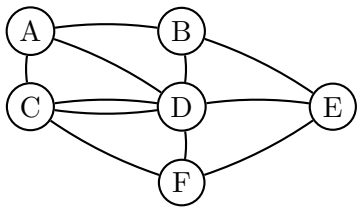
Chapter 2

A Randomized Algorithm for Minimum Cuts

A *randomized algorithm* is an algorithm that receives, in addition to its input, a stream of random bits which is used to make random choices. The random bits are assumed to be independent of the input. A salient feature is that repeated runs of a randomized algorithm with fixed input data will, in general, not produce the same result. You might be perplexed that such a lack of definiteness is desirable, but consider that this feature allows to transform deterministic algorithms with bad worst case behaviour into randomized algorithms that perform well with high probability *on any input*. I hope that the next example can convey that randomized algorithms are often *simple* and *efficient*.

§1 A Minimum Cut Algorithm

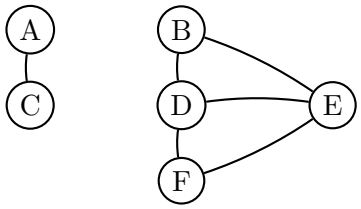
Let $G = (V, E)$ be a connected, undirected, loopfree multigraph with n vertices. A multigraph is like a graph but may contain multiple edges between two vertices, as the following example shows.



[0]© 2009 by Andreas Klappenecker. All rights reserved.

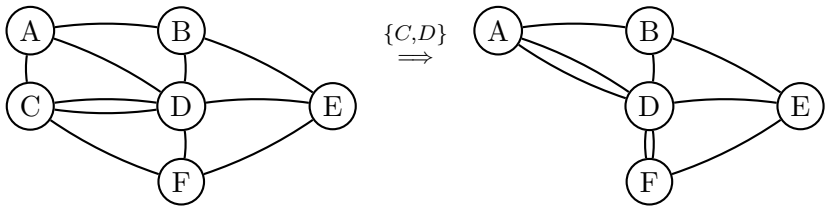
A *cut* in the multigraph $G = (V, E)$ is a partition of the vertex set V into two disjoint nonempty sets $V = V_1 \cup V_2$. An edge with one end in V_1 and the other in V_2 is said to *cross the cut*. The cut is often identified with the multiset of crossing edges.

The term *cut* is chosen because the removal of the edges in a cut partitions the multigraph. For example, if we partition $V = \{A, B, C, D, E, F\}$ into the sets $V_1 = \{A, C\}$ and $V_2 = \{B, D, E, F\}$ in the previous example, then this cut has five crossing edges, and removing these edges yields the disconnected multigraph:



The *size* of the cut is given by the number of edges crossing the cut. Our goal is to determine the minimum size of a cut in a given multigraph G.

We describe a very simple randomized algorithm for this purpose. If e is an edge of a loopfree multigraph G, then the multigraph G/e is obtained from G by contracting the edge $e = \{x, y\}$, that is, we identify the vertices x and y and remove all resulting loops.



The above figure shows a multigraph G and the multigraph $G/\{C, D\}$ resulting from contracting an edge between C and D. We keep the label of one vertex to avoid cluttered notations, but keep in mind that a node D in the graph $G/\{C, D\}$ really represents the set of all nodes that are identified with D.

Note that any cut of G/e induces a cut of G. For instance, in the above example the cut $\{A, B\} \cup \{D, E, F\}$ in $G/\{C, D\}$ induces the cut $\{A, B\} \cup \{C, D, E, F\}$ in G. In general, the vertices that have been identified in G/e are in the same partition of G.

The size of the minimum cut of G/e is at least the size of the minimum cut of G, because all edges are kept. Thus we can use successive contractions to estimate the size of the minimum cut of G. This is the basic idea of the following randomized algorithm.

Contract(G)
Input: A connected loopfree multigraph $G = (V, E)$ with at least 2 vertices.
1: **while** $|V| > 2$ **do**
2: **Select** $e \in E$ **uniformly at random**;
3: G := G/e;
4: **od**;
5: **return** $|E|$.
Output: An upper bound on the minimum cut of G.

The algorithms Contract selects uniformly at random one of the remaining edges and contracts this edge until two vertices remain. The cut determined by this algorithm contains precisely the edges that have not been contracted. Counting the edges between the remaining two vertices yields an estimate of the size of the minimum cut of G.

The algorithm is best understood by an example. Figure 2.1 shows two different runs of the algorithm Contract. Let us have a closer look at the run shown in the left column of this figure. The multigraph provided as an input is depicted in the top left. First the edge $\{D, E\}$ is contracted. The resulting graph is shown directly below. The edges $\{D, F\}$, $\{C, D\}$, and $\{B, D\}$ are respectively contracted in the remaining steps.

Each contraction identifies two vertices. The remaining two nodes A and B in the final multigraph in the lower left represent the sets $\{A\}$ and $\{B, C, D, E, F\}$, since the contractions produced the identifications

$$E \sim D \sim F \sim C \sim B,$$

respectively. Therefore, the cut $\{A\} \cup \{B\}$ in the final multigraph corresponds to the cut $\{A\} \cup \{B, C, D, E, F\}$ in the input multigraph.

Exercise 2.1 *Describe the cut in the input graph that is induced by the cut $\{C\} \cup \{D\}$ in the final multigraph in the right column of Figure 2.1. Assume that $\{D, F\}$ was the last contracted edge. If there is some ambiguity, then list all possibilities.*

The examples amply demonstrate some unsettling property of the algorithm Contract: The algorithm does not always produce the correct size of

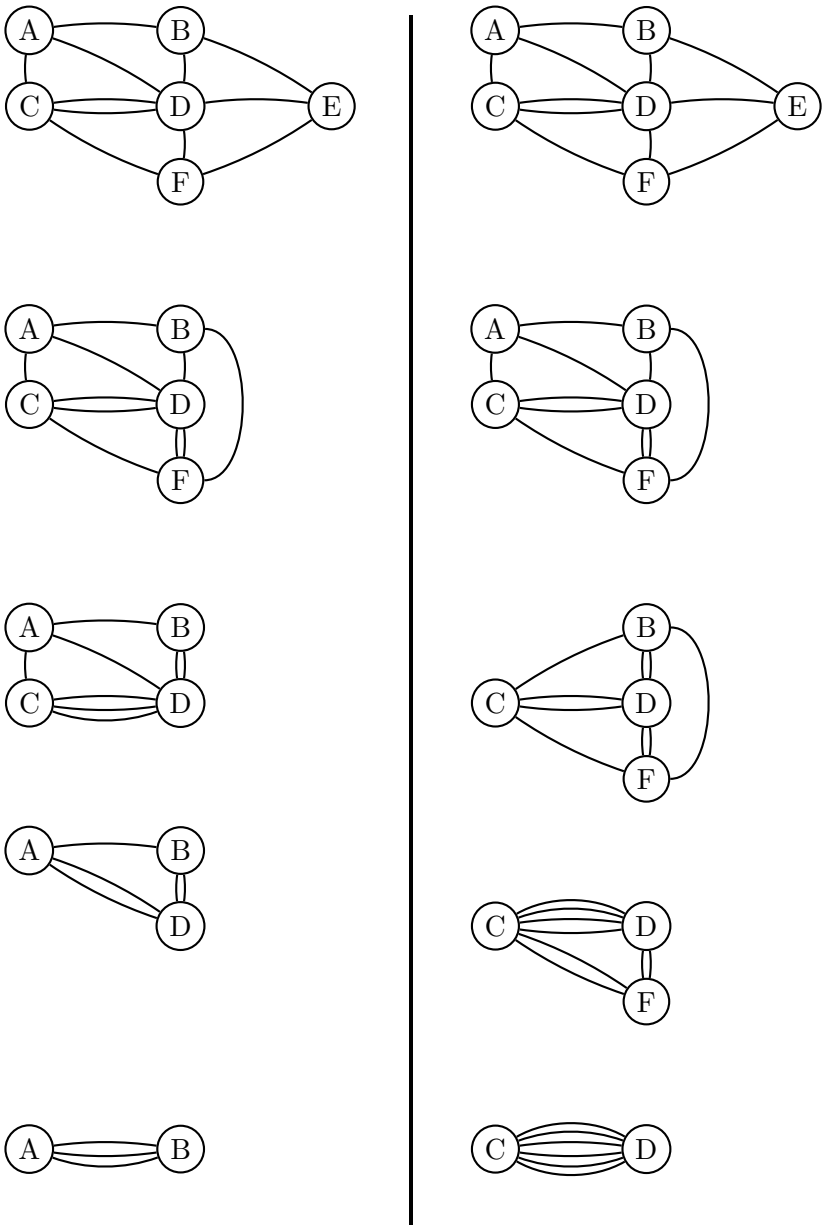


Fig. 2.1 Two different runs of the Contract algorithm. The algorithm does not always produce the correct result. The run shown in the left column correctly determines the minimum cut size to be 3. The run shown in the right column fails to produce the correct result; here the algorithm will claim that the size of the minimum cut is 6.

the minimum cut. As we will see in the next section, it is not difficult to show that the correct size of a minimum cut will be found by the algorithm Contract with probability $\Omega(n^2)$, where n denotes the number of vertices of the multigraph. Repeating the algorithm $O(n^2 \log n)$ times and choosing the smallest value returned by the runs yields the correct size of the minimum cut with high probability.

The beauty of this scheme is its simplicity. If G is represented as a labeled graph, where the labels denote the multiplicity of the edges, then Contract can be implemented with $O(n^2)$ operations; running the algorithm repeatedly, as suggested before, yields at total of $O(n^4 \log n)$ operations.

Remark. The running time of the best deterministic minimum cut algorithm is $O(nm + n^2 \log n)$, where m denotes the number of edges, that is, in the labeled graph representation the running time is at most $O(n^3)$, see for instance [2]. It turns out that size of the minimum cut can be determined with high probability in $O(n^2 \log^3 n)$ steps using a refined version of the contraction algorithm, see [1].

§2 Analysis

We now want to analyze the algorithm given in the previous section. Our goal is to determine a lower bound on the probability that the algorithm correctly determines the minimum cut. We will see that this algorithm produces the correct answer with probability $\Omega(1/n^2)$. We need remarkably few tools from probability theory in this proof: All we need is the innocuous formula

$$\boxed{\Pr[E \cap F] = \Pr[E|F] \Pr[F]}$$

Exercise 2.2 *Prove the following straightforward consequence of the previous formula*

$$\Pr[\cap_{\ell=1}^{n} E_\ell] = \left(\prod_{m=2}^{n} \Pr[E_m| \cap_{\ell=1}^{m-1} E_\ell] \right) \Pr[E_1].$$

If you expand the formula then you will immediately see the pattern.

Let me motivate the approach taken in the analysis by emphasizing a special case. Suppose that the multigraph has a uniquely determined minimum cut. If the algorithm selects in this case *any* edge crossing this cut, then the algorithm will fail to produce the correct result. The analysis is largely guided by this observation.

Exercise 2.3 *Give an example of a connected, loopfree multigraph with at least four vertices that has a uniquely determined minimum cut.*

Let $G = (V, E)$ be a loopfree connected multigraph with $n = |V|$ vertices. Note that each contraction reduces the number of vertices by one, so the algorithm terminates after $n - 2$ steps.

Suppose that C is a particular minimum cut of G. Let E_i denote the event that the algorithm selects in the ith step an edge that does not cross the cut C. Therefore, the probability that no edge crossing the cut C is ever picked during an execution of the algorithm is $\Pr[\cap_{j=1}^{n-2} E_j]$. By Exercise 2.2, this probability can be calculated by

$$\Pr[\cap_{m=1}^{n-2} E_m] = \left(\prod_{m=2}^{n-2} \Pr[E_m | \cap_{\ell=1}^{m-1} E_\ell] \right) \Pr[E_1]. \tag{2.1}$$

Suppose that the size of the minimum cut is k. This means that the degree of each vertex is at least k, hence there exist at least $kn/2$ edges. The probability to select an edge crossing the cut C in the first step is at most $k/(kn/2) = 2/n$. Consequently, $\Pr[E_1] \geq 1 - 2/n = (n-2)/n$.

Similarly, at the beginning of the mth step, with $m \geq 2$, there are $n-m+1$ remaining vertices. The minimum cut is still at least k, hence the multigraph has at this stage at least $k(n-m+1)/2$ edges. Assuming that none of the edges crossing C was selected in an earlier step, the probability to select an edge crossing the cut C is $2/(n-m+1)$. It follows that

$$\Pr\left[E_m | \bigcap_{j=1}^{m-1} E_j\right] \geq 1 - \frac{2}{n-m+1} = \frac{n-m-1}{n-m+1}.$$

Applying these lower bounds to the terms in equation (2.1) yields the result:

$$\Pr\left[\bigcap_{j=1}^{n-2} E_j\right] \geq \prod_{m=1}^{n-2} \left(\frac{n-m-1}{n-m+1}\right) = \frac{2}{n(n-1)}.$$

The last equality is obtained by canceling terms in the telescoping product.

In conclusion, we have shown that the contraction algorithm yields the correct answer with probability at least $\Omega(1/n^2)$.

Repetitions. We can repeatedly execute the randomized algorithm Contract and take the minimum of all results. Recall from calculus that

$$\left(1 + \frac{x}{n}\right)^n \leq e^x,$$

and, in fact, $\lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n = e^x$. The probability that the algorithm fails to produce the correct result in one execution is $\Pr[\text{failure}] = (1-2/n^2)$. Recall that for independent event E and F, the probability is given by $\Pr[E \cap F] = \Pr[E]\Pr[F]$. Therefore, if we execute the algorithm $n^2/2$ times, then the probability that the repeated executions will never reveal the correct size of the minimum cut is given by $(1-2/n^2)^{n^2/2} \leq e^{-1}$. We can conclude that repeating the contraction algorithm $O(n^2 \log n)$ times yields the correct size of the minimum cut with high probability.

Bibliography

[1] D.R. Karger and C. Stein. A new approach to the min-cut problem. *J. ACM*, 43(4):601–640, 1996.

[2] M. Stoer and F. Wagner. A simple min-cut algorithm. *J. ACM*, 44(4):585–591, 1997.