CS 3510 Algorithms Last updated: November 15, 2023

Lecture 24: Linear Programming and Max-Cut

Lecturer: Sahil Singla *Scribe(s): Joseph Gulian, Allen Chang*

Today we will begin discussing linear programming a popular technique for solving many real world problems. [1] It can also be used to approximate many NP-Complete problems. Our first motivating problem will be the Max-Cut problem, and our goal will be to map it to a set of linear equations called constraints, and then optimize some value while respecting those constraints.

[2]

1 Max-Cut

Recall the Min-Cut problem where in a flow network, the problem is to find the partition of nodes S and $V\ S$ which would minimize the size of the cut. Today we'll be talking about a similar problem Max-Cut. On this problem, given some undirected unweighted graph $G = (V, E)$. The problem is to output some partition $S \subseteq V$ such that the number of edges between S and $V\ S$ is maximized. It turns out that this problem is NP-Complete.

Even though this problem is hard, it is actually not complicated to come up with a 2-approximate algorithm. Let's say you have some cut in the graph [3] , you can attempt to marginally improve that by looking at each node in the graph, and considering if moving that node across the cut increases the size of the cut. If any node improves the size of the cut, move it over.

This idea is called local search: you have some solution, and you try to incrementally improve it until you can no longer. The benefit of this approach is that it is simple and can be applied to many optimization tasks like TSP or Max-Cut as we've seen.

To see an outline of this, let's consider our algorithm for Max-Cut.

1. Start with some intermediate solution $(S, V \setminus S)$.
2. Move vertices across the cut. If a vertex moving across the cut increases the size of the cut, move the vertex and return to step 1.

Now that we have an algorithm for this problem, let's analyze it. The first thing to notice about our local search solution is that it seems to be, potentially, an infinite loop. This is not the case however; notice that there are $\mathcal{O}(|E|)$ edges across the cut. At each step in our local search, to continue, we must have increased the size of the cut by at least one (the number of edges in a cut is some integer and we know it's greater than the current number). Then we know that this can't loop forever because the initial cut has size in the

[1] You can find many tools to solve these problems including OR-Tools.

[2] The first portion of this lecture is relevant to Exam 4, so it will be published independent of part 2. Later, we will re-add the second part.

[3] This isn't needed info, but if you picked nodes at random, you could actually get a 2-approximate solution in expectation.

range $[0, |E|]$, meaning there are a finite number of steps to $|E|$. If the search never reaches the optimal solution, we know that it must still stop; if it didn't stop, this would imply it would reach the optimal solution and stop there. Our discussion here also plays into the fact that we know the number of steps is linear with respect to the edges.

Then we just need to argue that the algorithm is 2-approximate. Let's consider just one node in the graph, the edges leaving that node can be one of two: internal edges (edges to vertices within the same partition), or cross edges (edges to vertices in the other partition). If we were to move the edge across the cut, we'd find that the cross edges and internal edges switch. Since we know we've reached some local optima when the algorithm terminates, we know that the number of cross edges is greater than the number of internal edges.

If you take the sum of all the cross edges divided by two (because two nodes share one cross edge), the result is the size of the cut. Similarly if you sum the internal edges divided by two, you get the total number of internal edges. Since an edge is either a cross edge or internal, the sum of these two is also the total number of edges. Because the number of cross edges is more than the number of internal edges, you know that it must be at least half the total number of edges. Refer to the equations below.

$$\begin{aligned}
\sum_{u \in V} \deg_{\text{internal}}(u) &\leq \sum_{u \in V} \deg_{\text{cross}}(u) \\
2 \ \# \text{ internal edges} &\leq 2 \ \# \text{ cross edges} \\
\# \text{ internal edges} &\leq \# \text{ cross edges} \\
\# \text{ internal edges} + \# \text{ cross edges} &\leq 2\# \text{ cross edges} \\
\# \text{ total edges} &\leq 2 \ \# \text{ cross edges} \\
\frac{1}{2} \ \# \text{ total edges} &\leq \# \text{ cross edges} \\
\frac{1}{2} \ \# \text{ optimal edges} &\leq \# \text{ cross edges}
\end{aligned}$$