the LP value of set cover and is the best possible value. Though, are we sure that every element of U is covered in the solution?

What is the probability that a particular element is covered/uncovered? Let us assume that the element, say u, is in sets $S_1, S_2, \cdots, S_k$. The probability that set S_1 is not picked is $1 - x_1$. Since, we are picking each set independently, the probability that u is not covered is,

$$(1 - x_1)(1 - x_2) \cdots (1 - x_k).$$

Obviously, this probability is not necessarily zero. That means, the solution given by our rounding might not be a set cover. We need to show that our solution is a set cover with high probability.

First, let us put a bound on the probability that an element of U is not covered. Using the approximation $(1 - x) \leq e^{-x}$ (Taylor series for e^{-x}), an element is uncovered with probability,

$$(1 - x_1)(1 - x_2) \cdots (1 - x_k) \leq e^{-(x_1 + x_2 + \cdots + x_k)} \leq e^{-1}.$$

Since there are $|U|$ elements, the only bound on the probability of our solution not being a set cover is $|U|\frac{1}{e}$. This probability is too large. One way to take care of this is, repeat the whole process again and include the new sets picked in our set cover. We don't need to repeat it just twice, we can repeat it till the probability of error becomes small.

The new rounding algorithm is,

1. Start with $I = \emptyset$.
2. Repeat the following step t times. Here t is a parameter which will be fixed later.
3. For all sets S_i, pick them with probability x_i independently. If a set is picked and is not already in I, include it in I.
4. Output I as the set cover.

What is the probability that the resulting I is still not a set cover? An element will not be covered with probability at most e^{-t} (why?). So, I will not be a set cover with probability at most $|U|e^{-t}$. This probability can be made a small constant with $t = \ln(|U|) + O(1)$ repetitions.

That is great, but what about the weight of I. Every round (repetition) can increase the weight by at most $\sum_i w_i x_i$. Using linearity of expectation, the expected value of the set cover is at most $O(\log(|U|))$ times the value of LP.

Summarizing, the algorithm outputs a set cover with high probability (any constant close to 1). The expected weight of the output is at most $O(\log(|U|))val(LP)$. In other words,

$$val(\mathbb{E}(Algo)) \leq O(\log(|U|))val(LP) \leq O(\log(|U|))val(\mathbb{E}(Algo)).$$

The second inequality is obvious because any set cover is a feasible solution of LP.

Also, since LP solution has value higher than ILP solution, our expected weight will be close to ILP optimal too.

$$val(\mathbb{E}(Algo)) \leq O(\log(|U|))val(ILP).$$

To convert this statement into an algorithm which works with high probability, we use Markov's inequality. It states that for any positive random variable X,

$$\Pr[X > c\mathbb{E}[X]] \leq 1/c.$$

Exercise 10. What will be X in our case?

If we pick random variable to be the output of the algorithm, Markov's inequality gives that

$$val(Algo) \leq O(\log(|U|))val(ILP),$$

with probability more than $1 - 1/c$. Notice that the constant(c) of Markov's inequality gets absorbed in $O(\log(|U|))$.

Exercise 11. Why is this enough? What value of c will you choose?

2 Assignment

Exercise 12. Read about randomized algorithms.

Exercise 13. Show that you can create arbitrary large gap between LP and ILP value of set cover.

References

1. L. Trevisan. Lecture 8: A linear programming relaxation of set cover. `https://theory.stanford.edu/~trevisan/cs261/lecture08.pdf`.
2. J. Vondrak. Polyhedral techniques in combinatorial optimization. `https://theory.stanford.edu/~jvondrak/MATH233B-2017/lec3.pdf`.