CSCI 5454 Ramdomized Min Cut

Sean Wiese, Ramya Nair

April 8, 2013

1 Randomized Minimum Cut

A classic problem in computer science is finding the minimum cut of an undirected graph. If we are presented with a graph, such as a computer network or transportation network, it can be useful to be able to identify a minimum cut of the graph were by removing a minimum number of edges which splits the graph into two separate connected components.

This problem has already been discussed in lectures 9 and 10 on maximum flows and minimum cuts. Using either Ford-Fulkerson or Edmonds-Karp which run in $O(E|f*|)$ and $O(V^2E)$ time. In this lecture we will show how we can use a random heuristic to determine the minimum cut in less than $O(V^2E)$ time.

2 Karger Algorithm

In the early 1990s a PHD student at Stanford devised an algorithm, the Karger Algorithm, for finding the minimum cut using random sampling. In this section we will discuss the various parts of this algorithm as well as provide a proof for its correctness and an analysis of the running time.

2.1 Contracting the Graph

The basic idea of this algorithm is that we reduce a graph down to 2 vertices with a single cut remaining between the two. To do this we use the contraction function described in section 2.1.2. However, before we get to that section will talk about how we can collapse an edge in a graph which is the basis for the contraction function.

2.1.1 Collapsing an Edge

The basis of Kargers algorithm is how we are able to contract a graph by collapsing edges within that graph. Collapsing an edge refers to combining two vertices to form a single vertex, modifying the graph such that all the edges from these two vertices now emerge from the new single vertex. If there are more than one edges between the selected two vertices, they are deleted. This also has the possibility of creating self loops, which we remove, and parallel edges which will remain.

The pseudocode to collapse an edge is given in the following algorithm for when our graph is represented by an adjacency matrix, the algorithm for collapsing an edge in an adjacency list is given in the section 5.2. The algorithm takes as input graph, G and two vertices, u and v, that need to be collapsed. This algorithm consists of four basic steps.

1. Add row v to row u. This step takes all edges whose source is v and makes their source u.

2. Add column v to column u. This step takes all edges whose target is v and makes their target u.

3. Zero column v and row v. This step deletes the vertex v.

4. Set the value of $A_{u,u}$ to zero. This step removes self loops on u.

Each one of these steps has a running time of $O(n)$, therefore, the total running time for collapsing an edge is the summation of the running time of each step which is $O(n)$.

2.1.2 Contracting a Graph

To contract a graph we can use algorithm 1 to repeatedly collapse vertices until we have a graph with only 2 vertices remaining.

Algorithm 1 CONTRACT-GRAPH(G)

```
1: while $len(G) > 2$ do
2:     pick a random edge (u,v) in G
3:     COLLAPSE-EDGE($G, u, v$)
4: end while
5: return G
```

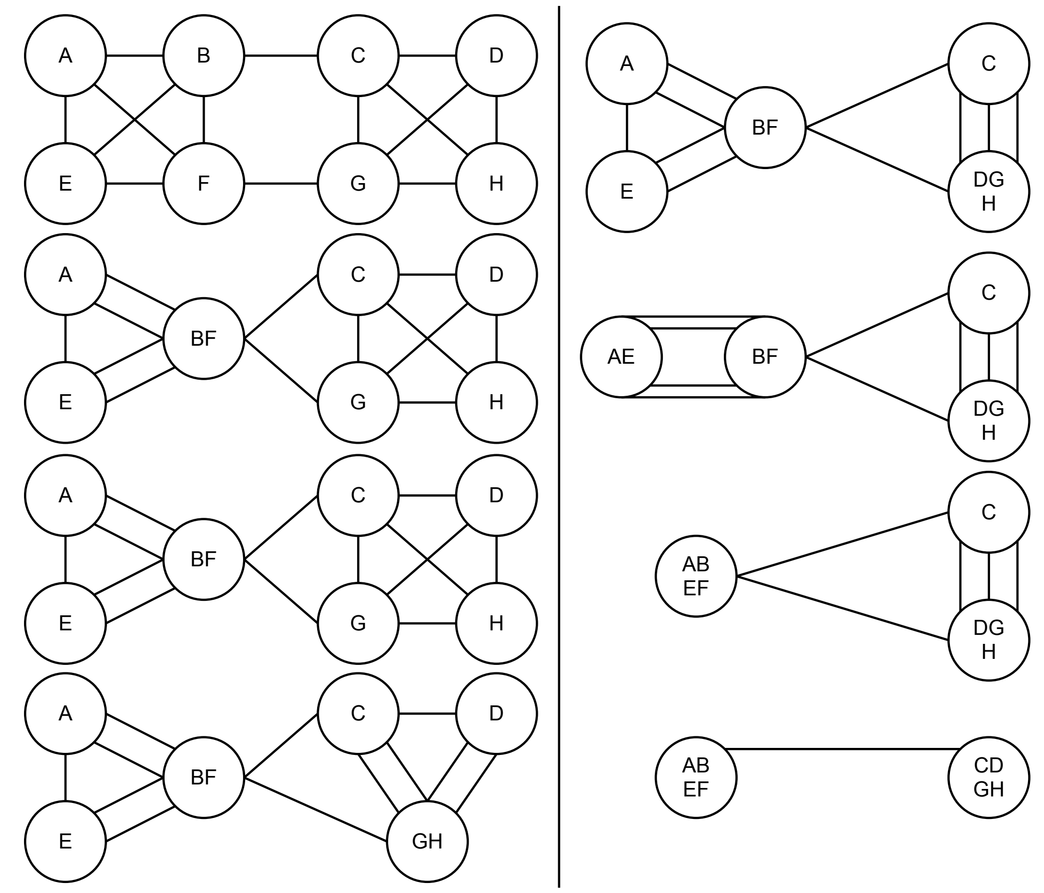
The above process essentially isolates the graph into two connected components where the remaining edges are those that connect the two components. We can show that this algorithm produces a valid cut by using the following proof.

Proof: Edges are deleted from the graph only when they connect two contracting vertices, and these vertices necessarily end up in the same partition. Therefore no edges between the two partitions are removed and they will exist in the final stage.

Running Time: We can also look at the running time of this function. The algorithm performs (n-2) edge contractions with each contraction taking $O(n)$ time making the running time $O(n^2)$

2.1.3 Example

Figure 1 Example of collapsing a graph



2.1.4 When do we find a min cut?

So far we have shown that we can isolate a cut using the contraction algorithm. However, we do not know when the algorithm would return a min cut. In this section we will provide a proof for when the algorithm produces a min cut.

Statement:

Let k be the actual min-cut size, and let $C = e1, e2, ..., e_k$ be a k-cut. Unless some $e \epsilon C$ is selected for contraction, C is output. We can find the min cut in the graph G, if and only if an edge in the set C is never collapsed.

Proof:

Let S and T be the partitions of G, and C be all edges (st) such that $s \epsilon S, t \epsilon T$.

If we don't contract an element in C , then the compound nodes stay entirely within S or T . At the end of the algorithm, we have two compound nodes covering the entire graph, each contained entirely in S or T . One must be S and the other T.

Only if we dont contract an element in C, then we have found the min cut. Suppose that we collapse an edge in C while running the algorithm. This will merge some node in S with some node in T. Therefore, we cannot output the components S and T when the algorithm finishes.

2.2 Probability of Finding Min-cut

After contracting a graph we are left with a single cut remaining. We also know, from the previous section, that if we never collapsed an edge that was included in every min-cut in G then the cut that we are left with is a min cut. To understand the likelihood that this cut that we found is the min cut we first need to look at the probability of choosing an edge in the min cut.

Suppose that we have a graph G with a min cut of size k. Since every vertex must have at least k edges we know that the graph must contain kn possible edge vertex pairs. Since every edge connects two vertices we know that our graph must contain at least $\frac{kn}{2}$ edges. From this we can now determine that the probability of selecting an edge in the min cut is at most $\frac{k}{\frac{kn}{2}}$ which reduces to $\frac{2}{n}$.

After collapsing the first edge the probability that the next edge is included in the min cut is $\frac{2}{n-1}$ and this pattern continues recursively until we are left with only two vertices. Therefore, the probability of finding the min cut is given by the following recurrence.

$$P(n) \geq \frac{n-2}{n} * P(n-1)$$

We can expand this recurrence into the following product and simplify.

$$P(n) \geq \prod_{i=3}^{n} \frac{i-2}{i} = \frac{\prod_{i=3}^{n}(i-2)}{\prod_{i=3}^{n} i} = \frac{\prod_{i=1}^{n-2}(i)}{\prod_{i=3}^{n} i} = \frac{(n-2)!}{\frac{n!}{2!}} = \frac{2}{n(n-1)}$$

This is pretty awesome in that we have probability of finding the min cut that scales polynomially while the number of cuts in the graph scales exponentially.

2.3 Kargers Algorithm

Algorithm 2 KARGER(G)

```
1: $minCut = \infty$
2: for all $i = 1$ to $T$ do
3:     X = CONTRACT-GRAPH(G)
4:     if $|X| < minCut$ then
5:         $minCut = |X|$
6:         $minX = |X|$
7:     end if
8: end for
9: return minX
```

The chance of finding the min cut when we are contracting the graph is not very high. To improve this probability we can perform algorithm 1 multiple times and return the lowest value found. This technique is referred to as amplification. This is Kargers algorithm which can be found in 2.

Since this equation is random we cannot guarantee that we will find the min cut. However, with a larger number of iterations, T, we can have a high probability of finding the minimum cut.

To begin our analysis we will first look at the probability that Kargers algorithm fails to return the actual min cut. This occurs when we run the algorithm T times and fail each time to find the true value. Our probability to find a min cut is independent and can be written with the following equation

$$P_S(n) \geq 1 - (1 - \frac{2}{n(n-1)})^T$$

We can simplify the above equation using the convenient inequality

$$1 - x \leq e^{-x}$$

Using the above inequality we simplify our probability to be greater than or equal to,

$$1 - e^{\frac{-2T}{n(n-1)}}$$

As T gets large we can get this probability to approach 1, but never equal to 1. If we set $T = n(n-1)/2$, we get the probability of success to be:

$$1 - \frac{1}{e}$$

If we desire to get a probability that scales polynomially with the value of T we can set T to

$$T = c\binom{n}{2}ln(n)$$

which when plugged into our previous equation we get a probability of

$$1 - e^{\frac{-2(c\binom{n}{2}ln(n))}{n(n-1)}} = 1 - e^{\frac{-2(c*\frac{n(n-1)}{2}*ln(n))}{n(n-1)}} =$$

$$1 - e^{-c*ln(n)} = 1 - \frac{1}{n^c}$$

Since the chance of failure is a polynomial fraction we can conclude that this algorithm finds the min cut with a high probability for the case where $T = c\binom{n}{2}ln(n)$.

The running time for the case where $T = c\binom{n}{2}ln(n)$ can be found by first looking at the cost of collapsing a graph and the number of times we run the collapsing algorithm. The running time of Kargers algorithm is the following when $T = c\binom{n}{2}ln(n)$

$$O(T * n^2) = O(n^2 * ln(n) * n^2) = O(n^4 * ln(n))$$

If we wish that the algorithm succeeds with very high probability, we can choose $N = \frac{n^2(n-1)}{2}$ which leads to a success probability of

$$1 - e^{-n}$$

However this comes at the cost of running time since our running time is now $O(n^5)$

The space used by Kargers algorithm depends on the data structure used for the graph. If we use an adjacency list the space used will be $O(V + E)$ while if we use the adjacency matrix the space used will be $O(V^2)$.

3 Karger and Stein

In order to explain this algorithm we would first discuss an interesting property of probability. For a graph with a min cut of k and total vertices n, as we have discussed earlier, the probability that the first edge we pick is in the minimum cut is no greater than $2/n$. In the beginning of graph contraction the value $2/n$ is very high but as the graph collapses (considering we have not collapsed any edges in the min cut yet), the probability of collapsing an edge in the min cut increases. For example, for a graph with 100 vertices, initially the probability is no greater than $2/100$ but consider when the graph has collapsed to 3 vertices, now the chance that we pick an edge in the mincut is less than $2/3$. Thus, every time we pick an edge that is not in the min cut we are increasing the probability to pick the next one in the min cut.

Karger and Stein algorithm utilizes this property to reduce the running time. As we continue to pick an edge (hence, increase the probability to pick an edge in the min cut) we also increase the number of random choices that we examine to pick the one with smallest cut. Thus, at the beginning we run fewer number of trials, since the probability to select an edge in min cut is low, the number of trials increases as the graph becomes smaller.

If we consider less than fifty percent chance of not choosing an edge in the min cut as good, every time we reach that limit we could increase the amplification. This is called Branching amplification. Taking this strategy would reduce the edge to $n/\sqrt{2}$ before the first amplification. The probability that we choose an edge in min cut could be derived from the expression in previous section as :

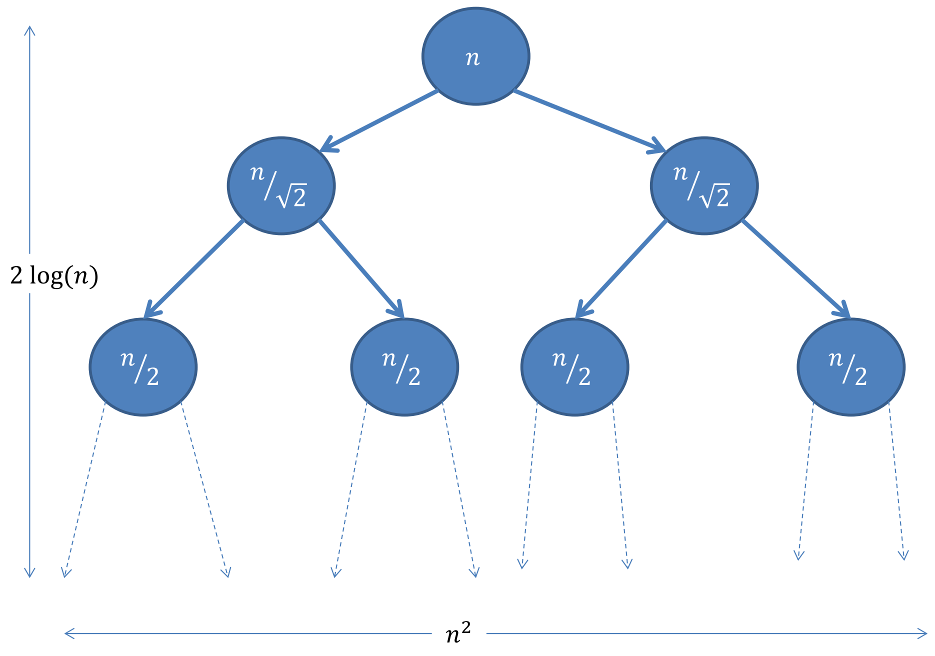
$$\prod_{i=n/\sqrt{2}}^{n} \frac{i-2}{i} = \frac{n/\sqrt{2}(n/\sqrt{2}-1)}{n(n-1)} \approx \frac{(n/\sqrt{2})^2}{n^2} \approx 1/2$$

This implies that the probability that we collapse an edge in min cut when we contract the graph to $n/\sqrt{2}$ nodes is greater than or equal to $1/2$. This probability decreases to less than half, when we collapse more than $n/\sqrt{2}$ vertices.

Now, the next edge that we select has higher probability of being in the min cut. Thus, in order to mitigate this risk , we make two edge selection and take the minimum cut amongst those two. We continue this recursively, that is , we contract this new graph by the same ratio, and then again increase the repetitions.

This could be represented in the binary tree format as shown in figure below

Figure 2 Karger and Stein Algorithm - Tree representation



Algorithm for Karger-Stein randomized min cut algorithm is given below: The CONTRACT function below takes as input parameter a Graph and the number of vertices (s) to which the graph needs to be collapsed. For each iteration, the function selects a random edge and collapses it. This is continued till the graph shrinks to s vertices. The BETTER-MIN-CUT function, recursively calls itself to contract the graph using the strategy explained above. Every call to BETTER-MIN-CUT function represents a split in the binary tree after which the graph is reduced to $n/\sqrt{2}$ vertices and then split again. Note that contraction of edges is randomized, which means every split has its own path down the binary tree.

Algorithm 3 BETTER-MIN-CUT(G)

```
1: $X_1$ = BETTER-MIN-CUT(CONTRACT($G, n/\sqrt{(2)}$))
2: $X_2$= BETTER-MIN-CUT(CONTRACT($G, n/\sqrt{(2)}$))
3: **return** min($X_1, X_2$)
```

Algorithm 4 CONTRACT(G, m)

```
1: for all $i = n$ to $m$ do
2:     pick a random edge (u,v) in G
3:     COLLAPSE-EDGE$(G, u, v)$
4: end for
5: return G
```

3.1 Probability

The BETTER-MIN-CUT function would return the minimum cut if it does not select min cut in either of the first two lines. Which means,

$$P(n) = 1 - (1 - P_{X1})(1 - P_{X2})$$

where P_{X1} is the probability that X1 selects the min cut and P_{X2} is the probability that X2 selects the min cut. Furthermore,

$$P_{X1} = P_{X2} < \frac{1}{2} \times P\Big(\frac{n}{\sqrt(2)}\Big)$$

Therefore the overall probability of success can be expressed as :

$$P(n) \geq 1 - \Big(1 - \frac{1}{2} \times P\Big(\frac{n}{\sqrt{2}}\Big)\Big)^2$$

$$P(n) \geq P\Big(\frac{n}{\sqrt{2}}\Big) - \frac{1}{4}P\Big(\frac{n}{\sqrt{2}}\Big)^2$$

We would prove by induction that probability $P(n) = 1/lgn$ by induction. The base case would be $P(2) = 1 = 1/lg(2)$.

$$\begin{aligned} P(n) &\geq P\Big(\frac{n}{\sqrt{2}}\Big) - \frac{1}{4}P\Big(\frac{n}{\sqrt{2}}\Big)^2 \\ &= \frac{1}{lg\Big(\frac{n}{\sqrt{2}}\Big)} - \frac{1}{4lg^2\Big(\frac{n}{\sqrt{2}}\Big)} \\ &= \frac{1}{lg(n) - \frac{1}{2}} - \frac{1}{4(lg(n) - \frac{1}{2})^2} \\ &= \frac{4lg(n) - 3}{4lg^2(n) - 4lg(n) + 1} \end{aligned}$$

$$= \frac{1}{lg(n)} + \frac{1 - \frac{1}{lg(n)}}{4lg^2(n) - 4lg(n) + 1}$$
$$> 1/lg(n)$$

We assume n to be greater than 2 (more than 2 vertices) so that the second term would always be positive.

The recurrence relation for the above algorithm can be written as:

$$T(n) = O(n^2) + 2T\Big(\frac{n}{\sqrt{2}}\Big)$$

By masters theorem, $a = 2$ and $b = \sqrt{2}$, hence,

$$T(n) = O(n^2 lg(n))$$

We can run the BETTER-MIN-CUT function $T = lg(n)ln(n)$ times, to increase the probability of finding a min cut. As shown above, probability of finding a min cut in one run is $1/lg(n)$. Thus, this algorithm fails only if all T attempts fail. The probability of success is therefore:

$$1 - \Big(1 - \frac{1}{lg(n)}\Big)^{lg(n)ln(n)} \geq 1 - e^{ln(n)} = 1 - \frac{1}{n}$$

The running time for this algorithm would then be $O(n^2 lg^2(n) ln(n))$. Note that we achieved the a polynomial fraction failure probability using Karger algorithm with a running time of $O(n^4 lg(n))$.

4 Overall analysis

The table 1 below, shows the probability of finding a min cut and running time of all the functions and algorithms discussed above.

Some major points to note here are, when we use Karger and Stein algorithm with number of trials as 1, instead of just the CONTRACT-GRAPH, even though there is a slight increase in the running time, we are able to increase the probability from $O(1/n^2)$ to $(O(1/lg(n))$.

Also, when we run Karger and Stein for more trials , small increase in number of trials, would give a considerable amount of increase in the probability of success.

One of the most important point to note here is that the Karger and Stein algorithm with T trials , gives us correct results with high probability and a running time which is lower than the running time of the deterministic Min Cut - Max flow algorithms that we have seen in previous lectures. The deterministic methods, where we knew exactly what is going on in every step takes $O(n^3)$ time, whereas these random algorithms do the same with high probability much faster.

Table 1: Comparison of probabilities and running time

Algorithm/Function	Probability of success	Running time
Contract-Graph	$\frac{2}{n(n-1)}$	$O(n^2)$
Karger $T = \frac{n(n-1)lg(n)}{2}$	$1 - \frac{1}{n}$	$O(n^4 lg(n))$
Karger $T = \frac{n^2(n-1)}{2}$	$1 - \frac{1}{e^n}$	$O(n^5)$
Karger and Stein $T = 1$	$\frac{1}{lg(n)}$	$O(n^2 lg(n))$
Karger and Stein $T = lg(n)ln(n)$	$1 - \frac{1}{n}$	$O(n^2 lg^3(n))$

5 Extensions

5.1 Weighted Graphs

To handle weighted graphs is a fairly minor alteration to the algorithm. When we are dealing with weighted graphs we simply include the weight of the edges in our graph representation. When randomly selecting and edge to collapse we need to take into the account the weight of the edge. The probability that an edge be selected to be collapsed needs to be scaled by the weight of the edge. This results in edges of higher weights being more likely to be selected. For example, an edge of weight 2 will have twice the probability of being selected to be collapsed.

5.2 Using Adjacency List

We represent an undirected graph as a symmetric directed graph using the adjacency list representation. Each vertex has a doubly linked list of edges adjacent to it and pointers to the beginning and the end of the list. An edge u, v is represented by two arcs, (u, v) and (v, u). These arcs have pointers to each other. An arc (u, v) appears on the adjacency list of u and has a pointer to v.

Suppose we contract an edge u, v. One way to implement the contraction is to do compact contraction as follows.

1. For each (v, w) on the adjacency list of v, replace the reverse arc (w, v) by (w, u). This operation takes $O(d(v))$ where $d(v)$ is the degree of v.

2. Append the arc list of v to the arc list of u. This takes constant time.
3. Delete v from $V(G)$. This takes constant time.
4. Delete self-loops in the adjacency list of u. This takes $O(d(u) + d(v))$ time.

The running time for this algorithm is of the order the sum of the degree of u and v.

$$O(d(u) + d(v))$$

It is worth noting that with very dense graphs this can be a worse than the $O(n)$ adjacency matrix implementation. This is due to the collapsing of an edge increasing the degree of v. Successive increases in the degree of a single vertex can cause the degree to increase to a size greater than the $O(n)$. Thus, the runtime can exceed the runtime of the adjacency matrix implementation of $O(n)$.

References

[1] Algorithms. Carnegie Mellon University Lecture, 2010. `http://www.cs.cmu.edu/afs/cs/academic/class/15451-s99/www/lectures/lect0318`.

[2] Chandra S. Chekuri, Andrew V. Goldberg, David R. Karger, Matthew S. Levine, and Cliff Stein. Experimental study of minimum cut algorithms. In *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, SODA '97, pages 324–333, Philadelphia, PA, USA, 1997. Society for Industrial and Applied Mathematics.

[3] Sanjoy Dasgupta. Ranomized algorithms 2. University of California at San Diego Lecture, 2010. `http://cseweb.ucsd.edu/~dasgupta/103/4b.pdf`.

[4] David R. Karger. Global min-cuts in rnc, and other ramifications of a simple min-out algorithm. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, SODA '93, pages 21–30, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics.

[5] David R. Karger and Clifford Stein. An ø(n2) algorithm for minimum cuts. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, STOC '93, pages 757–765, New York, NY, USA, 1993. ACM.

[6] David R. Karger and Clifford Stein. A new approach to the minimum cut problem. *J. ACM*, 43(4):601–640, July 1996.

[7] Kumar Piyush. Cse 373 analysis of algorithms. Florida State University Lecture, 2003. `http://www.compgeom.com/~piyush/teach/373/slides/lecture17.pdf`.