Chapter 11

Min Cut

By Sariel Har-Peled, December 7, 2009[①]

> **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.

11.1 Min Cut

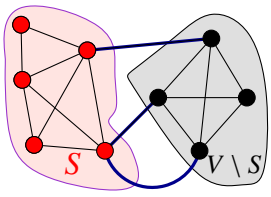
11.1.1 Problem Definition

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

Definition 11.1.1 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.

11.1.2 Some Definitions

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

$$\mathbf{Pr}[(X = x) \cap (Y = y)] = \mathbf{Pr}\Big[X = x \;\Big|\; Y = y\Big] \cdot \mathbf{Pr}[Y = y]. \tag{11.1}$$

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

The following is easy to prove by induction using Eq. (11.1).

Lemma 11.1.2 *Let $\mathcal{E}_1, \ldots, \mathcal{E}_n$ be n events which are not necessarily independent. Then,*

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

11.2 The Algorithm

The basic operation used by the algorithm is ***edge contraction***, depicted in Figure 11.1. 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 11.2.

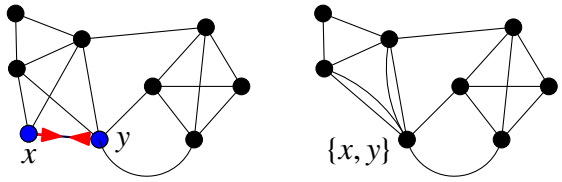


Figure 11.1: (a) A contraction of the edge xy. (b) The resulting graph.

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).

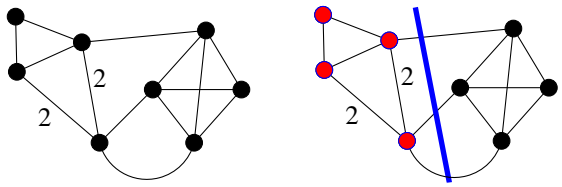


Figure 11.2: (a) A multi-graph. (b) A minimum cut in the resulting multi-graph.

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).

Observation 11.2.1 *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.*

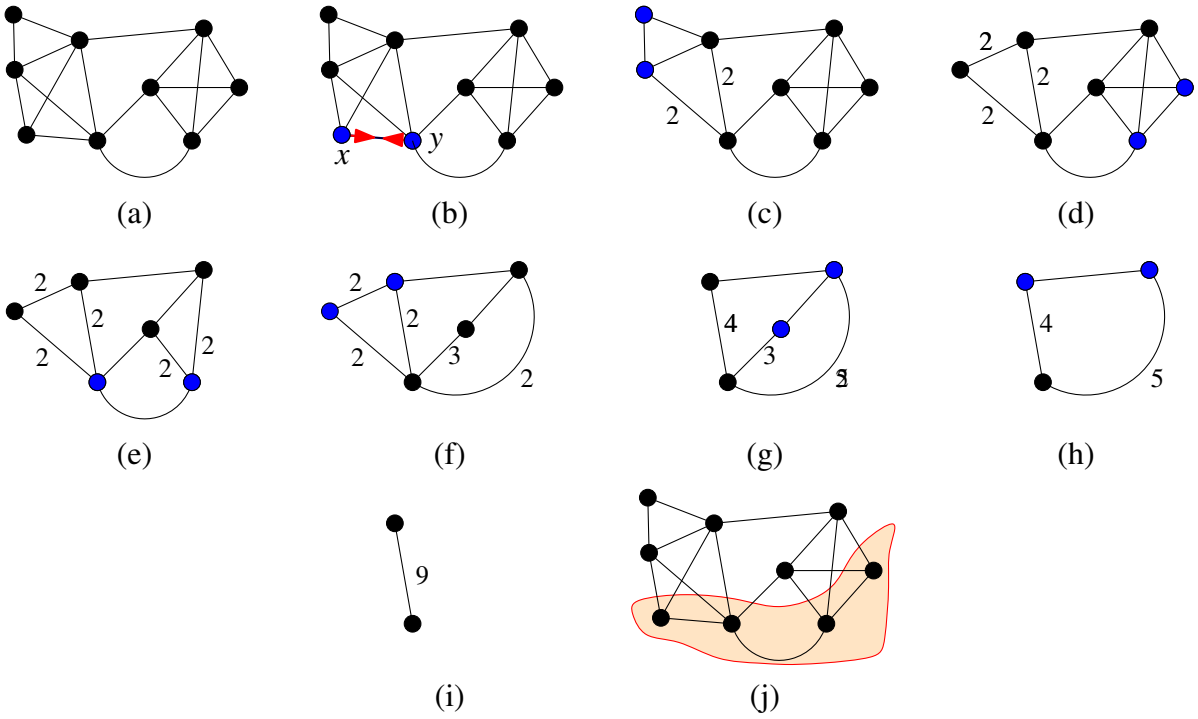


Figure 11.3: (a) Original graph. (b)–(j) a sequence of contractions in the graph, and (h) the cut in the original graph, corresponding to the single edge in (h). Note that the cut of (h) is not a mincut in the original graph.

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 11.3, 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 11.3 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 11.3 (e), as we had contracted an edge in the minimum cut. In the new graph, depicted in Figure 11.3 (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.

Observation 11.2.2 *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.*

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

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.

Using Observation 11.2.2 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 11.4 for the resulting algorithm **MinCut**.

Algorithm **MinCut**(G)
  $\mathsf{G}_0 \leftarrow G$
  $i = 0$
  while G_i has more than two vertices **do**
    Pick randomly an edge e_i from the edges of G_i
    $\mathsf{G}_{i+1} \leftarrow G_i / e_i$
    $i \leftarrow i + 1$
  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 11.4: The minimum cut algorithm.

11.2.1 Analysis

11.2.1.1 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 11.2.3 *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 $\sum_{v \in V} \text{degree}(v)/2 \geq nk/2$ edges in the graph. ■

Lemma 11.2.4 *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 11.2.5 **MinCut** *outputs the mincut with probability* $\geq \dfrac{2}{n(n-1)}$.

Proof: Let $\mathcal{E}_i$ be the event that e_i is not in the minimum cut of G_i. By Observation 11.2.2, **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 11.2.4, it holds $\mathbf{Pr}[\mathcal{E}_i \,|\, \mathcal{E}_1 \cap \ldots \cap \mathcal{E}_{i-1}] \geq 1 - \dfrac{2}{|V(G_i)|} = 1 - \dfrac{2}{n-i}$. Implying that

$$\begin{aligned} \Delta &= \mathbf{Pr}[\mathcal{E}_0 \cap \ldots \cap \mathcal{E}_{n-2}] \\ &= \mathbf{Pr}[\mathcal{E}_0] \cdot \mathbf{Pr}[\mathcal{E}_1 \,|\, \mathcal{E}_0] \cdot \mathbf{Pr}[\mathcal{E}_2 \,|\, \mathcal{E}_0 \cap \mathcal{E}_1] \cdot \ldots \cdot \mathbf{Pr}[\mathcal{E}_{n-3} \,|\, \mathcal{E}_0 \cap \ldots \cap \mathcal{E}_{n-4}] \end{aligned}$$

As such, we have

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

■

11.2.1.2 Running time analysis.

Observation 11.2.6 **MinCut** *runs in $O(n^2)$ time.*

Observation 11.2.7 *The algorithm always outputs a cut, and the cut is not smaller than the minimum cut.*

Definition 11.2.8 (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 11.2.9 *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 11.2.5. 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 11.2.10 *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.*

11.3 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]?

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

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

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

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

$$\begin{aligned}\mathbf{Pr}[\mathcal{E}_0 \cap \ldots \cap \mathcal{E}_{\nu-1}] &\geq \prod_{i=0}^{\nu-1}\left(1 - \frac{2}{n-i}\right) = \prod_{i=0}^{\nu-1}\frac{n-i-2}{n-i} \\ &= \frac{n-2}{n} * \frac{n-3}{n-1} * \frac{n-4}{n-2} \cdots = \frac{(n-\nu)(n-\nu-1)}{n\cdot(n-1)}. \end{aligned} \tag{11.2}$$

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

Lemma 11.3.1 *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 11.3.2 Show that one can modify **FastCut** so that it uses only $O(n^2)$ space.

Lemma 11.3.3 *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. (11.2). 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.

Lemma 11.3.4 **FastCut** *finds the minimum cut with probability larger than* $\Omega(1/\log n)$.

Proof: 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 11.3.3), 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 11.3.5 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 11.3.6 *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 11.3.1. The bound on the probability follows from Lemma 11.3.4, and using the amplification analysis as done in Lemma 11.2.9 for **MinCutRep**. ■

11.4 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].

Bibliography

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