Chapter 15

Min Cut

To acknowledge the corn - This purely American expression means to admit the losing of an argument, especially in regard to a detail; to retract; to admit defeat. It is over a hundred years old. Andrew Stewart, a member of Congress, is said to have mentioned it in a speech in 1828. He said that haystacks and cornfields were sent by Indiana, Ohio and Kentucky to Philadelphia and New York. Charles A. Wickliffe, a member from Kentucky questioned the statement by commenting that haystacks and cornfields could not walk. Stewart then pointed out that he did not mean literal haystacks and cornfields, but the horses, mules, and hogs for which the hay and corn were raised. Wickliffe then rose to his feet, and said, "Mr. Speaker, I acknowledge the corn".

Funk, Earle, A Hog on Ice and Other Curious Expressions

By Sariel Har-Peled, March 19, 2024[①]

15.1. Branching processes – Galton-Watson Process

15.1.1. The problem

In the 19th century, Victorians were worried that aristocratic surnames were disappearing, as family names passed on only through the male children. As such, a family with no male children had its family name disappear. So, imagine the number of male children of a person is an independent random variable $X \in \{0, 1, 2, \ldots\}$. Starting with a single person, its family (as far as male children are concerned) is a random tree with the degree of a node being distributed according to X. We continue recursively in constructing this tree, again, sampling the number of children for each current leaf according to the distribution of X. It is not hard to see that a family disappears if $\mathbb{E}[X] \leq 1$, and it has a constant probability of surviving if $\mathbb{E}[X] > 1$.

Francis Galton asked the question of what is the probability of such a blue-blood family name to survive, and this question was answered by Henry William Watson [WG75]. The Victorians were worried about strange things, see [Gre69] for a provocatively titled article from the period, and [Ste12] for a more recent take on this issue.

Of course, since infant mortality is dramatically down (as is the number of aristocrat males dying to maintain the British empire), the probability of family names to disappear is now much lower than it was in the 19th century. Interestingly, countries with family names that were introduced long time ago have very few surnames (i.e., Korean have 250 surnames, and three surnames form 45% of the population). On the other hand, countries that introduced surnames more recently have dramatically more surnames (for example, the Dutch have surnames only for the last 200 years, and there are 68,000 different family names).

Here we are going to look on a very specific variant of this problem. Imagine that starting with a single male. A male has exactly two children, and one of them is a male with probability half (i.e., the Y-chromosome is being passed only to its male children). As such, the natural question is what is the probability that h generations down, there is a male decedent that all his ancestors are male (i.e., it caries the original family name, and the original Y-chromosome).

15.1.2. On coloring trees

Let T_h be a complete binary tree of height h. We randomly color its edges by black and white. Namely, for each edge we independently choose its color to be either black or white, with equal probability (say, black indicates the child is male). We are interested in the event that there exists a path from the root of T_h to one of its leafs, that is all black. Let $\mathcal{E}_h$ denote this event, and let $\rho_h = \mathbb{P}[\mathcal{E}_h]$. Observe that $\rho_0 = 1$ and $\rho_1 = 3/4$ (see below).

To bound this probability, consider the root u of T_h and its two children u_l and u_r. The probability that there is a black path from u_l to one of its children is ρ_{h-1}, and as such, the probability that there is a black path from u through u_l to a leaf of the subtree of u_l is $\mathbb{P}[\text{the edge } uu_l \text{ is colored black}] \cdot \rho_{h-1} = \rho_{h-1}/2$. As such, the probability that there is no black path through u_l is $1 - \rho_{h-1}/2$. As such, the probability of not having a black path from u to a leaf (through either children) is $(1 - \rho_{h-1}/2)^2$. In particular, there desired probability, is the complement; that is

$$\rho_h = 1 - \left(1 - \frac{\rho_{h-1}}{2}\right)^2 = \frac{\rho_{h-1}}{2}\left(2 - \frac{\rho_{h-1}}{2}\right) = \rho_{h-1} - \frac{\rho_{h-1}^2}{4} = f(\rho_{h-1}) \qquad \text{for} \qquad f(x) = x - x^2/4.$$

The starting values are $\rho_0 = 1$, and $\rho_1 = 3/4$.

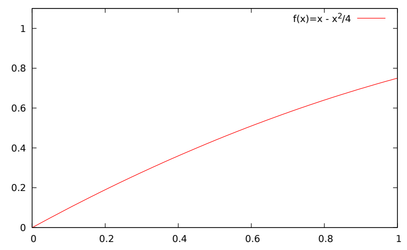


Figure 15.1: A graph of the function $f(x) = x - x^2/4$.

Lemma 15.1.1. *We have that $\rho_h \geq 1/(h+1)$.*

Proof: (Feel free to skip reading.) The proof is by induction. For $h = 1$, we have $\rho_1 = 3/4 \geq 1/(1+1)$.

Observe that $\rho_h = f(\rho_{h-1})$ for $f(x) = x - x^2/4$, and $f'(x) = 1 - x/2$. As such, $f'(x) > 0$ for $x \in [0,1]$ and $f(x)$ is increasing in the range $[0,1]$. As such, by induction, we have that

$$\rho_h = f(\rho_{h-1}) \geq f\left(\frac{1}{(h-1)+1}\right) = \frac{1}{h} - \frac{1}{4h^2}.$$

We need to prove that $\rho_h \geq 1/(h+1)$, which is implied by the above if

$$\frac{1}{h} - \frac{1}{4h^2} \geq \frac{1}{h+1} \quad \Leftrightarrow \quad 4h(h+1) - (h+1) \geq 4h^2 \quad \Leftrightarrow \quad 4h^2 + 4h - h - 1 \geq 4h^2 \quad \Leftrightarrow \quad 3h \geq 1,$$

which trivially holds. ■

Lemma 15.1.2. *We have that $\rho_h = O(1/h)$.*

Proof: (Feel free to skip reading.) We prove the claim for infinite number of values of h – the claim then follows for all h by fiddling with the constants. The claim trivially holds for small values of h. For any $j > 0$, let h_j be the minimal index such that $\rho_{h_j} \leq 1/2^j$. It is easy to verify that $\rho_{h_j} \geq 1/2^{j+1}$. We claim (mysteriously) that

$$h_{j+1} - h_j \leq \frac{\rho_{h_j} - \rho_{h_{j+1}}}{(\rho_{h_{j+1}})^2/4}.$$

Indeed, ρ_{k+1} is the number resulting from removing $\rho_k^2/4$ from ρ_k. Namely, the sequence $\rho_1, \rho_2, \ldots$ is a monotonically decreasing sequence of numbers in the interval $[0,1]$, where the gaps between consecutive numbers decreases. In particular, to get from ρ_{h_j} to $\rho_{h_{j+1}}$, the gaps used were of size at least $\Delta = \left(\rho_{h_{j+1}}\right)^2$, which means that there are at least $(\rho_{h_j} - \rho_{h_{j+1}})/\Delta - 1$ numbers in the series between these two elements. As such, since $\rho_{h_j} \le 1/2^j$ and $\rho_{h_{j+1}} \ge 1/2^{j+2}$, we have

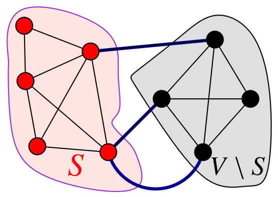
$$h_{j+1} - h_j \le \frac{\rho_{h_j} - \rho_{h_{j+1}}}{(\rho_{h_{j+1}})^2/4} \le \frac{1/2^j - 1/2^{j+2}}{1/2^{2(j+2)+2}} \le 2^{2j+6}/2^j = 2^{j+6}.$$

This implies that $h_j \le (h_j - h_{j-1}) + (h_{j-1} - h_{j-2}) + \cdots + (h_1 - h_0) \le 2^{j+6}$. As such, we have $\rho_{h_j} \le 1/2^j \le 2^6/2^{j+6} \le 2^6/h_j$, which implies the claim. ■

15.2. Min Cut

15.2.1. Problem Definition

Let $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ be an undirected graph with n vertices and m edges. We are interested in ***cuts*** in G.



Definition 15.2.1. A ***cut*** in G is a partition of the vertices of V into two sets S and $\mathsf{V} \setminus S$, where the edges of the cut are

$$(S, \mathsf{V} \setminus S) = \left\{ uv \;\middle|\; u \in S, v \in \mathsf{V} \setminus S, \text{ and } uv \in E \right\},$$

where $S \neq \emptyset$ and $\mathsf{V} \setminus S \neq \emptyset$. We will refer to the number of edges in the cut $(S, \mathsf{V} \setminus S)$ as the *size of the cut*. For an example of a cut, see figure on the right.

We are interested in the problem of computing the ***minimum cut*** (i.e., ***mincut***), that is, the cut in the graph with minimum cardinality. Specifically, we would like to find the set $S \subseteq \mathsf{V}$ such that $(S, \mathsf{V} \setminus S)$ is as small as possible, and S is neither empty nor $\mathsf{V} \setminus S$ is empty.

15.2.2. Some Definitions

We remind the reader of the following concepts. The ***conditional probability*** of X given Y is $\mathbb{P}\left[X = x \;\middle|\; Y = y\right] = \mathbb{P}[(X = x) \cap (Y = y)]/\mathbb{P}[Y = y]$. An equivalent, useful restatement of this is that

$$\mathbb{P}[(X = x) \cap (Y = y)] = \mathbb{P}\left[X = x \;\middle|\; Y = y\right] \cdot \mathbb{P}[Y = y]. \tag{15.1}$$

The following is easy to prove by induction using Eq. (15.1).

Lemma 15.2.2. *Let $\mathcal{E}_1, \ldots, \mathcal{E}_n$ be n events which are not necessarily independent. Then,*

$$\mathbb{P}[\cap_{i=1}^n \mathcal{E}_i] = \mathbb{P}[\mathcal{E}_1] * \mathbb{P}\left[\mathcal{E}_2 \;\middle|\; \mathcal{E}_1\right] * \mathbb{P}\left[\mathcal{E}_3 \;\middle|\; \mathcal{E}_1 \cap \mathcal{E}_2\right] * \ldots * \mathbb{P}\left[\mathcal{E}_n \;\middle|\; \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{n-1}\right].$$

15.3. The Algorithm

The basic operation used by the algorithm is ***edge contraction***, depicted in Figure 15.2. We take an edge $e = xy$ in G and merge the two vertices into a single vertex. The new resulting graph is denoted by G/xy. Note, that we remove self loops created by the contraction. However, since the resulting graph is no longer a regular graph, it has parallel edges – namely, it is a multi-graph. We represent a multi-graph, as a regular graph with multiplicities on the edges. See Figure 15.3.

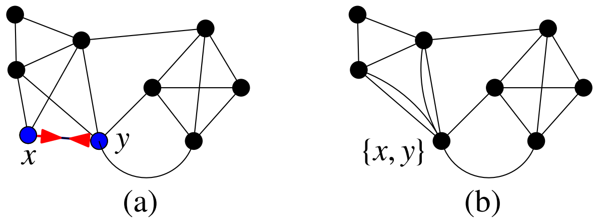


Figure 15.2: (a) A contraction of the edge xy. (b) The resulting graph.

The edge contraction operation can be implemented in $O(n)$ time for a graph with n vertices. This is done by merging the adjacency lists of the two vertices being contracted, and then using hashing to do the fix-ups (i.e., we need to fix the adjacency list of the vertices that are connected to the two vertices).

Note, that the cut is now computed counting multiplicities (i.e., if e is in the cut and it has weight w, then the contribution of e to the cut weight is w).

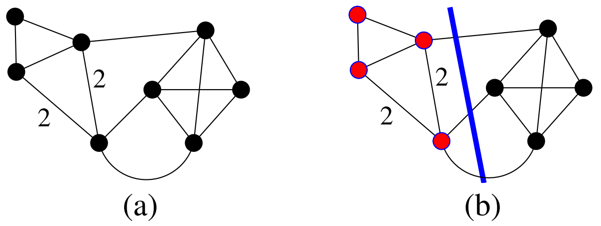


Figure 15.3: (a) A multi-graph. (b) A minimum cut in the resulting multi-graph.

Observation 15.3.1. *A set of vertices in G/xy corresponds to a set of vertices in the graph G. Thus a cut in G/xy always corresponds to a valid cut in G. However, there are cuts in G that do not exist in G/xy. For example, the cut $S = \{x\}$, does not exist in G/xy. As such, the size of the minimum cut in G/xy is at least as large as the minimum cut in G (as long as G/xy has at least one edge). Since any cut in G/xy has a corresponding cut of the same cardinality in G.*

Our algorithm works by repeatedly performing edge contractions. This is beneficial as this shrinks the underlying graph, and we would compute the cut in the resulting (smaller) graph. An "extreme" example of this, is shown in Figure 15.4, where we contract the graph into a single edge, which (in turn) corresponds to a cut in the original graph. (It might help the reader to think about each vertex in the contracted graph, as corresponding to a connected component in the original graph.)

Figure 15.4 also demonstrates the problem with taking this approach. Indeed, the resulting cut is not the minimum cut in the graph.

So, why did the algorithm fail to find the minimum cut in this case?[2] The failure occurs because of the contraction at Figure 15.4 (e), as we had contracted an edge in the minimum cut. In the new graph, depicted in Figure 15.4 (f), there is no longer a cut of size 3, and all cuts are of size 4 or more. Specifically, the algorithm succeeds only if it does not contract an edge in the minimum cut.

Observation 15.3.2. *Let $e_1, \ldots, e_{n-2}$ be a sequence of edges in G, such that none of them is in the minimum cut, and such that $\mathsf{G}' = G/\{e_1, \ldots, e_{n-2}\}$ is a single multi-edge. Then, this multi-edge corresponds to a minimum cut in G.*

Note, that the claim in the above observation is only in one direction. We might be able to still compute a minimum cut, even if we contract an edge in a minimum cut, the reason being that a minimum cut is not unique. In particular, another minimum cut might survived the sequence of contractions that destroyed other minimum cuts.

[2]Naturally, if the algorithm had succeeded in finding the minimum cut, this would have been **our** success.

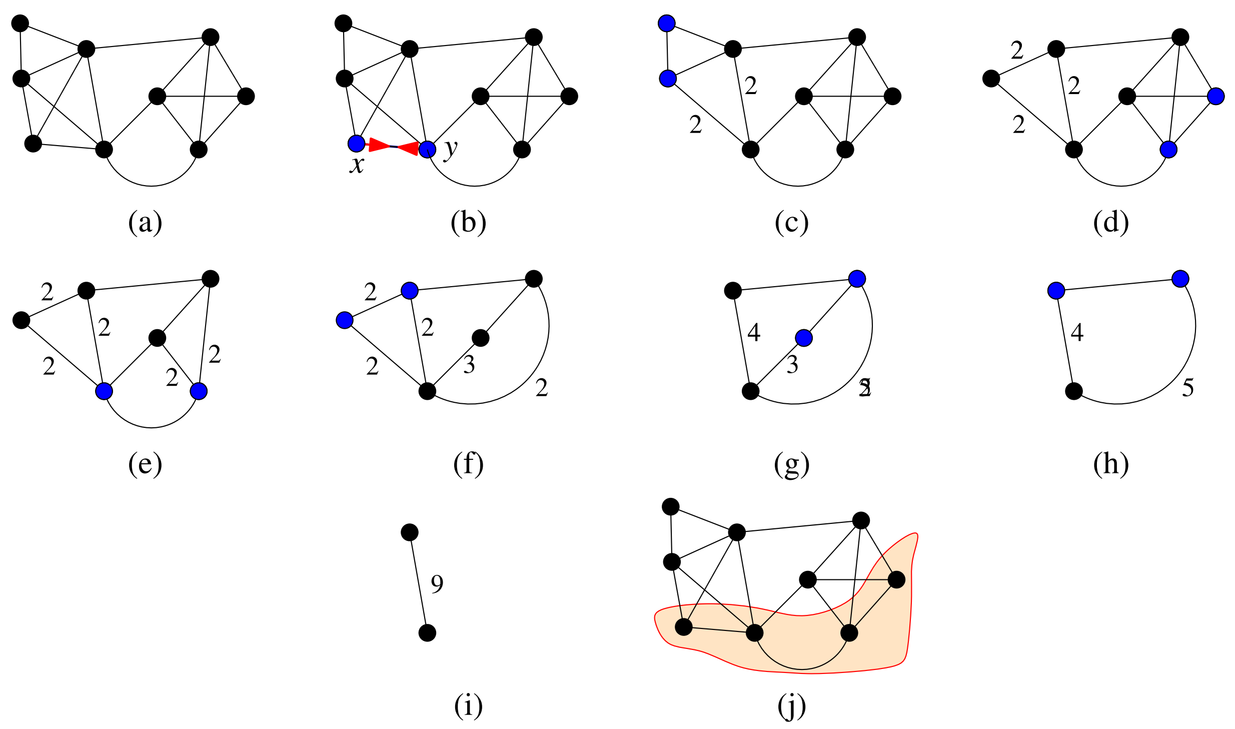


Figure 15.4: (a) Original graph. (b)–(j) a sequence of contractions in the graph, and (h) the cut in the original graph, corresponding to the single edge in (h). Note that the cut of (h) is not a mincut in the original graph.

```
Algorithm MinCut(G)
  $G_0 \leftarrow G$
  $i = 0$
  while $G_i$ has more than two vertices do
      Pick randomly an edge $e_i$ from the edges of $G_i$
      $G_{i+1} \leftarrow G_i / e_i$
      $i \leftarrow i + 1$
  Let $(S, V \setminus S)$ be the cut in the original graph
          corresponding to the single edge in $G_i$
  return $(S, V \setminus S)$.
```

Figure 15.5: The minimum cut algorithm.

Using Observation 15.3.2 in an algorithm is problematic, since the argumentation is circular, how can we find a sequence of edges that are not in the cut without knowing what the cut is? The way to slice the Gordian knot here, is to randomly select an edge at each stage, and contract this random edge.

See Figure 15.5 for the resulting algorithm **MinCut**.

15.3.1. Analysis

15.3.1.1. The probability of success

Naturally, if we are extremely lucky, the algorithm would never pick an edge in the mincut, and the algorithm would succeed. The ultimate question here is what is the probability of success. If it is relatively "large" then this algorithm is useful since we can run it several times, and return the best result computed. If on the other hand, this probability is tiny, then we are working in vain since this approach would not work.

Lemma 15.3.3. *If a graph* G *has a minimum cut of size* k *and* G *has* n *vertices, then* $|E(G)| \geq \frac{kn}{2}$.

Proof: Each vertex degree is at least k, otherwise the vertex itself would form a minimum cut of size smaller than k. As such, there are at least $\sum_{v \in \mathsf{V}} \text{degree}(v)/2 \geq nk/2$ edges in the graph. ■

Lemma 15.3.4. *Fix a specific minimum cut* $C = (S, \overline{S})$ *in the graph. If we pick in random an edge* e *from a graph* G*, uniformly at random, then with probability at most* $2/n$ *it belongs to the minimum cut* C.

Proof: There are at least $nk/2$ edges in the graph and exactly k edges in the minimum cut. Thus, the probability of picking an edge from the minimum cut is smaller then $k/(nk/2) = 2/n$. ■

The following lemma shows (surprisingly) that **MinCut** succeeds with reasonable probability.

Lemma 15.3.5. **MinCut** *outputs the mincut with probability* $\geq \dfrac{2}{n(n-1)}$.

7

Proof: Let $\mathcal{E}_i$ be the event that e_i is not in the minimum cut of G_i. By Observation 15.3.2, **MinCut** outputs the minimum cut if the events $\mathcal{E}_0, \ldots, \mathcal{E}_{n-3}$ all happen (namely, all edges picked are outside the minimum cut).

By Lemma 15.3.4, it holds $\mathbb{P}\Big[\mathcal{E}_i \,\Big|\, \mathcal{E}_0 \cap \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}\Big] \geq 1 - \dfrac{2}{|\mathsf{V}(G_i)|} = 1 - \dfrac{2}{n-i}$. Implying that

$$\Delta = \mathbb{P}[\mathcal{E}_0 \cap \ldots \cap \mathcal{E}_{n-3}] = \mathbb{P}[\mathcal{E}_0] \cdot \mathbb{P}\Big[\mathcal{E}_1 \,\Big|\, \mathcal{E}_0\Big] \cdot \mathbb{P}\Big[\mathcal{E}_2 \,\Big|\, \mathcal{E}_0 \cap \mathcal{E}_1\Big] \cdot \ldots \cdot \mathbb{P}\Big[\mathcal{E}_{n-3} \,\Big|\, \mathcal{E}_0 \cap \ldots \cap \mathcal{E}_{n-4}\Big].$$

As such, we have

$$\begin{aligned}\Delta &\geq \prod_{i=0}^{n-3}\left(1 - \frac{2}{n-i}\right) = \prod_{i=0}^{n-3} \frac{n-i-2}{n-i} \\ &= \frac{\cancel{n-2}}{n} * \frac{\cancel{n-3}}{n-1} * \frac{\cancel{n-4}}{\cancel{n-2}} * \frac{\cancel{n-5}}{\cancel{n-3}} * \frac{\cancel{n-6}}{\cancel{n-4}} * \cdots * \frac{\cancel{3}}{\cancel{5}} * \frac{2}{\cancel{4}} * \frac{1}{\cancel{3}} \\ &= \frac{2}{n(n-1)}.\end{aligned}$$

■

15.3.1.2. Running time analysis.

Observation 15.3.6. **MinCut** *runs in* $O(n^2)$ *time.*

Observation 15.3.7. *The algorithm always outputs a cut, and the cut is not smaller than the minimum cut.*

Definition 15.3.8. (informal) Amplification is the process of running an experiment again and again till the things we want to happen, with good probability, do happen.

Let **MinCutRep** be the algorithm that runs **MinCut** $n(n-1)$ times and return the minimum cut computed in all those independent executions of **MinCut**.

Lemma 15.3.9. *The probability that* **MinCutRep** *fails to return the minimum cut is* < 0.14.

Proof: The probability of failure of **MinCut** to output the mincut in each execution is at most $1 - \frac{2}{n(n-1)}$, by Lemma 15.3.5. Now, **MinCutRep** fails, only if all the $n(n-1)$ executions of **MinCut** fail. But these executions are independent, as such, the probability to this happen is at most

$$\left(1 - \frac{2}{n(n-1)}\right)^{n(n-1)} \leq \exp\left(-\frac{2}{n(n-1)} \cdot n(n-1)\right) = \exp(-2) < 0.14,$$

since $1 - x \leq e^{-x}$ for $0 \leq x \leq 1$. ■

Theorem 15.3.10. *One can compute the minimum cut in* $O(n^4)$ *time with constant probability to get a correct result. In* $O\left(n^4 \log n\right)$ *time the minimum cut is returned with high probability.*

15.3.2. An alternative implementation using MST

The algorithm. The above algorithm can be restated as follows. Randomly assign weights to the edges of G (say, by picking numbers in $[0, 1]$). Next, compute the MST T of the graph according to these weights. Remove the heaviest edge in the MST. The resulting partition of T into two trees, corresponds to a cut in the original graph. Return this cut as a candidate to be the minimum cut.

The analysis. To see that this algorithm is equivalent to **MinCut** (Figure 15.5), observe that the contraction algorithm simulates Kruskal's MST algorithm when run on randomly weighted edges. First, imagine implementing **MinCut** so that it keeps parallel edges. Then, the edges connecting two vertices that are not contracted are exactly the edges between the two connected components. Picking a random edge to contract, is equivalent to picking the edge with the minimum random weight. Thus, the MST algorithm here just simulates **MinCut** (or vice versa).

A small optimization. It is possible to compute the heaviest edge in the MST, and the partition it induces in (deterministic) linear time – it is a nice example of the search and prune technique.

Exercise 15.3.11. Given a graph G with weights on the edges, show how to compute the maximum weight edge in the MST of G in $O(n + m)$ time, where n are m are the number of vertices and edges of G, respectively.

Thus, this yields a $O(n + m)$ implementation of **MinCut**. We get the following result.

Lemma 15.3.12. **MinCut** *can implemented to run in* $O(n + m)$ *time, and it outputs the mincut with probability* $\geq \dfrac{2}{n(n-1)}$.

```
Contract ( G, $t$ )
begin
    while $|(G)| > t$ do
        Pick a random edge $e$ in G.
        G $\leftarrow G/e$
    return G
end
```

```
FastCut(G = $(V, E)$)
    G – multi-graph
begin
    $n \leftarrow |V(G)|$
    if $n \leq 6$ then
        Compute (via brute force) minimum cut
        of G and return cut.
    $t \leftarrow \left\lceil 1 + n/\sqrt{2} \right\rceil$
    $H_1 \leftarrow$ Contract$(G, t)$
    $H_2 \leftarrow$ Contract$(G, t)$
    /* Contract is randomized!!! */
    $X_1 \leftarrow$ FastCut$(H_1)$,
    $X_2 \leftarrow$ FastCut$(H_2)$
    return minimum cut out of $X_1$ and $X_2$.
end
```

Figure 15.6: **Contract**(G, t) shrinks G till it has only t vertices. **FastCut** computes the minimum cut using **Contract**.

15.4. A faster algorithm

The algorithm presented in the previous section is extremely simple. Which raises the question of whether we can get a faster algorithm[3]?

So, why **MinCutRep** needs so many executions? Well, the probability of success in the first ν iterations is

$$\mathbb{P}[\mathcal{E}_0 \cap \ldots \cap \mathcal{E}_{\nu-1}] \geq \prod_{i=0}^{\nu-1}\left(1 - \frac{2}{n-i}\right) = \prod_{i=0}^{\nu-1}\frac{n-i-2}{n-i}$$
$$= \frac{n-2}{n} * \frac{n-3}{n-1} * \frac{n-4}{n-2}\ldots = \frac{(n-\nu)(n-\nu-1)}{n\cdot(n-1)}. \tag{15.2}$$

Namely, this probability deteriorates very quickly toward the end of the execution, when the graph becomes small enough. (To see this, observe that for $\nu = n/2$, the probability of success is roughly $1/4$, but for $\nu = n - \sqrt{n}$ the probability of success is roughly $1/n$.)

So, the key observation is that as the graph get smaller the probability to make a bad choice increases. So, instead of doing the amplification from the outside of the algorithm, we will run the new algorithm more times when the graph is smaller. Namely, we put the amplification directly into the algorithm.

The basic new operation we use is **Contract**, depicted in Figure 15.6, which also depict the new algorithm **FastCut**.

Lemma 15.4.1. *The running time of* **FastCut**(G) *is* $O\left(n^2 \log n\right)$*, where* $n = |V(G)|$.

Proof: Well, we perform two calls to **Contract**(G, t) which takes $O(n^2)$ time. And then we perform two recursive calls on the resulting graphs. We have

$$T(n) = O(n^2) + 2T(n/\sqrt{2}).$$

The solution to this recurrence is $O\left(n^2 \log n\right)$ as one can easily (and should) verify. ■

[3]This would require a more involved algorithm, that is life.

Exercise 15.4.2. Show that one can modify **FastCut** so that it uses only $O(n^2)$ space.

Lemma 15.4.3. *The probability that* **Contract**$(\mathsf{G}, n/\sqrt{2})$ *had not contracted the minimum cut is at least* $1/2$.

Namely, the probability that the minimum cut in the contracted graph is still a minimum cut in the original graph is at least $1/2$.

Proof: Just plug in $\nu = n - t = n - \left\lceil 1 + n/\sqrt{2} \right\rceil$ into Eq. (15.2). We have

$$\mathbb{P}\left[\mathcal{E}_0 \cap \ldots \cap \mathcal{E}_{n-t}\right] \geq \frac{t(t-1)}{n \cdot (n-1)} = \frac{\left\lceil 1 + n/\sqrt{2} \right\rceil \left(\left\lceil 1 + n/\sqrt{2} \right\rceil - 1 \right)}{n(n-1)} \geq \frac{1}{2}. \qquad \blacksquare$$

The following lemma bounds the probability of success.

Lemma 15.4.4. **FastCut** *finds the minimum cut with probability larger than* $\Omega(1/\log n)$.

Proof: Let T_h be the recursion tree of the algorithm of depth $h = \Theta(\log n)$. Color an edge of recursion tree by black if the contraction succeeded. Clearly, the algorithm succeeds if there is a path from the root to a leaf that is all black. This is exactly the settings of Lemma 15.1.1, and we conclude that the probability of success is at least $1/(h+1) = \Theta(1/\log n)$, as desired. $\blacksquare$

Exercise 15.4.5. Prove, that running **FastCut** repeatedly $c \cdot \log^2 n$ times, guarantee that the algorithm outputs the minimum cut with probability $\geq 1 - 1/n^2$, say, for c a constant large enough.

Theorem 15.4.6. *One can compute the minimum cut in a graph* G *with* n *vertices in* $O(n^2 \log^3 n)$ *time. The algorithm succeeds with probability* $\geq 1 - 1/n^2$.

Proof: We do amplification on **FastCut** by running it $O(\log^2 n)$ times. The running time bound follows from Lemma 15.4.1. The bound on the probability follows from Lemma 15.4.4, and using the amplification analysis as done in Lemma 15.3.9 for **MinCutRep**. $\blacksquare$

15.5. Bibliographical Notes

The **MinCut** algorithm was developed by David Karger during his PhD thesis in Stanford. The fast algorithm is a joint work with Clifford Stein. The basic algorithm of the mincut is described in [MR95, pages 7–9], the faster algorithm is described in [MR95, pages 289–295].

Galton-Watson process. The idea of using coloring of the edges of a tree to analyze **FastCut** might be new (i.e., Section 15.1.2).

References

[Gre69] W. Greg. *Why are Women Redundant?* Trübner, 1869.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge, UK: Cambridge University Press, 1995.

[Ste12] E. Steinlight. Why novels are redundant: sensation fiction and the overpopulation of literature. *ELH*, 79(2): 501–535, 2012.

[WG75] H. W. Watson and F. Galton. On the probability of the extinction of families. *J. Anthrop. Inst. Great Britain*, 4: 138–144, 1875.