Second Level Algorithms

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 08

Lecture 40

Welcome to the 40th lecture of the second-level algorithms course. In the last class, we have seen Karger's algorithm, which is a randomized algorithm for computing a minimum cut. In this lecture, we will see a generalization or even a better algorithm than Karger's algorithm, which is due to Karger and Stein, and has a better running time. So, let us see, but we begin this lecture by drawing some important corollaries from Karger's algorithm, okay? So, let us begin.

Recall that we have proved in the last lecture that the probability that Karger's algorithm outputs any particular minimum cut is at least $\frac{2}{n(n-1)}$. This is about the basic Karger's algorithm without the boosting procedure using repetition.

So, from this, we can draw an interesting corollary: the number of minimum cuts in any undirected graph is at most $\frac{n(n-1)}{2}$. Proof: Let G be any graph; its min cuts are say $X_1, X_2, \ldots, X_k$. We will show that k is at most $\frac{n(n-1)}{2}$. What is the probability that Karger's algorithm outputs X_i for any i in 1 to k? That is at least $\frac{2}{n(n-1)}$. So, what is the probability that there exists an i such that Kargar's algorithm outputs you see if the Kargars algorithm outputs X_i, it cannot output any other X_j for $j \neq i$ for this run. So, these events are mutually exclusive. So, this is sum of the probabilities.

Probability that Karger's algorithm outputs X_i and each of this is at least $\frac{2}{n(n-1)}$. at least this number, but we know probability values is always less than 1. So, this probability is less than equal to 1. So, what we have is 1 is greater than equal to $\frac{2k}{n(n-1)}$ which shows k is less than equal to $\frac{n(n-1)}{2}$ this is what we need to show.

So, the next question is that bound tight does there exist an example of a graph which has $\frac{n(n-1)}{2}$ min cuts. and the example is very simple. So, let us write it as an observation. The cycle graph on n vertices has $\frac{n(n-1)}{2}$ different min cuts.

The size of the min cut for a cycle graph is 2, and there and you pick any two edges that form a min cut. So, the bound is tight. Now, we improve the running time of Karger's min cut algorithm by using a beautiful trick and thereby get Karger-Stein algorithm. Recall the running time of Karger's algorithm was $O(n^4)$. We will see that the running time of Karger-Stein algorithm is substantially better than that of Karger's algorithm. So, let us see the Karger-Stein algorithm. The idea is to run Karger's algorithm not until we have 2 vertices, but until we have substantially more vertices. That means, let us run Karger's algorithm for t iterations instead of $n-2$ iterations. How to decide the value of t?

We will choose t such that the probability that Karger's algorithm picks any edge from a fixed min cut for contraction within the first t iterations is at most half. That means there is substantial probability—that is, with probability at least half—that Karger's algorithm does not pick any edge from a fixed min cut within the first t iterations. That is how we choose the value of t, okay? So, we run Karger's algorithm for t iterations such that for any min cut set of edges, if the probability that no edge from F is picked for contraction within the first t iterations is at least half. But what is the probability that no edge of F is picked within the first t iterations? Probability that no edge from F is picked for contraction within the first t iterations.

This probability we have calculated in the last lecture; this is at least no edge from F is picked for contraction in the first iteration. That probability is at least $\frac{(n-2)}{n}$. For the next iteration, $\frac{(n-3)}{(n-1)}$. For the t-th iteration, $\frac{(n-t-1)}{(n-t+1)}$. This product again telescopes; the last two numerators and the first two denominators remain. So, this is $(n-t)\times(n-t-1)$, and the denominator is $n\times(n-1)$. I want this number to be greater than or equal to half. We choose p so that the above probability is at least half. Directly working with this fraction is a bit cumbersome.

So, instead of working with this, we use a bound on this. So, this is greater than or equal to $\frac{(n-t-1)^2}{n^2}$. So, it is enough to choose t such that this holds. So, we choose t to be the floor of this number because t is the number of iterations, which must be an integer, ok. So, you see for almost or actually more than half the iterations, there is substantial probability that no edge from F is contracted by the algorithm. Only in the later half of the iterations, the probability of picking an edge from F shoots up. So, the idea is to repeat the later half of the algorithm twice and take the best one. So, in particular, here is the pseudocode of the Karger-Stein algorithm.

So, it is a recursive procedure. So, step 1: the base case—if G has only two vertices then output the cut induced by these two vertices. Otherwise, let us assume that G has more than 2 vertices. We run Karger's min-cut algorithm for t iterations for that particular value of t. Run Karger's algorithm for t iterations for the particular value of t that we have calculated.

H be the resulting graph. So, on edge we now run two instances of the algorithm in parallel and take the best one, which means recursively. So, u1 be the recursively called Karger-Stein on H. And call again because it is a randomized algorithm. The algorithm is randomized; the same function call with the same input may result in different outputs. Had it been a deterministic function, calling it twice with the same input would not make sense, but for randomized algorithms, it can make sense.

So, again recursively call on H. So, we have two cuts for G and return the best one of them. So, this is the Karger-Stein algorithm. So, two things we need to analyze. One is the error probability of the Karger-Stein algorithm, which will tell us how many times we have to repeat the algorithm to boost the probability of success, and the second is the running time of the algorithm.

So, let us first consider the time complexity of the Karger-Stein algorithm. One run of Karger later when we do the error analysis, it will tell us how many times we have to repeat it, and with that number, we have to multiply this time complexity of one run of the Karger-Stein algorithm. So, it is a recursive procedure. T(n) be the worst-case time complexity of the Karger-Stein algorithm on any graph with at most n vertices.

So, this is one base case if The base case is if the number of vertices is 2, and the recursive call will be made if n is greater than or equal to 3, ok. Now, let us see the pseudocode of the Kargerstein algorithm. The number of vertices in edge on which the recursive call is made. Let us see how much the number of vertices is at $n-1-\frac{n}{\sqrt{2}}-1$, which is $\frac{n}{\sqrt{2}}+2$. So, this is, and we are calling it twice. So, $2\times T\left(\frac{n}{\sqrt{2}}+2\right)$, and how much time we are spending? So, we are performing this contraction procedure for order n iterations, and each contraction procedure can be performed in O(n) time using some not-so-difficult data structure. So, assuming that our $n\log n$ time. So, assuming that we have $O(n^2\log n)$, actually order n time. So, let us not write log n also. edge contraction can be done in $O(n)$ time, ok.

Now, solving this using the master's theorem, we get T(n) is $O(n^2\log n)$ for one run. So, now, here we cannot see the improvement because if you consider one run of Karger's algorithm, it involves $n-2$ contractions, and if each contraction can be done in $O(n)$ time, then one run of Karger's algorithm also takes $O(n^2)$. But the improvement will come from error analysis.

We will see that the probability of success of Karger's algorithm is much better than the probability of success of Karger's algorithm. We will see that the probability of success of one run of the Karger-Stein algorithm is at least some constant times $\frac{1}{\log n}$. Contrast this with the probability of success of one run of Karger's algorithm, which is $\frac{1}{n^2}$, OK.

So, we need to repeat $O(\log n)$ times to boost the success probability algorithm. Hence, the overall runtime of the Karger-Stein algorithm is $O(n^2 \log^2 n)$. Contrast this with the $O(n^4)$ runtime of Karger's algorithm.

The Karger-Stein algorithm's runtime is even better than the runtime of maximum flow-based algorithms. Of all the maximum flow-based algorithms, actually running one instance of flow itself takes $O(n^3)$ time. This is even better than that. So, in the next class, we will do the error analysis of the Karger-Stein algorithm, okay? So, let us stop here. Thank you.