really identifying the asymptotically "correct" value of t despite our use of the seemingly weak Union Bound.

13.2 Finding the Global Minimum Cut

Randomization naturally suggested itself in the previous example, since we were assuming a model with many processes that could not directly communicate. We now look at a problem on graphs for which a randomized approach comes as somewhat more of a surprise, since it is a problem for which perfectly reasonable deterministic algorithms exist as well.

The Problem

Given an undirected graph $G = (V, E)$, we define a *cut* of G to be a partition of V into two non-empty sets A and B. Earlier, when we looked at network flows, we worked with the closely related definition of an *s-t cut*: there, given a directed graph $G = (V, E)$ with distinguished source and sink nodes s and t, an *s-t cut* was defined to be a partition of V into sets A and B such that $s \in A$ and $t \in B$. Our definition now is slightly different, since the underlying graph is now undirected and there is no source or sink.

For a cut (A, B) in an undirected graph G, the *size* of (A, B) is the number of edges with one end in A and the other in B. A *global minimum cut* (or "global min-cut" for short) is a cut of minimum size. The term *global* here is meant to connote that any cut of the graph is allowed; there is no source or sink. Thus the global min-cut is a natural "robustness" parameter; it is the smallest number of edges whose deletion disconnects the graph. We first check that network flow techniques are indeed sufficient to find a global min-cut.

(13.4) *There is a polynomial-time algorithm to find a global min-cut in an undirected graph G.*

Proof. We start from the similarity between cuts in undirected graphs and *s-t* cuts in directed graphs, and with the fact that we know how to find the latter optimally.

So given an undirected graph $G = (V, E)$, we need to transform it so that there are directed edges and there is a source and sink. We first replace every undirected edge $e = (u, v) \in E$ with two oppositely oriented directed edges, $e' = (u, v)$ and $e'' = (v, u)$, each of capacity 1. Let G' denote the resulting directed graph.

Now suppose we pick two arbitrary nodes $s, t \in V$, and find the minimum *s-t* cut in G'. It is easy to check that if (A, B) is this minimum cut in G', then (A, B) is also a cut of minimum size in G *among all those that separate s from t*. But we know that the global min-cut in G must separate s from *something*,

since both sides A and B are nonempty, and s belongs to only one of them. So we fix any $s \in V$ and compute the minimum s-t cut in G' for every other node $t \in V - \{s\}$. This is $n - 1$ directed minimum-cut computations, and the best among these will be a global min-cut of G. ■

The algorithm in (13.4) gives the strong impression that finding a global min-cut in an undirected graph is in some sense a *harder* problem than finding a minimum s-t cut in a flow network, as we had to invoke a subroutine for the latter problem $n - 1$ times in our method for solving the former. But it turns out that this is just an illusion. A sequence of increasingly simple algorithms in the late 1980s and early 1990s showed that global min-cuts in undirected graphs could actually be computed just as efficiently as s-t cuts or even more so, and by techniques that didn't require augmenting paths or even a notion of flow. The high point of this line of work came with David Karger's discovery in 1992 of the Contraction Algorithm, a randomized method that is qualitatively simpler than all previous algorithms for global min-cuts. Indeed, it is sufficiently simple that, on a first impression, it is very hard to believe that it actually works.

Designing the Algorithm

Here we describe the Contraction Algorithm in its simplest form. This version, while it runs in polynomial time, is not among the most efficient algorithms for global min-cuts. However, subsequent optimizations to the algorithm have given it a much better running time.

The Contraction Algorithm works with a connected *multigraph* $G = (V, E)$; this is an undirected graph that is allowed to have multiple "parallel" edges between the same pair of nodes. It begins by choosing an edge $e = (u, v)$ of G uniformly at random and *contracting* it, as shown in Figure 13.1. This means we produce a new graph G' in which u and v have been identified into a single new node w; all other nodes keep their identity. Edges that had one end equal to u and the other equal to v are deleted from G'. Each other edge e is preserved in G', but if one of its ends was equal to u or v, then this end is updated to be equal to the new node w. Note that, even if G had at most one edge between any two nodes, G' may end up with parallel edges.

The Contraction Algorithm then continues recursively on G', choosing an edge uniformly at random and contracting it. As these recursive calls proceed, the constituent vertices of G' should be viewed as *supernodes*: Each supernode w corresponds to the subset $S(w) \subseteq V$ that has been "swallowed up" in the contractions that produced w. The algorithm terminates when it reaches a graph G' that has only two supernodes v_1 and v_2 (presumably with a number of parallel edges between them). Each of these super-nodes v_i has a corresponding subset $S(v_i) \subseteq V$ consisting of the nodes that have been

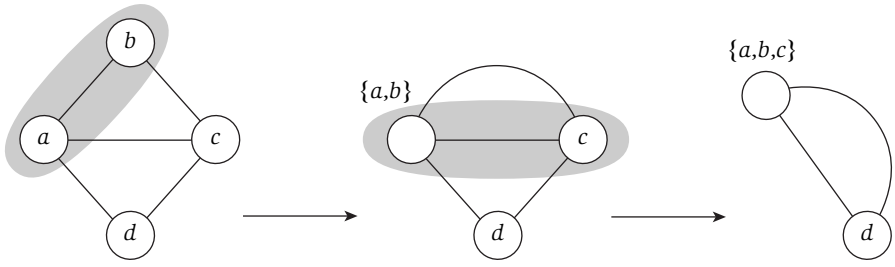

Figure 13.1 The Contraction Algorithm applied to a four-node input graph.

contracted into it, and these two sets $S(v_1)$ and $S(v_2)$ form a partition of V. We output $(S(v_1), S(v_2))$ as the cut found by the algorithm.

```
The Contraction Algorithm applied to a multigraph G = (V, E):
  For each node v, we will record
          the set S(v) of nodes that have been contracted into v
       Initially S(v) = {v} for each v
  If G has two nodes v1 and v2, then return the cut (S(v1), S(v2))
  Else choose an edge e = (u, v) of G uniformly at random
       Let G' be the graph resulting from the contraction of e,
           with a new node z_uv replacing u and v
       Define S(z_uv) = S(u) ∪ S(v)
       Apply the Contraction Algorithm recursively to G'
  Endif
```

Analyzing the Algorithm

The algorithm is making random choices, so there is some probability that it will succeed in finding a global min-cut and some probability that it won't. One might imagine at first that the probability of success is exponentially small. After all, there are exponentially many possible cuts of G; what's favoring the minimum cut in the process? But we'll show first that, in fact, the success probability is only polynomially small. It will then follow that by running the algorithm a polynomial number of times and returning the best cut found in any run, we can actually produce a global min-cut with high probability.

(13.5) *The Contraction Algorithm returns a global min-cut of G with probability at least $1/\binom{n}{2}$.*

Proof. We focus on a global min-cut (A, B) of G and suppose it has size k; in other words, there is a set F of k edges with one end in A and the other

in B. We want to give a lower bound on the probability that the Contraction Algorithm returns the cut (A, B).

Consider what could go wrong in the first step of the Contraction Algorithm: The problem would be if an edge in F were contracted. For then, a node of A and a node of B would get thrown together in the same supernode, and (A, B) could not be returned as the output of the algorithm. Conversely, if an edge not in F is contracted, then there is still a chance that (A, B) could be returned.

So what we want is an upper bound on the probability that an edge in F is contracted, and for this we need a lower bound on the size of E. Notice that if any node v had degree less than k, then the cut $(\{v\}, V - \{v\})$ would have size less than k, contradicting our assumption that (A, B) is a global min-cut. Thus every node in G has degree at least k, and so $|E| \geq \frac{1}{2}kn$. Hence the probability that an edge in F is contracted is at most

$$\frac{k}{\frac{1}{2}kn} = \frac{2}{n}.$$

Now consider the situation after j iterations, when there are $n - j$ supernodes in the current graph G', and suppose that no edge in F has been contracted yet. Every cut of G' is a cut of G, and so there are at least k edges incident to every supernode of G'. Thus G' has at least $\frac{1}{2}k(n-j)$ edges, and so the probability that an edge of F is contracted in the next iteration $j+1$ is at most

$$\frac{k}{\frac{1}{2}k(n-j)} = \frac{2}{n-j}.$$

The cut (A, B) will actually be returned by the algorithm if no edge of F is contracted in any of iterations $1, 2, \ldots, n-2$. If we write $\mathcal{E}_j$ for the event that an edge of F is not contracted in iteration j, then we have shown $\Pr[\mathcal{E}_1] \geq 1 - 2/n$ and $\Pr[\mathcal{E}_{j+1} \mid \mathcal{E}_1 \cap \mathcal{E}_2 \cdots \cap \mathcal{E}_j] \geq 1 - 2/(n-j)$. We are interested in lower-bounding the quantity $\Pr[\mathcal{E}_1 \cap \mathcal{E}_2 \cdots \cap \mathcal{E}_{n-2}]$, and we can check by unwinding the formula for conditional probability that this is equal to

$$\begin{aligned}
&\Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdots \Pr[\mathcal{E}_{j+1} \mid \mathcal{E}_1 \cap \mathcal{E}_2 \cdots \cap \mathcal{E}_j] \cdots \Pr[\mathcal{E}_{n-2} \mid \mathcal{E}_1 \cap \mathcal{E}_2 \cdots \cap \mathcal{E}_{n-3}] \\
&\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right) \cdots \left(1 - \frac{2}{n-j}\right) \cdots \left(1 - \frac{2}{3}\right) \\
&= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right) \cdots \left(\frac{2}{4}\right)\left(\frac{1}{3}\right) \\
&= \frac{2}{n(n-1)} = \binom{n}{2}^{-1}. \quad \blacksquare
\end{aligned}$$

So we now know that a single run of the Contraction Algorithm fails to find a global min-cut with probability at most $(1 - 1/\binom{n}{2})$. This number is very close to 1, of course, but we can amplify our probability of success simply by repeatedly running the algorithm, with independent random choices, and taking the best cut we find. By fact (13.1), if we run the algorithm $\binom{n}{2}$ times, then the probability that we fail to find a global min-cut in any run is at most

$$\left(1 - 1/\binom{n}{2}\right)^{\binom{n}{2}} \leq \frac{1}{e}.$$

And it's easy to drive the failure probability below $1/e$ with further repetitions: If we run the algorithm $\binom{n}{2} \ln n$ times, then the probability we fail to find a global min-cut is at most $e^{-\ln n} = 1/n$.

The overall running time required to get a high probability of success is polynomial in n, since each run of the Contraction Algorithm takes polynomial time, and we run it a polynomial number of times. Its running time will be fairly large compared with the best network flow techniques, since we perform $\Theta(n^2)$ independent runs and each takes at least $\Omega(m)$ time. We have chosen to describe this version of the Contraction Algorithm since it is the simplest and most elegant; it has been shown that some clever optimizations to the way in which multiple runs are performed can improve the running time considerably.

Further Analysis: The Number of Global Minimum Cuts

The analysis of the Contraction Algorithm provides a surprisingly simple answer to the following question: Given an undirected graph $G = (V, E)$ on n nodes, what is the maximum number of global min-cuts it can have (as a function of n)?

For a directed flow network, it's easy to see that the number of minimum s-t cuts can be exponential in n. For example, consider a directed graph with nodes $s, t, v_1, v_2, \ldots, v_n$, and unit-capacity edges (s, v_i) and (v_i, t) for each i. Then s together with any subset of $\{v_1, v_2, \ldots, v_n\}$ will constitute the source side of a minimum cut, and so there are 2^n minimum s-t cuts.

But for global min-cuts in an undirected graph, the situation looks quite different. If one spends some time trying out examples, one finds that the n-node cycle has $\binom{n}{2}$ global min-cuts (obtained by cutting any two edges), and it is not clear how to construct an undirected graph with more.

We now show how the analysis of the Contraction Algorithm settles this question immediately, establishing that the n-node cycle is indeed an extreme case.

(13.6) *An undirected graph $G = (V, E)$ on n nodes has at most $\binom{n}{2}$ global min-cuts.*

Proof. The key is that the proof of (13.5) actually established more than was claimed. Let G be a graph, and let $C_1, \ldots, C_r$ denote all its global min-cuts. Let $\mathcal{E}_i$ denote the event that C_i is returned by the Contraction Algorithm, and let $\mathcal{E} = \cup_{i=1}^{r} \mathcal{E}_i$ denote the event that the algorithm returns any global min-cut.

Then, although (13.5) simply asserts that $\Pr[\mathcal{E}] \geq 1/\binom{n}{2}$, its proof actually shows that for each i, we have $\Pr[\mathcal{E}_i] \geq 1/\binom{n}{2}$. Now each pair of events $\mathcal{E}_i$ and $\mathcal{E}_j$ are disjoint—since only one cut is returned by any given run of the algorithm—so by the Union Bound for disjoint events (13.49), we have

$$\Pr[\mathcal{E}] = \Pr\left[\cup_{i=1}^{r} \mathcal{E}_i\right] = \sum_{i=1}^{r} \Pr[\mathcal{E}_i] \geq r/\binom{n}{2}.$$

But clearly $\Pr[\mathcal{E}] \leq 1$, and so we must have $r \leq \binom{n}{2}$. ∎

13.3 Random Variables and Their Expectations

Thus far our analysis of randomized algorithms and processes has been based on identifying certain "bad events" and bounding their probabilities. This is a qualitative type of analysis, in the sense that the algorithm either succeeds or it doesn't. A more quantitative style of analysis would consider certain parameters associated with the behavior of the algorithm—for example, its running time, or the quality of the solution it produces—and seek to determine the *expected* size of these parameters over the random choices made by the algorithm. In order to make such analysis possible, we need the fundamental notion of a *random variable*.

Given a probability space, a random variable X is a function from the underlying sample space to the natural numbers, such that for each natural number j, the set $X^{-1}(j)$ of all sample points taking the value j is an event. Thus we can write $\Pr[X = j]$ as loose shorthand for $\Pr[X^{-1}(j)]$; it is because we can ask about X's probability of taking a given value that we think of it as a "random variable."

Given a random variable X, we are often interested in determining its *expectation*—the "average value" assumed by X. We define this as

$$E[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j],$$