The Min-Cut Problem

We now return to the min-cut problem considered in Section 1.1. Let $G(V,E)$ be an undirected multigraph with $n$ vertices and $m$ edges. A multigraph is permitted more than one edge between any given pair of vertices. A cut in $G$ is a partition of the vertices $V = (C, \overline{C})$ into two non-empty sets; we refer to this as the cut $C$ with the understanding that $\overline{C}$ is $V \setminus C$.

The value or size of a cut $C$ is the number of edges crossing the cut, i.e., edges with one end-point in each of the two sets $C$ and $\overline{C}$. A multiple edge will contribute its multiplicity to the value of the cut. A min-cut is a cut of minimum value; the min-cut problem is that of finding a min-cut in an input graph $G$.

The value of a min-cut is sometimes referred to as the edge connectivity of the graph, as it is the minimum number of edges that must be removed from the graph to render it disconnected.

We assume that the input graph $G$ is connected, since otherwise the problem is trivially solved by determining the connected components of $G$ in time $O(m)$. The above definitions generalize to weighted graphs, where the value of a cut is defined to be the sum of the weights of the edges crossing the cut. We restrict ourselves to non-negative edge weights. Permitting negative edge weights would make the problem NP-complete since it would then include as a special case the max-cut problem, a classical NP-complete problem.

The min-cut problem should be contrasted with the s-t min-cut problem. In the latter, two distinguished vertices $s$ and $t$ are specified in the input, and the solutions are restricted to the cuts $C$ with the property that $s \in C$ and $t \notin C$.

Exercise 10.8: Show that the min-cut problem for a graph $G$ can be solved via a polynomial number of invocations of an s-t min-cut algorithm applied to the same graph.

The classical duality result in network flows states that the value of a maximum s-t flow in a network equals the value of a s-t min-cut. In fact, computing a maximum s-t flow yields an s-t min-cut as a side-effect. It follows that the min-cut problem can be solved via a polynomial number of invocations of a maximum flow algorithm. Actually, it can be shown that $n-1$ flow computations suffice for this purpose. Since the best deterministic maximum flow algorithm runs in time $O(mn \log(n^2/m))$, this approach to the min-cut problem would require $O(mn^2)$ time. Fortunately, the $n-1$ maximum flow computations needed for the min-cut problem can be implemented in time proportional to the cost of a single maximum flow computation, and so we can compute a min-cut in time $O(mn \log(n^2/m))$.

A very interesting question is whether the s-t min-cut problem can be solved faster than the s-t max-flow problem. Note that whereas a flow computation immediately yields the cut, the converse does not seem to be true. In this section we show that at least for the min-cut problem (without the s-t requirement),
there is an efficient randomized algorithm running in \( O\left(n^2 \log^3 n\right) \) time. For dense graphs this is significantly better than the running time of the best-known max-flow algorithm.

10.2.1. The Contraction Algorithm Revisited

We start by reviewing the the contraction algorithm described in Section 1.1. Actually, we present only an abstract version of this algorithm and leave the implementation details as an exercise.

Given an edge \((x, y)\) in a multigraph \(G(V, E)\), a contraction of the edge \((x, y)\) corresponds to replacing the vertices \(x\) and \(y\) by a new vertex \(z\), and for each \(v \neq \{x, y\}\) replacing any edge \((x, v)\) or \((y, v)\) by the edge \((x, v)\); the rest of the graph remains unchanged. Any multiple edges created are to be retained. The graph obtained by this contraction is denoted by \(G/(x, y)\).

Given a collection of edges \(F \subseteq E\), the effect of contracting the edges in \(F\) is independent of the order of contraction, and the resulting graph is denoted by \(G/F\). The vertex set and edge set of a graph \(G/F\) are denoted by \(V/F\) and \(E/F\). The "meta-vertices" in \(V/F\) correspond to a (connected) set of vertices in \(V\), and the edges in \(E/F\) are exactly those edges in \(E\) whose end-points do not get collapsed into the same meta-vertex in \(V/F\). In Problem 10.9, the reader is asked to show that it is possible to maintain the graph \(G/F\) under an online sequence of edge contractions at a cost of \(O(n)\) time per contraction, keeping track of the correspondence between the elements of \(V/F\) and \(V\), and \(E/F\) and \(E\).

The basic idea behind the contraction algorithm is summarized below. We assume that the Algorithm Contract uses the data structure developed in Problem 10.9 to implement the edge contractions.

```
Algorithm Contract:

Input: A multigraph \(G(V, E)\).
Output: A cut \(C\).
1. \(H \leftarrow G\).
2. while \(H\) has more than 2 vertices do
   2.1. choose an edge \((x, y)\) uniformly at random from the edges in \(H\).
   2.2. \(F \leftarrow F \cup \{(x, y)\}\).
   2.3. \(H \leftarrow H/(x, y)\).
3. \((C, \overline{C}) \leftarrow \) the sets of vertices corresponding to the two meta-vertices in \(H = G/F\).
```

290
10.2 THE MIN-CUT PROBLEM

The only implementation issue remaining in this algorithm is the selection of the edge \((x, y)\) uniformly at random from the set of all edges in the graph \(H\). In Problem 10.10, the reader is asked to show that this can be done in \(O(n)\) time per random selection. The results from Problems 10.9–10.11 yield the following theorem.

**Theorem 10.10:** Algorithm Contract can be implemented to run in \(O(n^2)\) time on any \(n\)-vertex multigraph \(G\).

The running time of this algorithm is independent of the number of (multi) edges in the graph \(G\). This may seem surprising at first since the number of such edges is not bounded by \(\binom{n}{2}\). However, as suggested in Problem 10.9, the multiplicity of an edge can be represented by an integer weight on the edge and hence the number of edges can effectively be bounded by \(\binom{n}{2}\).

Of course, this just shows that the Contract algorithm terminates in \(O(n^2)\) time with a cut \(C\). There is no guarantee that the cut will indeed be a min-cut. We now briefly review the argument from Section 1.1 that established that this algorithm finds a min-cut with a non-negligible probability.

**Lemma 10.11:** A cut \(C\) is produced as output by Algorithm Contract if and only if none of the edges crossing this cut is contracted by the algorithm.

Fix any one min-cut \(K\) in the graph \(G\). Let \(k\) denote the value of a min-cut in \(G\); in particular, \(k\) is the value of the cut \(K\). We would like to compute the probability that \(K\) is produced as the output of Algorithm Contract. By Lemma 10.11, this will happen if and only if none of the \(k\) edges crossing the cut is contracted during the course of the algorithm's execution. To determine the probability of this event, we make use of the following obvious facts.

**Lemma 10.12:** In an \(n\)-vertex multigraph \(G\) with min-cut value \(k\), no vertex has degree smaller than \(k\). Further, the total number of edges in the graph satisfies \(m \geq nk/2\).

**Lemma 10.13:** Given an edge \((x, y)\) in a graph \(G\), the min-cut value in \(G/(x, y)\) is at least as large as the min-cut value in \(G\).

The number of vertices in the graph \(H\) decreases by exactly one during each iteration of Algorithm Contract. After \(n - 2\) iterations the number of vertices is reduced from \(n\) to 2. At the \(i\)th iteration, there are \(n_i = n - i + 1\) vertices in \(H\). Suppose that none of the edges in \(K\) is contracted during the first \(i - 1\) iterations. Since \(K\) is also a cut in \(H\), Lemma 10.13 implies that \(H\) has min-cut value \(k\), and then Lemma 10.12 implies that the number of edges in \(H\) is at least \(nk/2\). Thus, the probability that any edge of \(K\) is contracted during this iteration is at most \(2/n_i\). It follows that the probability that no edge of \(K\) is ever
contracted can be bounded as follows.

\[
\Pr[K \text{ is output by Algorithm Contract}] \geq \prod_{i=1}^{n-2} \left(1 - \frac{2}{n_i}\right) \\
= \prod_{i=1}^{n-2} \left(1 - \frac{2}{n - i + 1}\right) \\
= \prod_{j=1}^{3} \left(\frac{j-2}{j}\right) \\
= 1/\binom{n}{2} = \Omega(n^{-2}).
\]

We have established the following theorem.

**Theorem 10.14:** Any specific min-cut \(K\) is output by Algorithm Contract with probability \(\Omega(n^{-2})\).

Since the graph must have at least one min-cut, it follows that the probability of success of this algorithm is \(\Omega(n^{-2})\). Repeating the algorithm \(O(n^3 \log n)\) times gives a reasonable probability that some invocation of the algorithm produces a min-cut; then, the smallest cut produced by these invocations is very likely to be the min-cut. This gives a Monte Carlo algorithm running in \(O(n^4 \log n)\) time. Before trying to improve this result, we note the following variant of Theorem 10.14.

**Lemma 10.15:** Suppose that the Algorithm Contract is terminated when the number of vertices remaining in the contracted graph is exactly \(t\). Then any specific min-cut \(K\) survives in the resulting contracted graph with probability at least

\[
\binom{t}{2} / \binom{n}{2} = \Omega\left(\left(\frac{t}{n}\right)^2\right).
\]