Randomized Algorithms

Andreas Klappenecker *Texas A&M University*

Lecture notes of the course Randomized Algorithms, Fall 2003. Preliminary draft.

© 2003 by Andreas Klappenecker. All rights reserved.

Chapter 1

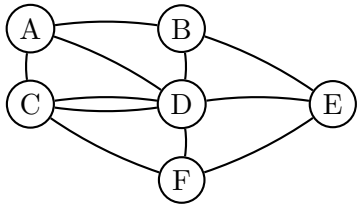
Algorithmic Appetizers

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*.

We give a short exposition of selected randomized algorithms to provide you with an impression of the flavor of the subject. I hope that these examples can convey that randomized algorithms are often *simple* and *efficient*.

§1 A Minimum Cut Algorithm

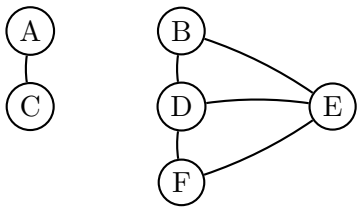
Our first example is a graph-theoretic algorithm. Let $G = (V, E)$ be a connected, undirected, loopfree multigraph with n vertices. A multigraph may contain multiple edges between two vertices, as the following example shows.



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

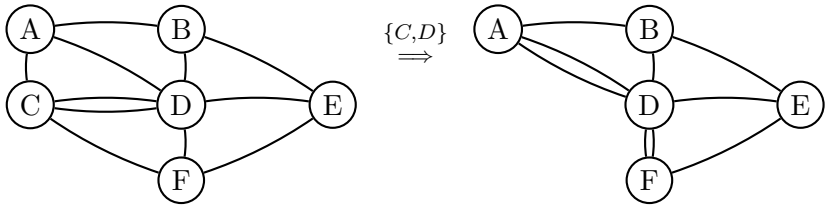
in V_2 is said to *cross the cut*. The cut is often identified with the multiset of crossing edges.

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 1.1 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 1.1 *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 1.1. 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 the minimum cut. 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

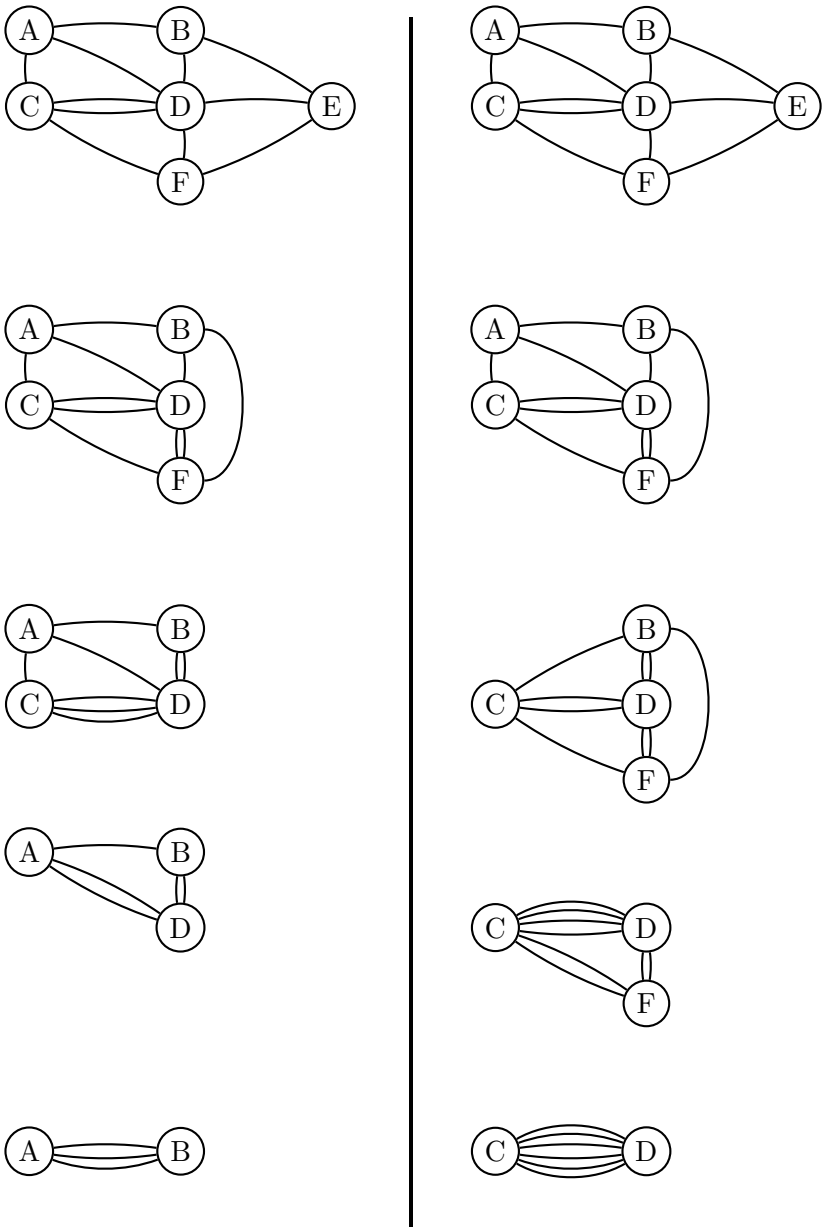


Fig. 1.1 Two different runs of the Contract algorithm. The algorithm does not always produce the correct result. The run shown in the left column correctly determines the minimum cut size to be 3. The run shown in the right column fails to produce the correct result; here the algorithm will claim that the size of the minimum cut is 6.

the correct size of the minimum cut with high probability. (We will explain these facts in detail after reviewing the basics of probability theory).

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 String Comparison

Suppose that a company has one division at the East Coast and another one at the West Coast. The database of the company is mirrored in both divisions, meaning that each update is performed at both divisions. The company relies–tragically–on products of the software giant Millisoft, which is capable to produce more software bugs per minute than any competitor.

The system administrators seek to detect inconsistencies between the two mirrors as early as possible. They decide to run consistency checks between the two databases every evening. It is not feasible to communicate the whole database from coast to coast for such testing purposes. Instead, the system administrators decide to use fingerprinting techniques.

The content of each database can be interpreted as a long string of bits. We can interpret such a string as an integer. Suppose that the content of the database on the East Coast is represented by integer x, and the one on the West Coast by y. We can choose a prime p and calculate $x_p := x \bmod p$. Assuming that p is not too large, we can send the residue x_p to the West Coast division. This division calculates $y := y \bmod p$ and compares the two residues. If the two residues differ, then we clearly have a database inconsistency.

A deterministic choice of the prime p is not advisable, because there exist many distinct integers x and y such that $x - y$ is a multiple of p, such that $x \equiv y \bmod p$, but $x \neq y$.

A randomized choice of the prime p avoids this problem. So, rather than agreeing on a prime in advance, we select a new prime in each run of the algorithm.

Algorithm E (Equality)

E1. Select a prime $p < M$ uniformly at random.

E2. Compute $x_p := x \bmod p$ and send (x_p, p) to the West Coast.

E3. The West Coast division computes $y_p := y \bmod p$.

E4. If $x_p \neq y_p$ then announce *inconsistent* otherwise *consistent*.

Note that the algorithm might fail, for the same reason why we rejected the deterministic version. We will now show that if the prime p is chosen from a suitably large range $2 \leq p < M$, then the probability that the algorithm fails will be very small for any two strings x and y.

Let $\pi(N)$ denote the number of primes less than N. It is a well-known fact from number theory that asymptotically $\pi(N) \sim N/\ln N$. Moreover, if $A < 2^n$, then the number of primes dividing A is less than $\pi(n)$. Equipped with these two facts from number theory, we can calculate the probability that Algorithm E fails to produce the correct answer.

When does Algorithm E fail? The failure occurs if $x \neq y$ but $x \equiv y \bmod p$, meaning that p divides the number $D = |x - y|$. Suppose that x and y have n bits, then the number D is less than 2^n. Therefore, the number of primes dividing D is less than $\pi(n)$. Recall that we selected the prime p from a total of $\pi(M)$ primes, hence the probability of failure is

$$\Pr[\text{failure}] = \frac{|\{p \,|\, p < M, p \text{ is prime}, p \text{ divides } |x-y|\}}{\pi(M)} \leq \frac{\pi(n)}{\pi(M)}.$$

If the integers x and y are $n = 10^{12}$ bits long, and $M = 2^{64}$, then the above formula bounds the probability of failure by $\pi(n)/\pi(M) \approx 8.71 \times 10^{-8}$. If the system administrators are paranoid, then they can run this test several times to further reduce the probability of failure.

In summary, Algorithm E is a simple algorithm that gets the job done. A deterministic algorithm for the same task is problematic, and usually inferior to this simple randomized algorithm.

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.