Stanford University — CS261: Optimization | Handout 13
Luca Trevisan | February 17, 2011

Lecture 13

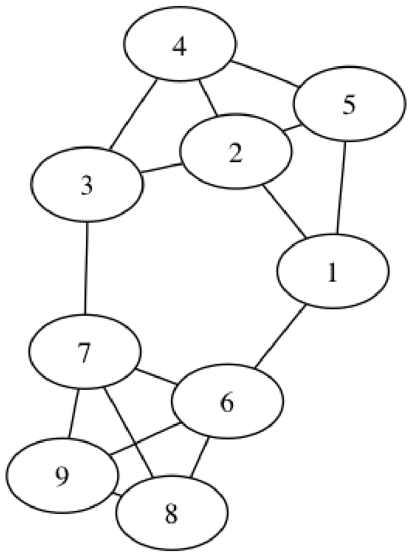
In which we describe a randomized algorithm for finding the minimum cut in an undirected graph.

1 Global Min-Cut and Edge-Connectivity

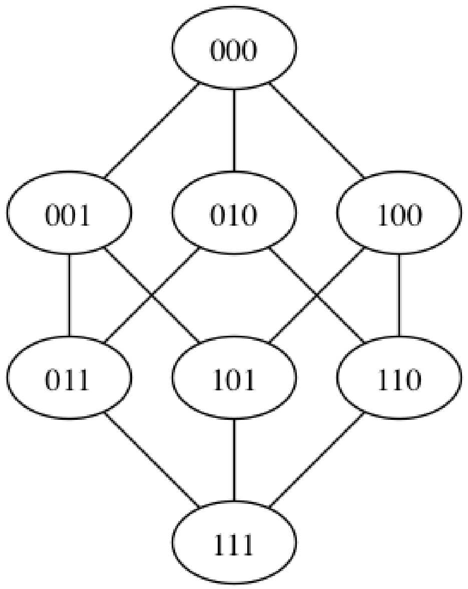
Definition 1 (Edge connectivity) *We say that an undirected graph is k-edge-connected if one needs to remove at least k edges in order to disconnect the graph. Equivalently, an undirected graph is k-edge-connected if the removal of any subset of $k-1$ edges leaves the graph connected.*

Note that the definition is given in such a way that if a graph is, for example 3-edge-connected, then it is also 2-edge-connected and 1-edge connected. Being 1-edge-connected is the same as being connected.

For example, the graph below is connected and 2-edge connected, but it is not 3-edge connected, because removing the two edges $(3, 7)$ and $(1, 6)$ disconnects the graph.



As another example, consider the 3-cube:



The 3-cube is clearly not 4-edge-connected, because we can disconnect any vertex by removing the 3 edges incident on it. It is clearly connected, and it is easy to see that it is 2-edge-connected; for example we can see that it has a Hamiltonian cycle (a simple cycle that goes through all vertices), and so the removal of any edge still leaves a path that goes trough every vertex. Indeed the 3-cube is 3-connected, but at this point it is not clear how to argue it without going through some complicated case analysis.

The *edge-connectivity* of a graph is the largest k for which the graph is k-edge-connected, that is, the minimum k such that it is possible to disconnect the graph by removing k edges.

In graphs that represent communication or transportation networks, the edge-connectivity is an important measure of reliability.

Definition 2 (Global Min-Cut) *The global min-cut problem is the following: given in input an undirected graph $G = (V, E)$, we want to find the subset $A \subseteq V$ such that $A \neq \emptyset$, $A \neq V$, and the number of edges with one endpoint in A and one endpoint in $V - A$ is minimized.*

We will refer to a subset $A \subseteq V$ such that $A \neq \emptyset$ and $A \neq V$ as a *cut* in the graph, and we will call the number of edges with one endpoint in A and one endpoint in $V - A$ the *cost* of the cut. We refer to the edges with one endpoint in A and one endpoint in $V - A$ as the edges that *cross* the cut.

We can see that the Global Min Cut problem and the edge-connectivity problems are in fact the same problem:

- if there is a cut A of cost k, then the graph becomes disconnected (in particular, no vertex in A is connected to any vertex in $V - A$) if we remove the k edges that cross the cut, and so the edge-connectivity is at most k. This means that the edge-connectivity of a graph is at most the cost of its minimum cut;
- if there is a set of k edges whose removal disconnects the graph, then let A be the set of vertices in one of the resulting connected components. Then A is a cut, and its cost is at most k. This means that the cost of the minimum cut is at most the edge-connectivity.

We will discuss two algorithms for finding the edge-connectivity of a graph. One is a simple reduction to the maximum flow problem, and runs in time $O(|E| \cdot |V|^2)$. The other is a surprising simple randomized algorithm based on *edge-contractions* – the surprising part is the fact that it correctly solves the problem, because it seems to hardly be doing any work. We will discuss a simple $O(|V|^3)$ implementation of the edge-contraction algorithm, which is already better than the reduction to maximum flow. A more refined analysis and implementation gives a running time $O(|V|^2 \cdot (\log |V|)^{O(1)})$.

1.1 Reduction to Maximum Flow

Consider the following algorithm:

```
• Input: undirected graph $G = (V, E)$
• let $s$ be a vertex in $V$ (the choice does not matter)
• define $c(u, v) = 1$ for every $(u, v) \in E$
• for each $t \in V - \{s\}$
    – solve the min cut problem in the network $(G, s, t, c)$, and let
      $A_t$ be the cut of minimum capacity
• output the cut $A_t$ of minimum cost
```

The algorithm uses $|V| - 1$ minimum cut computations in networks, each of which can be solved by a maximum flow computation. Since each network can have a maximum flow of cost at most $|V| - 1$, and all capacities are integers, the Ford-Fulkerson algorithm finds each maximum flow in time $O(|E| \cdot opt) = O(|E| \cdot |V|)$ and so the overall running time is $O(|E| \cdot |V|^2)$.

To see that the algorithm finds the global min cut, let k be edge-connectivity of the graph, E^* be a set of k edges whose removal disconnects the graph, and let A^* be the connected component containing s in the disconnected graph resulting from the removal of the edges in E^*. So A^* is a global minimum cut of cost at most k (indeed, exactly k), and it contains s.

In at least one iteration, the algorithm constructs a network (G, s, t, c) in which $t \notin A^*$, which means that A^* is a valid cut, of capacity k, for the network, and so when the algorithm finds a minimum capacity cut in the network it must find a cut of capacity at most k (indeed, exactly k). This means that, for at least one t, the cut A_t is also an optimal global min-cut.

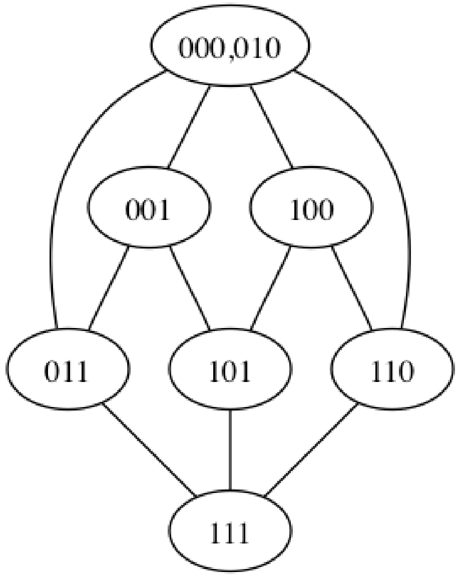
1.2 The Edge-Contraction Algorithm

Our next algorithm is due to David Karger, and it involves a rather surprising application of random choices.

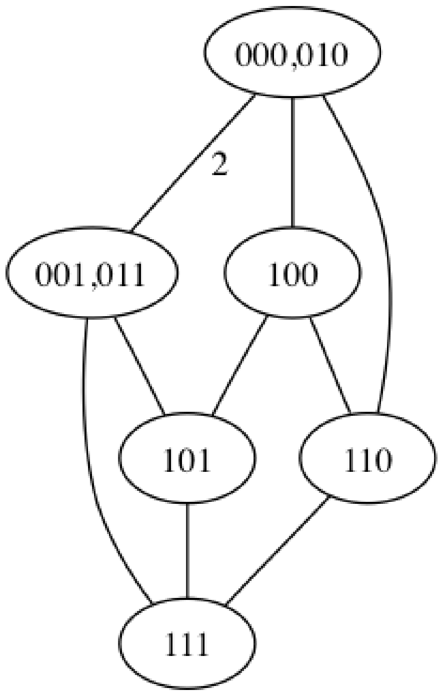
The algorithm uses the operation of *edge-contraction*, which is an operation defined over multi-graphs, that is graphs that can have multiple edges between a given pair of vertices or, equivalently, graphs whose edges have a positive integer weight.

If, in an undirected graph $G = (V, E)$ we *contract* an edge (u, v), the effect is that the edge (u, v) is deleted, and the vertices u and v are removed, and replaced by a new vertex, which we may call $[u, v]$; all other edges of the graph remain, and all the edges that were incident on u or v become incident on the new vertex $[u, v]$. If u had i edges connecting it to w, and v had j edges connecting it to w, then in the new graph there will be $i + j$ edges between w and $[u, v]$.

For example, if we contract the edge $(000, 010)$ in the 3-cube we have the following graph.



And if, in the resulting graph, we contract the edge $(001, 011)$, we have the following graph.



Note that, after the two contractions, we now have two edges between the "macro-vertices" $[000, 010]$ and $[001, 011]$.

The basic iteration of Karger's algorithm is the following:

- while there are ≥ 3 vertices in the graph
  - pick a random edge and contract it
- output the set A of vertices of the original graph that have been contracted into one of the two final macro-vertices.

One important point is that, in the randomized step, we sample uniformly at random among the edges of the multi-set of edges of the current *multi*-graph. So if there are 6 edges between the vertices (a, b) and 2 edges between the vertices (c, d), then a contraction of (a, b) is three times more likely than a contraction of (c, d).

The algorithm seems to pretty much pick a subset of the vertices at random. How can we hope to find an optimal cut with such a simple approach?

(In the analysis we will assume that the graph is connected. if the graph has two connected components, then the algorithm converges to the optimal min-cut of cost zero. If there are three or more connected components, the algorithm will discover them when it runs out of edges to sample, In the simplified pseudocode above we omitted the code to handle this exception.)

The first observation is that, if we fix for reference an optimal global min cut A^* of cost k, and if it so happens that there is never a step in which we contract one of the k edges that connect A^* with the rest of the graph then, at the last step, the two macro-vertices will indeed be A^* and $V - A^*$ and the algorithm will have correctly discovered an optimal solution.

But how likely is it that the k edges of the optimal solution are never chosen to be contracted at any iteration?

The key observation in the analysis is that if we are given in input a (multi-)graph whose edge-connectivity is k, then it must be the case that every vertex has degree $\geq k$, where the degree of a vertex in a graph or multigraph is the number of edges that have that vertex as an endpoint. This is because if we had a vertex of degree $\leq k - 1$ then we could disconnect the graph by removing all the edges incident on that vertex, and this would contradict the k-edge-connectivity of the graph.

But if every vertex has degree $\geq k$, then

$$|E| = \frac{1}{2} \sum_v \text{degree}(v) \geq \frac{k}{2} \cdot |V|$$

and, since each edge has probability $1/|E|$ of being sampled, the probability that, at the first step, we sample one of the k edges that cross the cut A^* is only

$$\frac{k}{|E|} \leq \frac{2}{|V|}$$

What about the second step, and the third step, and so on?

Suppose that we were lucky at the first step and that we did not select any of the k edges that cross A^*. Then, after the contraction of the first step we are left with a graph that has $|V|-1$ vertices. The next observation is that this new graph *has still edge-connectivity* k because the cut defined by A^* is still well defined. If the edge-connectivity is still k, we can repeat the previous reasoning, and conclude that the probability that we select one of the k edges that cross A^* is at most

$$\frac{2}{|V|-1}$$

And now we see how to reason in general. If we did not select any of the k edges that cross A^* at any of the first step $t-1$ step, then the probability that we select one of those edges at step t is at most

$$\frac{2}{|V|-t+1}$$

So what is the probability that we never select any of those edges at any step, those ending up with the optimal solution A^*? If we write E_t to denote the event that "at step t, the algorithm samples an edge which does not cross A^*," then

$$\mathbb{P}[E_1 \wedge E_2 \wedge \cdots \wedge E_{n-2}]$$

$$\begin{aligned}&= \mathbb{P}[E_1] \cdot \mathbb{P}[E_2|E_1] \cdot \ldots \mathbb{P}[E_{n-2}|E_1 \wedge \cdots \wedge E_{n-3}] \\ &\geq \left(1-\frac{2}{|V|}\right) \cdot \left(1-\frac{2}{|V|-1}\right) \cdot \ldots \left(1-\frac{2}{3}\right)\end{aligned}$$

If we write $n := |V|$, the product in the last line is

$$\frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdot \frac{n-5}{n-3} \cdot \ldots \frac{4}{6} \cdot \frac{3}{5} \cdot \frac{2}{4} \cdot \frac{1}{3}$$

which simplifies to

$$\frac{2}{n \cdot (n-1)}$$

Now, suppose that we repeat the basic algorithm r times. Then the probability that it does not find a solution in any of the r attempts is at most

$$\left(1-\frac{2}{n \cdot (n-1)}\right)^r$$

So, for example, if we repeat the basic iteration $50n\cdot(n-1)$ times, then the probability that we do not find an optimal solution is at most

$$\left(1 - \frac{2}{n\cdot(n-1)}\right)^{50n\cdot(n-1)} \leq e^{-100}$$

(where we used the fact that $1 - x \leq e^{-x}$), which is an extremely small probability.

One iteration of Karger's algorithm can be implemented in time $O(|V|)$, so overall we have an algorithm of running time $O(|V|^3)$ which has probability at least $1 - e^{-100}$ of finding an optimal solution.