CME 305: Discrete Mathematics and Algorithms
Instructor: Professor Aaron Sidford (sidford@stanford.edu)
January 9, 2018

Lecture 1 - Edge Connectivity and Global Minimum Cut[1]

1 Edge Connectivity

In this lecture we be begin our tour of discrete mathematics and algorithms by looking at one of the most fundamental and classic problems in combinatorial optimization and graph theory, computing the *edge connectivity* of an undirected, unweighted graph. In the next few lectures we will take a more principled and fundamental approach to graph theory and connectivity. The goal of today is to get a taste of what is to come in the rest of unit 1 and more broadly, this course.

In the rest of the lecture we assume that we have an undirected, unweighted graph $G = (V, E)$ with *vertices* or *nodes* V and *edges* E. Graphs are a natural way to model pairwise relationships. They are one of the simplest types of *structured data* and are one of the first tools often used to model relationships between entities. As we will see over the course they occur everywhere and algorithms for optimizing over them have broad implications.

In the next lecture we will look at slightly broader classes of graphs (directed and weighted graphs), but throughout this lecture our graph, $G = (V, E)$ will be undirected and unweighted. Here, V simply denotes an arbitrary finite set called the *nodes* or *vertices* and $E = \{e_1, ...e_m\}$ simply denotes a multi-set of un-ordered pairs $\{u, v\}$ where $u, v \in V$. Often we let $n \stackrel{\text{def}}{=} |V|$ and $m \stackrel{\text{def}}{=} |E|$ and that will be the case for the remainder of this lecture. If we had wanted to restrict to *simple-graphs* we would have disallowed both *self-loops,* i.e. the case of $\{u, v\} \in E$ for $u = v$, and *multi-edges*, i.e. $e_i = e_j$ for $i \neq j$. However, here we allow each for reasons we will see later.

The applications and modeling power of such graphs is vast. However here are a few natural instances to keep in mind that motivate the problem we are considering. Suppose we have a social network and wish to model who is friends with who. Here the vertices of the graph would be people and we would say that $\{i, j\} \in E$ if and only if person i and person j are friends. We could imagine we have a communication network, where the nodes are computers, routers, or cell-towers and there is an edge between them if they can communicate. We could also imagine that we have a transportation network, where the edges correspond to segments of road and the junctions between the roads are the vertices.

Once we have a abstracted these networks as graphs, there are a number of properties we could try to ask about about graphs that might be meaningful for the particular network they represent. The particular question we will be concerned about this lecture is one of network robustness or how well connected a graph is.

While there are many ways of quantifying connectivity of an undirected graph (and we may see a few in this course), the definition of whether or not a graph is connected is standard. We define connectivity in a graph in terms of the existence of paths, i.e. ways of following edges to get from one vertex to another.

Definition 1 (Path)**.** In an undirected graph $G = (V, E)$ a set of edges $\{u_1, v_1\}, ..., \{u_k, v_k\}$ is called a u_1, v_k-*path* or a *path* from u_1 to v_k if and only if $\{u_i, v_i\} \in E$ for all $i \in [k]$ and $v_i = u_{i+1}$ for all $i \in [k-1]$.[2]

Note that in an undirected graph, if there is a $u - v$-path there is always trivially a $v - u$-path obtained simply by reversing the edges of the path. With this in mind we define connectivity as follows.

[1]These lecture notes are a work in progress and there may be typos, awkward language, omitted proof details, etc. Moreover, content from lectures might be missing. These notes are intended to converge to a polished superset of the material covered in class so if you would like anything clarified, please do not hesitate to post to Piazza and ask.

[2]We use $[i] \stackrel{\text{def}}{=} \{1, ...i\}$ with the convention that $[0] \stackrel{\text{def}}{=} \emptyset$ throughout the notes. See notation file.

Definition 2 ((Undirected) Graph Connectivity)**.** In an undirected graph $G = (V, E)$ we say that vertex v is *connected* to u or *reachable* from u if and only if there is a $u - v$-path in G. We say that $G = (V, E)$ is *connected* if and only if every pair of vertices $u, v \in E$ are connected.

Intuitively, this definition says that an undirected graph is connected if and only if there is a way to follow edges to get from any vertex to any other vertex. For example, in the context of a social network this says that the graph associated with a social network is connected if and only if every person is is a friend of a friend of a friend of a friend, etc. of everybody else.

Now intuitively not all connected graphs feel as connected as each other. Consider the following two canonical graphs.

Definition 3 (Line Graph)**.** We call a graph $L_n = (V, E_{line})$ with $V = [n]$ the *n-node line graph if* $E_{line} = \{\{i, i+1\} \mid i \in [n-1]\}$.

Definition 4 (Complete Graph)**.** We call a graph $K_n = (V, E_{clique})$ with $V = [n]$ the *n-node complete graph or clique if* $E_{clique} = \{\{i, j\} \mid i, j \in [n], i \neq j\}$.

There are many ways to measure connectivity of a graph, but the one we will be concerned about this lecture is known as the *edge connectivity* of G defined as follows.

Definition 5 ((Undirected) Graph Edge-Connectivity)**.** The *edge connectivity* of an undirected graph $G = (V, E)$, denoted $\lambda(G)$, is the fewest number of edges that need to be removed from G to make it disconnected.

The central question we address for the rest of this lecture is how do we compute the edge connectivity of a given G.

2 Global Minimum Cut

Now, before we attempt to tackle this question, let's take a step back and try to get a better understanding of its structure. We will do this repeatedly in the course. Before directly attacking a problem like this we will attempt to see what sort of alternative characterizations of it and the optimal solution we can find. This approach often both sheds light both on the structure of the objects we wish to study (e.g. road network and transportation networks) as well as provides us with tools to design efficient algorithms.

So, before we we try to build algorithms for this problem, let's ask are the question, *are there any equivalent characterizations of the problem that someone can think of?* Even more fundamentally, before we ask this question of edge connectivity we should ask it of connectivity itself. How would you convince me that a graph is disconnected? Intuitively a graph is disconnected if there are two vertices that cannot reach each other. If we look at the set of vertices that each vertex can reach and add the remaining vertices to one of the two sets, the there are no edges crossing. This suggest the following definitions.

Definition 6 (Graph Cut)**.** Given an undirected graph $G = (V, E)$ a *cut* in G is a pair (S, T) that non-trivially partition V, i.e. $S \subseteq V$ and $T = V \setminus S$ with $S, T \notin \{\emptyset, V\}$. For any sets $S, T \subseteq V$ we let $E(S, T) \stackrel{\text{def}}{=} \{\{u, v\} \in E \mid u \in S, v \in T\}$ and for cut (S, T) we call $E(S, T)$ the *cut set* or the *cut edges* and call $|E(S, T)|$ the size of the cut. For brevity we often just denote a cut by a non-trivial set $S \subseteq V$, i.e. $S \notin \{\emptyset, V\}$, and say it *induces cut* $(S, V \setminus S)$ *of size* $C(S) \stackrel{\text{def}}{=} |E(S, V \setminus S)|$.

We will explore optimizing over cuts a lot in the remainder of the course. As we have suggested they characterize connectivity as follows.

Lemma 7 (Cut Characterization of Connectivity). *A graph $G = (V, E)$ is connected if and only if the cut-set of every cut is non-empty.*

Proof. Suppose G is connected and let S be a non-trivial subset of V. Since is non-trivial there exist $u \in S$ and $v \notin S$. By the definition of connectivity there is a $u - v$ path, i.e. $\{u_1, v_1\}, \{u_2, v_2\}, ..., \{u_k, v_k\} \in E$ with $u_1 = u$, $v_k = v$, and $v_i = u_{i+1}$ for all $i \in [k]$. Since $u \in S$ and $v \notin S$ there must be some $\{u_i, v_i\} \in E$ with $u_i \in S$ and $v_i \notin S$. Consequently, the cut induced by S is non-empty. Since S was arbitrary we have that every cut is non empty.

On the other hand, suppose that G is not connected. Then there exists some $u, v \in V$ such that there is no path from u to v. Let S be the set of vertices reachable from u. Clearly, S is non-trivial since $u \in S$ and $v \notin S$. However, $E(S, V \setminus S) = \emptyset$ since if there was $i \in S$ and $j \in V \setminus S$ then j is reachable from u by going to i and then j. □

From this lemma we see that removing a set of edges makes the graph disconnected if and only if make a cut size zero. This yields the following.

Corollary 8 (Cut Characterization of Edge Connectivity). *For undirected graph $G = (V, E)$ we have $\lambda(G) = \min_{S \subseteq V, S \notin \{\emptyset, V\}} C(S)$.*

This latter problem, computing $\min_{S \subseteq V, S \notin \{\emptyset, V\}} C(S)$ is known as the *(global) minimum cut problem.*

Definition 9. Given an undirected graph $G(V, E)$, a *global min-cut* is a partition of V into two subsets (A, B) such that the number of edges between A and B is minimized. The *global minimum-cut problem* is the problem of computing a *global min-cut.*

The word *glo*bal is to distinguish it from what is known as the perhaps more well known s, t*-minimum cut problem* of

$$\min_{S \subseteq V, s \in S, t \in V \setminus S} C(S) .$$

We will discuss this problem more in Lecture #4.

3 Minimum Cut Algorithm Attempts

So how should we try to compute the global minimum cut? Here we briefly go over some natural attempts with undesirable properties before looking at the main algorithm we will consider.

3.1 Brute Force

Computing the minimum cut in a graph is a a natural discrete optimization or combinatorial optimization problem. There are only a finite number of cuts, $2^n - 2$ to be precise (or $2^{n-1} - 1$ if we only care about the cut edges) and thus we are just optimizing over a finite set. So one natural algorithm is simply to try all cuts and output the one of minimum size. This works fine, it just takes a really long time, time exponential in the size of the input to be more precise. This is a standard feature of combinatorial optimization problems, there is often a natural exponential time algorithm, and the question we seek to address is does the problem admit sufficient structure to let us do better. If we were optimizing over an arbitrary functions of cuts, we could not, but the hope here is that we could do better.

3.2 s-t Minimum Cut

As we have already, discussed instead of trying to compute global minimum cut, we could try to compute the s-t minimum for all pairs s and t, that is the minimum cut $(S, V \setminus S)$ such that $s \in S$ and $t \in V \setminus S$. We know that the cut needs to separate at least two vertices and thus we could simply try all $\binom{n}{2}$ pairs. Consequently with roughly n^2 calls to an algorithm that solves s-t minimum cut we can solve the problem.

Note, that we could do a bit better by observing that any vertex s must be in one side of the minimum cut and thus we could try all $n-1$ vertices that could be on the other side. Consequently with roughly $n-1$ calls to an algorithm that solves s-t minimum cut we can solve the problem.

In Lecture #4 we will show how to solve the s-t minimum cut efficiently. For a long time, this was the fastest way known to compute global minimum cut and thus it was thought that the global minimum cut problem is computationally more challenging the the $s-t$ minimum cut problem. However, by the end of the course we will show how to compute the global minimum cut in a time we do not know how to achieve for the $s-t$ minimum cut.

3.3 Greedy

Another natural idea is to try to simply pick edges one at a time that improve some sort of potential. We could try just decreasing degrees (the number of edges incident to a vertex) or something like that. However, there is a graph known as the barbell graph (two complete graphs connected by an edge) that easily shows why many natural such variants of this don't work. Moreover, in Lecture #3 we will have a more complete discussion of the greedy algorithm and when exactly it works.

It is worth noting that there is actually a way to make an algorithm like this work though. If we start with a set S consisting of a single vertex and repeatedly add the vertex $i \in V$ to S such that $E(S, i)$ is maximized then it turns out that if one considers the vertices encountered in order, i.e. $v_1, ..., v_n$ then either $(\{v_1, ..., v_{n-1}\}, v_n)$ is a global minimum cut or v_{n-1} and v_n are on the same side of the global minimum cut. This gives an efficient algorithm (and is an instance of an algorithm for a broader problem known as symmetric submodular function minimization), but this is outside the scope of this lecture.

4 Karger's Algorithm

So the idea we'll use instead is to try to recurse on graphs of smaller size. This is a common iterative algorithmic technique that we will see several times in the course. However, the particular instantiation of it for the minimum cut problem isn't like many of the algorithms will see in the class and generalizing it to other problems has been somewhat difficult. If you want a nice problem to think about while you read through these lecture notes, try to think about what algorithmic paradigm is actually at work here and how it could be used for things like $s-t$ minimum cut or solving problems like symmetric submodular minimization.

With this idea in mind, our goal will be to reduce the problem of computing the min-cut on an n-node graph to computing the min-cut on a $n-1$-node graph. The way we will do this is through *vertex contraction.*

Definition 10 (Vertex Contraction)**.** Given a graph $G = (V, E)$ and vertices $u, v \in V$ let graph resulting from contracting u, v be the graph G where vertices u and v are replaced with a new vertex and ever edge where one of its endpoints was u or v is set to to the new vertex.

There are a few things quickly to note about this operation. First, it clearly decreases the number of vertices by 1. Second, it might create some self-loops (in particular if $\{u, v\} \in E$). Lastly, note the there is a natural bijection between cut-sets induced by cuts that do not separate u and v and cut-sets in the contraction graph. An immediate consequence of this is that cuts in the contracted graph have the same size as a corresponding cut in the original graph where each contracted vertex is set to the set of vertices that were contracted to it.

Note that if we contract two vertices on the same side of the minimum-cut, then solving the problem on the contracted graph corresponds to solving the problem on the original graph! This seems useful, however if all we did was contract random pairs of vertices until the graph was of size 2 then this is no different then picking a random cut and the probability this succeeds is simply the number of minimum-cuts over the total number of cuts which is exponential. As we will soon see, relatively there aren't too many minimum cuts and thus this algorithm fails.

However, looking at the barbell graph and thinking about the problem a little bit we see that we are trying to find the minimum cut, so most of the edges in the graph should not cross any particular minimum cut. Consequently, if we contract the endpoints of a random edge, rather than just contracting a random pair, maybe this algorithm is more likely to succeed? The resulting algorithm is known as Karger's min-cut algorithm and is given below.

Algorithm 1 Karger's randomized global min-cut

```
repeat
    Remove all self-loops
    Choose an edge $\{u, v\}$ uniformly at random from $E$.
    Contract vertices $u$ and $v$
until $G$ has only 2 vertices.
Report the corresponding cut in the original graph
```

4.1 Algorithm Analysis

Let's analyze Karger's algorithm. The main question we wish to ask is, what is the probability that this algorithm succeeds? We analyze this in pieces. First, what is the probability that a single iteration picks an edge crossing any particular minimum cut?

Lemma 11. *Let $G = (V, E)$ be a connected undirected unweighted graph with no-self loops and let $S \subseteq V$ induce a minimum cut, i.e. $C(S) = \lambda(G)$. Then the probability a random edge in the graph cross the cut induced by S is at most* $2/|V|$.

Proof. Let $m = |E|$. The probability that a random edge is in $E(S, V \setminus S)$ is simply $C(S)/m$, the number of edges in $E(S, V \setminus S)$ divided by m. So to upper bound the probability the edge cross the cut, we simply need to upper bound $C(S) = \lambda(G)$. Consequently, we simply need to show that a sufficiently small cut exists. To do that, let's just look at a random degree cut, i.e. $C(\{v\})$ for a random $v \in V$. Note that each edge $\{u, v\}$ contributes 1 to $C(u)$ and 1 to $C(v)$ and consequently $\sum_{v \in V} C(\{v\}) = 2m$ and there the expected size of a random degree cut is

$$\frac{1}{|V|} \sum_{v \in V} C(\{v\}) = \frac{2m}{n} .$$

Consequently, there is at least one degree cut whose size is at most $\frac{2m}{n}$. Consequently, $\lambda(G) \leq 2m/n$ and the result follows.

The trick we used in this lemma about reasoning about a random process and using the expected value to prove existence of a certain value is known as the *probabilistic method* and we will discuss it in greater detail later in the course. □

Applying this lemma repeatedly we can obtain our bound.

Theorem 12. *Let $G = (V, E)$ be a connected undirected unweighted graph with no-self loops and let $S \subseteq V$ induce a minimum cut, i.e. $C(S) = \lambda(G)$, the probability that the algorithm outputs S is at least $\frac{2}{n(n-1)}$.*

Proof. Let E_i by the event that the the i^{th} edge is not in $E(S, V \setminus S)$. by the previous lemma we have that $\Pr[E_1] \geq 1 - \frac{2}{n}$. Furthermore this implies that

$$\begin{aligned} Pr(E_2|E_1) &\geq 1 - \frac{2}{n-1} \\ Pr(E_i|E_1 \cap E_2 \cap \ldots) &\geq 1 - \frac{2}{n-i+1} \end{aligned}$$

Combining these to get the total probability

$$\begin{aligned} \Pr[\text{output } S] &= \Pr[E_1 \cap E_2 \cap \ldots \cap E_{n-2}] \\ &\geq \Pr[E_1]\Pr[E_2|E_1]\ldots\Pr[E_{n-2}|E_1 \cap \ldots E_{n-3}] \\ &\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\cdots\left(1 - \frac{2}{3}\right) \\ &\geq \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\cdots\left(\frac{1}{3}\right) \\ &\geq 2\frac{(n-2)!}{n!} = \frac{2}{n(n-1)}. \end{aligned}$$

□

This lower bound on the probability of finding a min-cut seems rather low, and goes to zero as $n \to \infty$. However, there is a standard trick we can apply to boost the success probability of a randomized algorithms. If we run the same algorithm t times then the probability that each execution fails is

$$\left(1 - \frac{2}{n^2}\right)^t \leq \exp\left(-\frac{2t}{n^2}\right)$$

where we used that $1 + x \leq \exp(x)$ for all x. Furthermore, if we simply compare all the cut sizes computed and output the smallest where $t = cn^2$ this yields that the probability we do not output the smallest cut is at most $e^{-2c} \leq e^{-c}$. Consequently, it decays exponentially with c which seems quite good. If we want to succeed with probability $1 - p$ we simply need to run the algorithm $n^2 \log(1/p)$ times.

Another common way the success probability is dealt with is to set set $c = \alpha \log n$. In this case the success probability is $1 - \frac{1}{n^\alpha}$. This is known as succeeding, *with high probability* n, i.e. the failure probability is inverse polynomial in n where we can control the degree of the polynomial simply by increasing the number of executions by a constant. This is a nice notion, as it means that if the procedure is called repeatedly inside any algorithm that calls it a polynomial number of times (a common measure of efficiency), then by union bound the overall algorithm succeeds with high probability. In other words, this notion composes well.

4.2 Structural Implications

While we motivated our algorithm through some structural understanding of the minimum cut problem, our algorithm analysis in turn yields new structural insights about the minimum cut problem. A common theme in this course is using algorithms and their analysis to reason about the structure of discrete objects.

As an example of this principle, let us ask the question, in a connected n-node graph how many global min-cuts can their be?

Corollary 13. *Every connected unweighted undirected graph has at most $\binom{n}{2}$ global minimum cuts.*

Proof. We showed that any particular minimum cut is output with probability $1/\binom{n}{2}$. So if k is the number of minimum cuts, the expected number of minimum cuts output is $k/\binom{n}{2}$. But clearly the expected number of cuts output is at most 1 and the result follows. $\square$

By considering the cycle on n-vertices it is easy to see that this bound is tight, i.e. there exist n-node connected graphs with $\binom{n}{2}$ distinct global minimum cuts. A similar argument can be used to bound the number of approximate minimum-cuts in the graph. We call a cut $S \subseteq V$ an α-min-cut if $C(S) \leq \alpha \cdot \lambda(G)$. It can be shown that

Corollary 14. *Every connected unweighted undirected graph has at most $n^{2\alpha}$ α-min-cuts.*

Also note that when we run Karger's algorithm to succeed with high probability, with high probability it computes every single minimum cut. Since union bounding over the n^2 cuts gets absorbed into the constants for a high probability bound.