Chapter 13

Min Cut

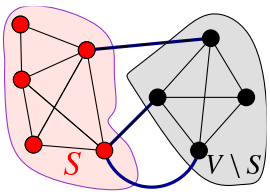
By Sariel Har-Peled, December 30, 2014[①] Version: 1.0

> I built on the sand
> And it tumbled down,
> I built on a rock
> And it tumbled down.
> Now when I build, I shall begin
> With the smoke from the chimney.
> – Leopold Staff, Foundations.

13.1. Min Cut

13.1.1. Problem Definition

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



Definition 13.1.1. A ***cut*** in G is a partition of the vertices of V into two sets S and $V \setminus S$, where the edges of the cut are

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

where $S \neq \emptyset$ and $V \setminus S \neq \emptyset$. We will refer to the number of edges in the cut $(S, V \setminus S)$ 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 $S \subseteq V$ such that $(S, V \setminus S)$ is as small as possible, and S is neither empty nor $V \setminus S$ is empty.

13.1.2. Some Definitions

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

$$\mathbf{Pr}\Big[(X = x) \cap (Y = y)\Big] = \mathbf{Pr}\Big[X = x \;\Big|\; Y = y\Big] \cdot \mathbf{Pr}[Y = y]\,. \tag{13.1}$$

Two events X and Y are ***independent***, if $\mathbf{Pr}\left[X = x \cap Y = y\right] = \mathbf{Pr}\left[X = x\right] \cdot \mathbf{Pr}\left[Y = y\right]$. In particular, if X and Y are independent, then $\mathbf{Pr}\left[X = x \;\middle|\; Y = y\right] = \mathbf{Pr}\left[X = x\right]$.

The following is easy to prove by induction using Eq. (13.1).

Lemma 13.1.2. *Let $\mathcal{E}_1, \ldots, \mathcal{E}_n$ be n events which are not necessarily independent. Then,*

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

13.2. The Algorithm

The basic operation used by the algorithm is ***edge contraction***, depicted in Figure 13.1. 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 13.2.

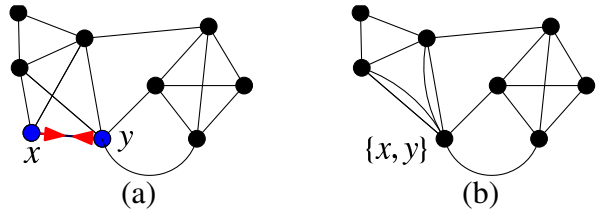


Figure 13.1: (a) A contraction of the edge xy. (b) The resulting graph.

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).

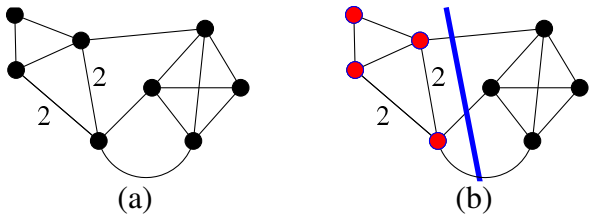


Figure 13.2: (a) A multi-graph. (b) A minimum cut in the resulting multi-graph.

Observation 13.2.1. *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 13.3, 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 13.3 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 13.3 (e), as we had contracted an edge in the minimum cut. In the new graph, depicted in

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

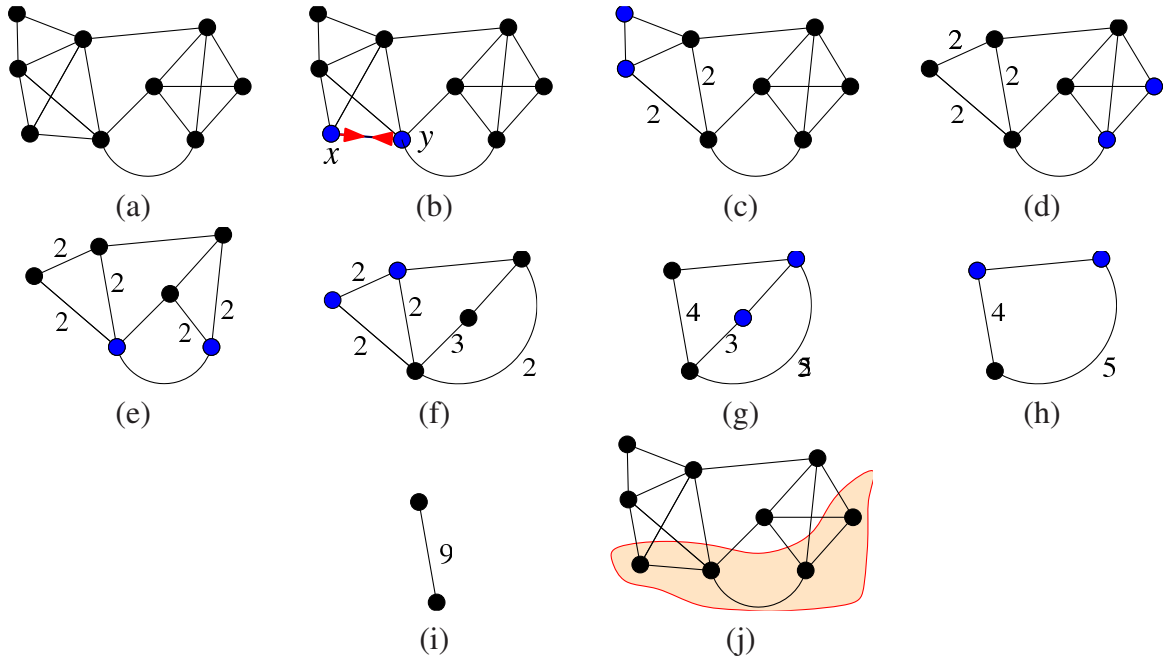


Figure 13.3: (a) Original graph. (b)–(j) a sequence of contractions in the graph, and (h) the cut in the original graph, corresponding to the single edge in (h). Note that the cut of (h) is not a mincut in the original graph.

Figure 13.3 (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.

13.2.1. The resulting algorithm

Observation 13.2.2. *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.

Algorithm MinCut(G)
  $\mathsf{G}_0 \leftarrow G$
  $i = 0$
  while G_i has more than two vertices **do**
    $e_i \leftarrow$ random edge from $\mathsf{E}(\mathsf{G}_i)$
    $\mathsf{G}_{i+1} \leftarrow G_i/e_i$
    $i \leftarrow i + 1$
  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 13.4: The minimum cut algorithm.

Using Observation 13.2.2 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 13.4 for the resulting algorithm **MinCut**.

13.2.1.1. On the art of randomly picking an edge

Every edge has a weight associated with it (which is the number of edges in the original graph it represents). A vertex weight is the total weight associated with it. We maintain during the contraction for every vertex the total weight of the edges adjacent to it. We need the following easy technical lemma.

Lemma 13.2.3. *Let $X = \{x_1, \ldots, x_n\}$ be a set of n elements, and let $\omega(x_i)$ be an integer positive weight. One can pick randomly, in $O(n)$ time, an element from the set X, with the probability of picking x_i being $\omega(x_i)/W$, where $W = \sum_{i=1}^{n} \omega(x_i)$.*

Proof: Pick randomly a real number r in the range 0 to W. We also precompute the prefix sums $\beta_i = \sum_{k=1}^{i} \omega(x_k) = \beta_{i-1} + \omega(x_i)$, for $i = 1, \ldots, n$, which can be done in linear time. Now, find the first index i, such that $\beta_{i-1} < r \le \beta_i$. Clearly, the probability of x_i to be picked is exactly $\omega(x_i)/W$. ■

Now, we pick a vertex randomly according to the vertices weight in $O(n)$ time, and we pick randomly an edge adjacent to a vertex in $O(n)$ time (again, according to the weights of the edges). Thus, we uniformly sample an edge of the graph, with probability proportional to its weight, as desired.

13.2.2. Analysis

13.2.2.1. 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 13.2.4. *If a graph G has a minimum cut of size k and G has n vertices, then $|\mathsf{E}(\mathsf{G})| \ge kn/2$.*

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 $\sum_{v \in V} \text{degree}(v)/2 \ge nk/2$ edges in the graph. ■

Lemma 13.2.5. *If we pick in random an edge e from a graph G, then with probability at most $2/n$ it belong to the minimum cut.*

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 13.2.6. **MinCut** *outputs the mincut with probability* $\ge \dfrac{2}{n(n-1)}$.

Proof: Let $\mathcal{E}_i$ be the event that e_i is not in the minimum cut of G_i. By Observation 13.2.2, **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 13.2.5, it holds $\mathbf{Pr}\left[\mathcal{E}_i \,\middle|\, \mathcal{E}_0 \cap \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}\right] \ge 1 - \dfrac{2}{|V(G_i)|} = 1 - \dfrac{2}{n-i}$. Implying that

$$\begin{aligned}
\Delta &= \mathbf{Pr}[\mathcal{E}_0 \cap \ldots \cap \mathcal{E}_{n-3}] \\
&= \mathbf{Pr}[\mathcal{E}_0] \cdot \mathbf{Pr}\left[\mathcal{E}_1 \,\middle|\, \mathcal{E}_0\right] \cdot \mathbf{Pr}\left[\mathcal{E}_2 \,\middle|\, \mathcal{E}_0 \cap \mathcal{E}_1\right] \cdot \ldots \cdot \mathbf{Pr}\left[\mathcal{E}_{n-3} \,\middle|\, \mathcal{E}_0 \cap \ldots \cap \mathcal{E}_{n-4}\right]
\end{aligned}$$

As such, we have

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

■

13.2.2.2. Running time analysis.

Observation 13.2.7. **MinCut** *runs in* $O(n^2)$ *time.*

Observation 13.2.8. *The algorithm always outputs a cut, and the cut is not smaller than the minimum cut.*

Definition 13.2.9. (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 13.2.10. *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 13.2.6. 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 13.2.11. *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.*

13.3. 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

$$\begin{aligned}\mathbf{Pr}\left[\mathcal{E}_0 \cap \ldots \cap \mathcal{E}_{\nu-1}\right] &\geq \prod_{i=0}^{\nu-1}\left(1-\frac{2}{n-i}\right) = \prod_{i=0}^{\nu-1}\frac{n-i-2}{n-i} \\ &= \frac{n-2}{n} * \frac{n-3}{n-1} * \frac{n-4}{n-2}\cdots = \frac{(n-\nu)(n-\nu-1)}{n\cdot(n-1)}. \qquad (13.2)\end{aligned}$$

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

```
Contract( G, t )
  while |(G)| > t do
      Pick a random edge e in G.
      G ← G/e
  return G
```

```
FastCut(G = (V, E))
  G – multi-graph
begin
  n ← |V(G)|
  if n ≤ 6 then
      Compute (via brute force) minimum cut
      of G and return cut.
  t ← ⌈1 + n/√2⌉
  H_1 ← Contract(G, t)
  H_2 ← Contract(G, t)
  /* Contract is randomized!!! */
  X_1 ← FastCut(H_1),
  X_2 ← FastCut(H_2)
  return minimum cut out of X_1 and X_2.
end
```

Figure 13.5: **Contract**(G, t) shrinks G till it has only t vertices. **FastCut** computes the minimum cut using **Contract**.

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 13.5, which also depict the new algorithm **FastCut**.

Lemma 13.3.1. *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:

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

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

Exercise 13.3.2. Show that one can modify **FastCut** so that it uses only $O(n^2)$ space.

Lemma 13.3.3. *The probability that* **Contract**$(G, n/\sqrt{2})$ *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. (13.2). We have

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

■

The following lemma bounds the probability of success. A more elegant argument is given in Section 13.3.1 below.

Lemma 13.3.4. **FastCut** *finds the minimum cut with probability larger than* $\Omega(1/\log n)$.

Proof: Do not read this proof – a considerably more elegant argument is given in Section 13.3.1.

Let $P(n)$ be the probability that the algorithm succeeds on a graph with n vertices.

The probability to succeed in the first call on H_1 is the probability that **Contract** did not hit the minimum cut (this probability is larger than $1/2$ by Lemma 13.3.3), times the probability that the algorithm succeeded on H_1 in the recursive call (those two events are independent). Thus, the probability to succeed on the call on H_1 is at least $(1/2) * P(n/\sqrt{2})$, Thus, the probability to fail on H_1 is $\leq 1 - \frac{1}{2}P\left(\frac{n}{\sqrt{2}}\right)$.

The probability to fail on both H_1 and H_2 is smaller than

$$\left(1 - \frac{1}{2}P\left(\frac{n}{\sqrt{2}}\right)\right)^2,$$

since H_1 and H_2 are being computed independently. Note that if the algorithm, say, fails on H_1 but succeeds on H_2 then it succeeds to return the mincut. Thus the above expression bounds the probability of failure. And thus, the probability for the algorithm to succeed is

$$P(n) \geq 1 - \left(1 - \frac{1}{2}P\left(\frac{n}{\sqrt{2}}\right)\right)^2 = P\left(\frac{n}{\sqrt{2}}\right) - \frac{1}{4}\left(P\left(\frac{n}{\sqrt{2}}\right)\right)^2.$$

We need to solve this recurrence. (This is very tedious, but since the details are non-trivial we provide the details of how to do so.) Divide both sides of the equation by $P\left(n/\sqrt{2}\right)$ we have:

$$\frac{P(n)}{P(n/\sqrt{2})} \geq 1 - \frac{1}{4}P(n/\sqrt{2}).$$

It is now easy to verify that this inequality holds for $P(n) \geq c/\log n$ (since the worst case is $P(n) = c/\log n$ we verify this inequality for this value). Indeed,

$$\frac{c/\log n}{c/\log(n/\sqrt{2})} \geq 1 - \frac{c}{4\log(n/\sqrt{2})}.$$

As such, letting $\Delta = \log n$, we have

$$\frac{\log n - \log\sqrt{2}}{\log n} = \frac{\Delta - \log\sqrt{2}}{\Delta} \geq \frac{4(\log n - \log\sqrt{2}) - c}{4(\log n - \log\sqrt{2})} = \frac{4(\Delta - \log\sqrt{2}) - c}{4(\Delta - \log\sqrt{2})}.$$

Equivalently, $4(\Delta - \log\sqrt{2})^2 \geq 4\Delta(\Delta - \log\sqrt{2}) - c\Delta$. Which implies $-8\Delta\log\sqrt{2} + 4\log^2\sqrt{2} \geq -4\Delta\log\sqrt{2} - c\Delta$. Namely,

$$c\Delta - 4\Delta\log\sqrt{2} + 4\log^2\sqrt{2} \geq 0,$$

which clearly holds for $c \geq 4\log\sqrt{2}$.

We conclude, that the algorithm succeeds in finding the minimum cut in probability

$$\geq 2\log 2/\log n.$$

(Note that the base of the induction holds because we use brute force, and then $P(i) = 1$ for small i.) ∎

Exercise 13.3.5. 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 13.3.6. *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 13.3.1. The bound on the probability follows from Lemma 13.3.4, and using the amplification analysis as done in Lemma 13.2.10 for **MinCutRep**. ∎

13.3.1. On coloring trees and min-cut

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. 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 $\rho_h = \mathbf{Pr}[\mathcal{E}_h]$. 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 $\mathbf{Pr}\Big[\text{the edge } uu_l \text{ is colored black}\Big] \cdot \rho_{h-1} = \rho_{h-1}/2$. 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

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

Lemma 13.3.7. *We have that* $\rho_h \geq 1/(h+1)$.

Proof: 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. ∎

The recursion tree for **FastCut** corresponds to such a coloring. Indeed, it is a binary tree as every call performs two recursive calls. Inside such a call, we independently perform two (independent) contractions reducing the given graph with n vertices to have $n/\sqrt{2}$ vertices. If this contraction succeeded (i.e., it did not hit the min-cut), then consider this edge to be colored by black (and white otherwise). Clearly, the algorithm succeeds, if and only if, there is black colored path from the root of the tree to the leaf. Since the tree has depth $H \leq 2 + \log_{\sqrt{2}} n$, and by Lemma 13.3.7, we have that the probability of **FastCut** to succeed is at least $1/(h+1) \geq 1/(3 + \log_{\sqrt{2}} n)$.

Galton-Watson processes. Imagine that you start with a single node. The node is going to have two children, and each child survives with probability half (independently). If a child survives it is going to have two children, and so on. Clearly, a single node give a rise to a random tree. The natural question is what is the probability that the original node has descendants h generations in the future. In the above we proved that this probability is at least $1/(h+1)$. See below for more details on this interpretation.

13.4. 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 13.3.1). It is inspired by ***Galton-Watson processes*** (which is a special case of a branching process). The problem that initiated the study of these processes goes back to the 19th century [WG75]. 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 $\mathbf{E}[X] \leq 1$, and it has a constant probability of surviving if $\mathbf{E}[X] > 1$. In our case, X was the number of the two children of a node that their edges were colored black.

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., Koreans 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).

Bibliography

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

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