4. Karger's Algorithm

Global Minimum Cut

Cut - Set of edges you delete to make the graph disconnected

$$\text{Min-cut} = \text{Cut of minimum cardinality}$$

This is closely related to the max-flow problem.
First, we'll work with the max-flow problem where all edges have capacity 1.
As it turns out, if we have a source to sink, the max-flow between the source to sink is the minimum cut between the source and sink.

s-t cut - If you pick a source and destination (s and t respectively) we are finding the minimum cut which disconnects the source from the destination. This can be done by finding the max-flow between s-t. This can be used to find the global min-cut as we can just iterate over all the possible s, t pairs in the graph and running max flow.

Ford Fulkerson - $O(|c| \cdot (|E| + |V|))$, where c is the max flow of the graph.

In an unweighted graph, the capacity is bounded by the number of edges, hence it can be rewritten as $O(|E|(|E| + |V|))$.
If we are working with a simple graph, we can instead write it as $O(|V|(|E| + |V|))$ as there are no multiedges, and the bound on the flow is the min out degree of s and in degree of t.

Hence, we can use the max flow algorithm and find the global minimum cut by running it between every pair.

Without loss of generality, we just assume a connected graph (as for disconnected it's just 0).

Can we get better guarantees with randomization?

Karger's Algorithm

Supernode - A set of vertices
Superedge - For X, Y which are supernodes, the super edge between them is a set which contains all the edges (x, y) such that $x \in X, y \in Y$.

We had some structure on the initial graph with vertices and edges

Say we had a graph of 4 vertices, and edges

$$(1, 4), (2, 4), (3, 4)$$

We can define 2 super nodes

$$X = \{1, 2, 3\}$$

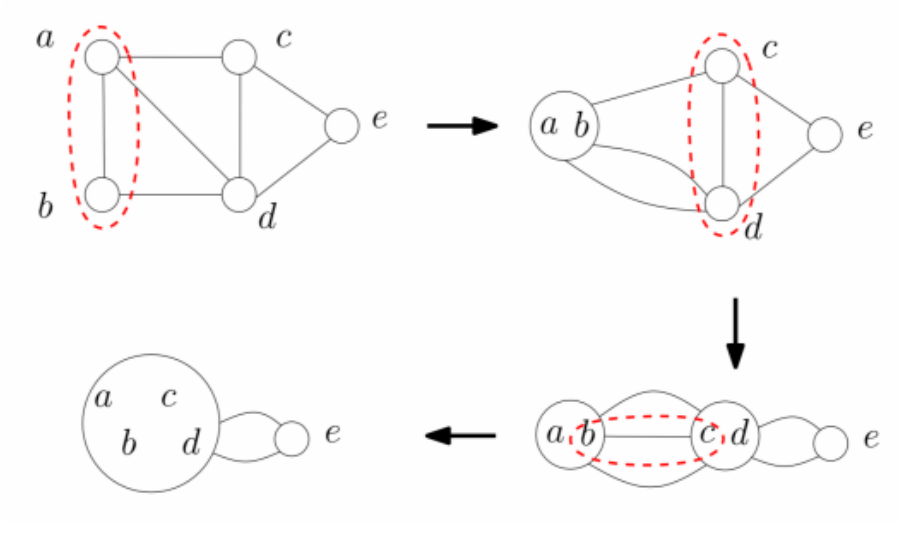
$$Y = \{4\}$$

And the set of super edges is defined as

$$E_{XY} = \{(u, v) | u \in X, v \in Y\}$$

Karger's algorithm roughly works as follows -

1. While number of supernodes > 2, pick an edge from E, and merge the two end points of that edge
2. Output the superedge when $\#\text{supernodes} = 2$



In this algorithm, we continue to merge supernodes till the the number of supernodes is 2, and then we take a look at the cardinality of the remaining super edge between the superedges, which is a cut for the graph.

Formally,
Γ : Set of supernodes
F : Set of superedges
$V(U)$: nodes in supernode(n)
$\bar{v}$: supernode containing v

Initially $\Gamma \leftarrow V, F \leftarrow E$
$\text{Merge}(a, b, \Gamma)$
Suppose we want to merge $a, b \in \Gamma$

1. $x \leftarrow$ new empty supernode
2. $V(x) \leftarrow V(a) \cup V(b)$
3. For each $d \in \Gamma \backslash \{a, b\}$, $E_{dx} \leftarrow E_{da} \cup E_{db}$
4. $\Gamma \leftarrow (\Gamma \backslash \{a, b\}) \cup \{x\}$

$\text{Karger}(G(V, E))$

1. Initialize (Γ, E)
2. While $|\Gamma| > 2$
    1. Pick an edge e from F randomly.
    2. $\Gamma \leftarrow \text{Merge}(\bar{u}, \bar{v}, \Gamma)$
    3. $F \leftarrow F \backslash E_{\bar{u}, \bar{v}}$
3. Return the only superedge

Karger's algorithm states that the superedge is definitely a cut, and a min cut with some probability.

Time Complexity Analysis of Karger's

We can see that the number of iterations of step 2 in the above algorithm is $\leq n - 2$ as each merge reduces the number of supernodes by 1.
Hence, the time complexity is $O(n \cdot (|\text{union}| + |\text{set union}|))$

Claim: If k is the min cut size then G has at least $\frac{kn}{2}$ edges.
We know that if k is the min cut size, then the min degree of any node is k.
If there is some node of degree $k - 1$ or lesser, we could just remove all the edges incident on it to make it disconnected, achieving a min cut less than k.

Now, we can get a lower bound on the number of edges.

$$\#\text{Edges} \geq \frac{\text{minimum degree} * n}{2}$$

(The min degree doesnt have to be exactly k, you can take the example of two densely connected components connected by one edge)

Correctness Analysis of Karger's

If we want this algorithm to return the min cut, we need to ensure that in each of the iterations, we do not want the end points of an edge from min-cut to be contracted/merged.
The selection of edges in each iteration is random.

E_i- Event that an edge from the mincut does not get contracted in the ith iteration. $1 \leq i \leq n-2$
If it's a connected graph, it'll need $n-2$ iterations. If it's disonnected, then it'll run in less iterations
For the algorithm to run correct, all of these events must occur, i.e

$$\cap_{i=1}^{n-2} E_i$$

We want to find the probability of this occurring.

In this analysis, we're assuming that we only pick valid edges (i.e edges connecting two different supernodes)

Hence, we want to find

$$Pr[\cap_{i=1}^{n-2} E_i] = Pr[E_1] \times Pr[E_2|E_1] \times \cdots \times Pr[E_{n-2}| \cap_{j=1}^{n-3} E_j]$$

$$Pr[E_1] = \frac{|E| - k}{|E|} \geq 1 - \frac{k}{\frac{kn}{2}} = 1 - \frac{2}{n}$$

$$Pr[E_2|E_1] = \frac{|F_1| - k}{|F_1|} \geq 1 - \frac{2}{n-1}$$

Now, we claim that

$$Pr[E_i| \cap_{j=1}^{i-1} E_j] \geq 1 - \frac{2}{n-(i-1)}$$

Say the first iteration E_1 occurs. After this, the number of nodes in the graph is $n-1$, hence the minimum number of edges in the graph is $\frac{k(n-1)}{2}$ now. Making this generalized, just before E_i occurs, there will be $n-(i-1)$ vertices in the graph, implying that there are at least $\frac{k(n-1)}{2}$ edges, which leads to

$$\frac{|E_j|-k}{|E_j|} = 1 - \frac{2}{n-j+1}$$

This would give us

$$Pr[\cap_{i=1}^{n-2} E_i] \geq \prod_{i=1}^{n-2}\left(1-\frac{2}{n-(i-1)}\right) = \prod_{i=1}^{n-2}\left(\frac{n-i-1}{n-i+1}\right) = \frac{2}{n(n-1)}$$

Boosting Success Probability

- Run a random experiment s times independently. Probability that it gives the correct answer is p.
- Probability that none of the s trials give me the correct answer is $(1-p)^s$. Therefore with probability of $1-(1-p)^s$, the algorithm gives use the correct answer.
  In the case of mincut, $p = \frac{1}{n^2}$
  Therfore the probability of all failing are

  $$\leq (1-p)^s \leq \left(1-\frac{1}{n^2}\right)^s \approx e^{-\frac{s}{n^2}}$$

  If $s \approx n^2$, then

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

  Therefore the total time complexity is the time taken to run one instance of Karger's algorithm, into s, the number of times we run Karger's.
  Hence, we get

  $$T \cdot s = n \cdot |\text{merge}| \cdot n^2 = n^3 \cdot |\text{merge}|$$

  Theorem: With probability $1-e^{-\frac{s}{n^2}}$ and in running time $s \cdot (n \cdot |\text{merge}|)$ min cut can be found.

Question - We only considered unweighted case, but in flows we consider the weighted case. What do we do? We no longer have a bound on the number of

edges, but we have a bound on the weight of edges.
Therefore the statement

$$|E| \geq \frac{kn}{2}$$

doesn't hold anymore

Monte Carlo vs Las Vegas

Quicksort

- In Quicksort, we had an expected running time of $O(n \log n)$. Worst case is $O(n^2)$
- The final solution of quicksort is always correct
- This is a **Las Vegas** algorithm, run time is probabilistic.

Randomized Min-cut

- Runtime is $n - 2$ iterations with merge (can be done with DSU, which has the inverse Ackermann function time complexity)
- The correctness is not guaranteed.
- This is a **Monte Carlo** algorithm, correctness is probabilistic.