Random Sampling in Cut, Flow, and Network Design Problems

David R. Karger*
Department of Computer Science
Stanford University
karger@cs.stanford.edu

Abstract

We explore random sampling as a tool for solving undirected graph problems. We show that the sparse graph, or *skeleton*, which arises when we randomly sample a graph's edges will accurately approximate the value of *all* cuts in the original graph. This makes sampling effective for problems involving cuts in graphs.

We apply these tools in fast randomized (Monte Carlo and Las Vegas) algorithms for approximating and exactly finding cuts and flows in an unweighted, undirected graph. We also give weighted-graph versions of these algorithms which use a form of scaling to achieve a sublinear dependence on the maximum edge weight. Our methods also reduce the work done by some parallel cut algorithms.

Our sampling theorems also yield faster algorithms for several other cut based problems, including approximating the best balanced cut of a graph, finding a k-connected orientation of a $2k$-connected graph, and finding integral multicommodity flows in graphs with a great deal of excess capacity.

Our sampling theorems apply even when the sampling probabilities are different for different edges. Using this fact, we show how to use randomized rounding in $(1+o(1))$-approximation algorithm for the $\mathcal{NP}$-complete *minimum k-connected subgraph problem* when $k \gg \log n$; the best previous approximation factor was 2. Our techniques generalize to many other survivable network design problems.

*Supported by a Hertz Foundation Graduate Fellowship, by NSF Grant CCR-9010517, and grants from Mitsubishi and OTL.

1 Introduction

A *cut* in an undirected graph is a partition of the graph's vertices into two nonempty sets. The *value* of the cut is the number, or for a weighted graph the total weight, of edges with one endpoint in each group. Cuts play an important role in determining the solutions to many graph problems. Most obviously, the connectivity of a graph is the value of the minimum graph cut. Similarly, the s-t minimum cut and maximum flow are determined by smallest of all cuts which place s and t on opposite sides. In the $\mathcal{NP}$-complete *network design problem,* the goal is to output a graph which satisfies certain specified connectivity requirements by containing no small cuts. A special case is to find a minimum size (number of edges) k-connected subgraph of a k-connected graph. Other problems to which cuts are relevant include finding a minimum *balanced* cut (in which both sides of the cut are "large") and finding an orientation (assignment of directions) of the edges of an undirected graph which makes it k-connected as a directed graph. Cuts also play an important role in multicommodity flow problems, though the connection is not as tight as for the standard max-flow problem.

We show that random sampling is a powerful tool for cut-dependent undirected graph problems. We use the *graph skeleton* introduced in [15]. Given a graph, a skeleton is constructed on the same set of vertices by including a small random sample from the graph's edges. Our main result is that skeletons accurately approximate all cut values in the original graph. This means random subgraphs can often be used as substitutes for the original graphs in cut and flow problems. Since the subgraphs are small, improved time bounds result.

For example, by computing minimum cuts and maximum flows in the skeleton, we get fast algorithms for approximating s-t minimum cuts and maximum flows. Furthermore, a divide and conquer technique based on [14] allows us to find exact solutions to s-t min-cut and global min-cut problems more quickly than before. For example, we improve the time to find a minimum cut of value c from $\tilde{O}(mc)$ [7] to $\tilde{O}(m\sqrt{c})$ in a graph with minimum cut c. This in turn yields faster algorithms for constructing the cactus representation of minimum cuts in a graph and for augmenting graph connectivity optimally. Our methods also improve the total work done by some parallel cut and flow algorithms.

Our techniques extend to many other problems for which cuts are important. For example, in the minimum *quotient cut* or *balanced cut* problem, the goal is to output a cut of small value but with many vertices on each side. Since in the skeleton all cuts are near their expected values, the minimum quotient cut is also near its expected value, so a balanced cut in the sparse graph is a balanced cut in the original graph. We also address the problem of orienting the edges of an undirected graph so as to maximize the connectivity of the resulting directed graph. We give an extremely simple linear time algorithm which gives an almost optimal answer. We also give a simple algorithm for finding integral multicommodity flows in graphs with a great deal of excess capacity.

From random sampling, it is a small step to show that *randomized rounding* can be effectively applied to graphs with fractional edge weights, yielding integrally weighted graphs with roughly the same cut values. This makes randomized rounding a useful tool in network design problems. We concentrate on the $\mathcal{NP}$-complete *minimum k-connected subgraph problem*, and give a $(1 + o(1))$-approximation algorithm when $k \gg \log n$. Our technique is to find an optimal fractional solution by linear programming and then to use randomized rounding to turn it into an integral one. This technique generalizes to many other network design problems.

All of our techniques apply only to undirected graphs, as cuts in directed graphs do not appear to have the same predictable behavior under random sampling.

1.1 Cuts and Flows

We present algorithms for approximating and for exactly finding s-t and global minimum cuts and maximum flows. To this end, we make the following definition:

Definition 1.1 *An* α-minimal cut *is a cut whose weight is at most α times that of the (global) minimum cut. An α-minimal s-t cut is defined similarly. An α-maximal s-t flow is an s-t flow of value at least α times the optimum.*

Throughout this paper, we will focus attention on an n vertex, m edge graph with min-cut value c. Supposing that the s-t minimum cut has value v, we have randomized Monte Carlo (MC) and Las Vegas (LV) algorithms[1] to find the following objects in unweighted, undirected graphs:

- An s-t maximum flow in $\tilde{O}(mv/\sqrt{c})$ time (LV),
- A global minimum cut in $\tilde{O}(m\sqrt{c})$ time (LV),
- A $(1-\epsilon)$-maximal s-t flow in $\tilde{O}(mv/\epsilon c)$ time (LV),
- A $(1+\epsilon)$-minimal s-t cut in $O(mv/\epsilon^3 c^2)$ time (MC) or $\tilde{O}(mv/\epsilon c)$ time (LV),
- A $(1+\epsilon)$-minimal cut in $\tilde{O}(m + n/\epsilon^3)$ time (MC) or $\tilde{O}(m/\epsilon + n/\epsilon^3)$ time (LV).

Our cut approximation algorithms extend to weighted graphs with roughly the same time bounds. The flow approximation algorithms and exact algorithms can use a "scaling" technique that, for a given maximum edge weight U, increases the time bounds of the flow algorithms by a factor of $\sqrt{U}$ rather than the naive factor of U.

1.2 Network Design

A *network design problem* is specified by an input graph G with each edge assigned a cost. The goal is to output a subgraph of G satisfying certain connectivity requirements at minimum cost (measured as the sum of the costs of edges used). These requirements are described by specifying a minimum number of edges which must cross each cut of G. Since the number of cuts is exponential, the requirements are typically described implicitly, for example by specifying the desired connectivity of the output graph. This formulation easily captures many classic problems, some $\mathcal{NP}$-complete, including the perfect matching, minimum cost flow, Steiner tree, minimum T-join, and minimum k-connected subgraph problems. Recently Williamson et al [29, 11] gave approximation algorithms for a large class of such problems. Their approximation ratio is $O(\log f_{\max})$, where $f_{\max}$ is the maximum demand across any cut. They also note that for a wide range of problems (including all those just mentioned) a *fractional* solution can be found in polynomial time by interior point methods.

We extend the techniques of the graph skeleton to allow edges to be sampled with different probabilities. This lets us apply the *randomized rounding* technique of [27] to the fractional solutions and get good approximation ratios, despite the fact that the rounding must simultaneously satisfy exponentially many constraints. Though our techniques are broadly applicable, due to space limitations we focus on the *minimum k-connected subgraph problem.* Given an input graph, the goal is to find a k-connected subgraph containing the minimum

[1] A *Monte Carlo* algorithm runs in the given time bound but has a small probability of outputting an incorrect answer. A *Las Vegas* algorithm has a small probability of exceeding the specified bound but is guaranteed to output the right answer when it terminates.

possible number of edges. This can modeled as a network design problem in which all edges have unit cost and the demand across every cut is k. We present a randomized rounding algorithm that approximates the optimal (and in fact the optimal fractional) solution to within a $1 + O(\sqrt{(\ln n)/k})$ factor (our algorithm therefore shows that the *integrality gap*, or difference between the factional and integral solutions to the problem, is small). For any $k \gg \ln n$, this improves on the previous best known approximation factor of 2 given by Khuller and Vishkin [18].

1.3 Related Work

Previously, the best time bound for computing maximum flows in unweighted graphs was $O(m \cdot \min(v, n^{2/3}, \sqrt{m}))$, achieved using blocking flows (cf. [28]). In the *unit graphs* that arise in bipartite matching problems, a running time of $O(m\sqrt{n})$ is achieved. Our exact algorithm bounds dominate these whenever the ratio $v/\sqrt{c}$ is small, and in particular when c is large. Our approximation algorithms are even better: for example, we can approximate s-t minimum cuts to within any constant factor in $\tilde{O}(m)$ time so long as $c = \Omega(\sqrt{n})$. We are aware of no previous work on approximating s-t minimum cuts or maximum flows, although blocking flows can be used to achieve a certain large absolute error bound.

For global minimum cuts there are presently two incomparable algorithms. The Contraction Algorithm of Karger and Stein [16] runs in $O(n^2 \log^3 n)$ time while the Round Robin Algorithm of Gabow [7] runs in $O(mc \log(n^2/m))$ time (and also works for directed graphs). Matula [22] gave a deterministic $(2 + \epsilon)$-approximation algorithm for the minimum cut. It runs in $O(m)$ time for any constant ϵ. A skeleton-based randomized $O(m)$-time $(1+\epsilon)$-approximation algorithm for minimum cuts was given in [15]; however it had a worse dependence on ϵ than the one we give here. Furthermore, both the Contraction Algorithm and the approximation algorithm of [15] are Monte Carlo, while our algorithms are Las Vegas.

Skeletons are conceptually related to *sparse graph certificates.* Certificates apply to any monotone increasing property of graphs—one which holds for G if it holds for some subgraph of G. Given such a property, a sparse certificate for G is a sparse subgraph which has the property, proving that G as it as well. The advantage is that since the certificate is sparse, the property can be verified more quickly. The skeleton is a kind of sparse *approximate* certificate. Sparse certificates have already been used successfully in several areas. Eppstein et al [4] give sparsification techniques which improve the running times of dynamic algorithms for numerous graph problems such as connectivity, bipartitioning, and and minimum spanning trees. In [14], we developed a randomized sparsification technique for minimum spanning trees which led to the development of the first linear time minimum spanning tree algorithm by Klein and Tarjan [20].

An important preprocessing subroutine for cut and flow problems is the *sparse k-connectivity certificate* algorithm of Nagamochi and Ibaraki [25, 24]. This algorithm takes an input graph G and parameter k and returns a kn-edge subgraph of G such that all cuts of value at most k in G have the same value in the subgraph. It plays a fundamental role in Matula's algorithm. Gabow uses it to improve his min-cut algorithm's running time to $\tilde{O}(m + nc^{3/2})$. Like Gabow's, all of our cut and flow algorithms can use this preprocessing algorithm. As a result, m can be replaced by cn in all the bounds for our min-cut algorithms and min-cut approximation algorithms (since if we find a $2cn$-connectivity certificate, it will have the same minimum cuts and approximate minimum cuts). Similarly, m can be replaced by nv in all of our s-t cut and flow algorithms. However, it clarifies the presentation to keep m in the time bounds and leave the substitution to the reader.

2 Random Sampling

Our algorithms are all based upon the following model of random sampling in graphs. We are given an unweighted graph G with a *sampling probability* p_e for each edge e, and we construct a random subgraph, or *skeleton,* on the same vertices by placing each edge e in the skeleton independently with probability p_e. Let $\hat{G}$ denote the weighted graph with the vertices and edges of G and with edge weight (capacity) p_e assigned to edge e, and let $\hat{c}$ be the minimum cut of $\hat{G}$. The graph $\hat{G}$ is in some sense the "expected value" of G, since the value of a cut in $\hat{G}$ is the expected value of the same cut in G. The quantity $\hat{c}$ is the minimum expected value of any cut, though not necessarily the expected value of the minimum cut. Our main theorem says that so long as $\hat{c}$ is sufficiently large, every cut in the skeleton takes on roughly its expected value.

Theorem 2.1 (Sampling) *Let $\epsilon = \sqrt{2(d+2)(\ln n)/\hat{c}}$. Then with probability $1 - O(1/n^d)$, every cut in the skeleton of G has value between $(1-\epsilon)$ and $(1+\epsilon)$ times its expected value.*

To prove this theorem, we require two lemmas:

Lemma 2.2 (Karger [13]) *In an undirected graph with minimum cut c, the number of cuts of value less than αc is less than $n^{2\alpha}$.*

Lemma 2.3 (Chernoff [1]) *Let X be a sum of Bernoulli random variables on n trials with success* *probability p and expected number of successes $\mu = np$. Then*

$$\Pr[|X-\mu| > \epsilon\mu] \leq e^{-\epsilon^2\mu/2}.$$

Lemma 2.2 says that the number of cuts within an α factor of the minimum increases exponentially with α. On the other hand, Lemma 2.3 says that the probability one such cut corresponds diverges too far from its expected value decreases exponentially with α. Combining these two lemmas and balancing the exponential rates will prove the theorem:

Proof of Theorem 2.1: Consider any particular cut of value $\alpha\hat{c}$ in $\hat{G}$. Recall that this is the expected value of that cut in the skeleton. By the Chernoff bound, the probability that the cut in the skeleton deviates from this expected value by more than a factor of ϵ is less than $P(\alpha) = e^{-\epsilon^2\alpha\hat{c}/2}$. Now let $f(\alpha)$ be the number of cuts of value $\alpha\hat{c}$ in $\hat{G}$. Then the probability that the lemma fails to hold is just the probability that some cut in the skeleton exceeds its bounds, which is at most

$$\sum_{\alpha \geq 1} f(\alpha)P(\alpha).$$

We will bound this sum by allowing an adversary to select f and showing a bound regardless of his choice. We restrict the adversary according to Lemma 2.2: Defining $F(x) = \sum_{\alpha \leq x} f(x)$, we require that $F(x) \leq n^{2x}$. In fact, we will allow the adversary to choose an arbitrary real valued function, rather than restricting him to integers. This relaxation allows us to transform the sum above into an integral: $f(\alpha)$ becomes $dF/d\alpha$, and the adversary's goal is to maximize

$$\int_1^\infty P(\alpha)\frac{dF}{d\alpha}\,d\alpha$$

under the restriction that $F(x) \leq n^{2x}$.

Suppose the adversary chooses a function F such that for some x, $F(x) < n^{2x}$. Then since $P(\alpha)$ decreases with α, the adversary can increase the value of the integral by increasing $F(x)$. In other words, the adversary will take $F(x) = n^{2x}$ for every x. It then becomes easy to compute the integral; it is just

$$\begin{aligned}\int_1^\infty P(\alpha)\frac{dF}{d\alpha}\,d\alpha &= n^2P(1) + \int_1^\infty e^{-\epsilon^2\alpha\hat{c}/2}n^{2\alpha}\ln n^2\,d\alpha \\ &= 3/n^d\end{aligned}$$

■We also have the following immediate corollary by fixing some probability p and setting $p_e = p$ for every edge. We call the resulting sample a *p-skeleton* of G and denote it $G(p)$.

Corollary 2.4 *Let G be any graph with minimum cut c and let $p = 2(d+2)(\ln n)/\epsilon^2c$. Then the probability that the value of some cut in $G(p)$ has value more than $(1+\epsilon)$ or less than $(1-\epsilon)$ times its expected value is $O(1/n^d)$.*

Proof: Note that the minimum expected cut is pc and apply Theorem 2.1. ■

Given the values p_e, it is clearly trivial to construct a skeleton in $\tilde{O}(m)$ time by flipping an appropriately biased coin for each edge.

3 Min-Cuts and Max-Flows

We now show how the skeleton approach can be applied to minimum cuts and maximum flows. We have the following immediate extension of Corollary 2.4:

Theorem 3.1 *Let G be any graph with minimum cut c and let $p = \Theta((\ln n)/\epsilon^2c)$. Suppose the s-t minimum cut for G has value v. Then with high probability, the s-t minimum cut in $G(p)$ has value between $(1-\epsilon)pv$ and $(1+\epsilon)pv$, and the minimum cut has value between $(1-\epsilon)pc$ and $(1+\epsilon)pc$.*

Our approximation algorithms are based upon constructing a p-skeleton. In these algorithms, given ϵ, we will want to sample with the corresponding $p = \Theta((\ln n)/(\epsilon^2c))$ of Theorem 3.1 in order to ensure that in the skeleton no cut diverges in value by more than ϵ times its expectation. This would appear to require knowledge of c, which is what what we want to compute in the first place. However, we can get around this problem by using Matula's linear-time min-cut approximation algorithm to find a 3-approximation c' upper bounding the min-cut value. If we use this overestimate to determine a corresponding sampling probabililty p', then we know that p' is larger than the correct p, so that ϵ remains an upper bound on the likely deviation in cut values. At the same time, since p' exceeds the correct p by only a constant factor, the expected number of edges in our sample will be of the same order as the number of edges using the correct p. These two properties are sufficient to guarantee the correctness and time bounds of our algorithms. Thus, for simplicity, we will argue as if the correct p for a given ϵ is known to us, so that given ϵ we can construct a corresponding p-skeleton.

We recall the classic *augmenting path* algorithm for maximum flows (cf. [28]). Given an uncapacitated graph and an s-t flow of value f, a linear time depth first search of the residual graph will either show how to augment the flow to one of value $f+1$ or will prove that f is the value of the maximum flow. This algorithm can be used to find a maximum flow of value v in $O(mv)$ time by finding v augmenting paths.

3.1 Approximate Minimum Cuts

The most obvious application of Theorem 3.1 is to approximating minimum cuts. We can find an approximate s-t minimum cut by finding an s-t minimum cut in a skeleton.

Lemma 3.2 *In a graph with minimum cut c, a $(1+\epsilon)$-approximation to the s-t minimum cut of value v can be computed in $\tilde{O}(mv/\epsilon^4c^2)$ time (MC).*

Proof: Given ϵ, determine the corresponding $p = O((\log n)/\epsilon^2 c)$ from Theorem 3.1. Construct a p-skeleton $G(p)$ in $O(m)$ time. Suppose we compute an s-t maximum flow in $G(p)$. By Theorem 3.1, $1/p$ times the value of the computed maximum flow gives a $(1+\epsilon)$-approximation to the s-t min-cut value (with high probability). Furthermore, any flow-saturated cut in $G(p)$ will be a $(1+\epsilon)$-minimal s-t cut in G.

By the Chernoff bound, the skeleton has $O(pm)$ edges with high probability. Also, by Theorem 3.1, the s-t minimum cut in the skeleton has value $O(pv)$. Therefore, the standard path algorithm can find a skeletal s-t maximum flow in $O((pm)(pv)) = \tilde{O}(mv/\epsilon^4c^2)$ time. ■

3.2 Exact Maximum Flows

We next use sampling to speed up the familiar augmenting paths algorithm for maximum flows. Our approach is a randomized divide-and-conquer algorithm which we analyze by treating each subproblem as a (non-independent) random sample. Our technique is based on that of [14], which gives a general approach to solving packing problems. The flow which we are attempting to find can be seen as a packing of disjoint s-t paths. We use the following algorithm:

1. Randomly split the edges of G into two groups (each edge goes to one or the other group with probability 1/2), yielding graphs G_1 and G_2.

2. Recursively compute s-t maximum flows in G_1 and G_2.

3. Add the two flows, yielding an s-t flow f in G.

4. Use augmenting paths (or blocking flows) to increase f to a maximum flow.

This algorithm is clearly correct; the only question is how fast it runs. Suppose the s-t maximum flow is v. Consider G_1. Since each edge of G is in G_1 with probability 1/2, we can apply Theorem 3.1 to deduce that with high probability the s-t maximum flow in G_1 is $v(1-\tilde{O}(1/\sqrt{c}))/2$ and the global minimum cut is $\Theta(c/2)$. The same holds for G_2 (even though the two graphs are not independent). It follows that the flow f has value $v(1-\tilde{O}(1/\sqrt{c})) = v - \tilde{O}(v/\sqrt{c})$. Therefore the number of augmentations which must be performed in G to make f a maximum flow is $\tilde{O}(v/\sqrt{c})$. By deleting isolated vertices as they arise, we can ensure that $m \geq n$ on every problem instance. Thus each augmentation takes $O(m')$ time on an m'-edge graph, and we have the following sort of recurrence for the running time of the algorithm in terms of m, v, and c:

$$T(m, v, c) = 2T(m/2, v/2, c/2) + \tilde{O}(mv/\sqrt{c}).$$

This recurrence is in fact a *probabilistic recurrence* in that the sizes of the subproblems are random variables. The above recurrence can be seen as measuring the *expected* running time (where we use the fact that each of the two subproblems expects to contain $m/2$ edges). A high-probability analysis will appear in the full paper. Solving this recurrence yields the following result:

Theorem 3.3 *In a graph with minimum cut c, an s-t maximum flow of value v can be found in $\tilde{O}(mv/\sqrt{c})$ time (LV).*

Note that we cannot apply sampling in the cleanup phase, because the graph we are manipulating in the cleanup phase is directed, while our sampling theorems apply only to undirected graphs.

Corollary 3.4 *A $(1+\epsilon)$-minimal s-t cut can be found with high probability in $\tilde{O}(mv/\epsilon^3c^2)$ time (MC).*

Proof: Apply the cut approximation algorithm of the previous section. Instead of using the standard augmenting path max-flow algorithm, use the faster one just presented. Since the skeleton has minimum cut $\Theta(pc)$, the running time of the skeletal max-flow computation is improved from $O((pm)(pv))$ to $\tilde{O}((pm)(pv)/\sqrt{pc})$. ■

3.3 Approximate Maximum Flows

A slight variation on the previous algorithm will compute approximate maximum flows.

Lemma 3.5 *In a graph with minimum cut c and s-t maximum flow v, a $(1-\epsilon)$-maximal s-t flow can be found in $\tilde{O}(mv/\epsilon c)$ time (MC).*

Proof: Given p as determined by ϵ, randomly partition the edges into $1/p$ groups, creating $1/p$ graphs. Each looks like a p-skeleton and thus has a maximum flow of value at least $pv(1-\epsilon)$ which can be computed in $O((pm)(pv)/\sqrt{pc})$ time as in the previous section. Adding the $1/p$ flows which result gives a flow of value $v(1-\epsilon)$. The running time is $O((1/p)(pm)(pv)/\sqrt{pc})$. ■

3.4 Las Vegas Algorithms

Our max-flow and min-cut approximation algorithms are both Monte Carlo, since they are not *guaranteed* to give the correct output (though the error probability can be made arbitrarily small). However, by combining the two approximation algorithms, we can certify the correctness of our results and obtain a *Las Vegas* algorithm for both problems—one which is guaranteed to find the right answer, but has a small probability of taking a long time to do so.

Corollary 3.6 *In a graph with minimum cut c and s-t maximum flow v, a $(1-\epsilon)$-maximal s-t flow and a $(1+\epsilon)$-minimal s-t cut can be found in $\tilde{O}(mv/\epsilon c)$ time (LV).*

Proof: Run both the approximate min-cut and approximate max-flow algorithms, obtaining a $(1-\epsilon/2)$-maximal flow of value v_0 and a $(1-\epsilon/2)$-minimal cut of value v_1. We know that $v_0 \leq v \leq v_1$, so to verify the correctness of the results all we need do is check that $(1+\epsilon/2)v_0 \geq (1-\epsilon/2)v_1$, which happens with high probability. To make the algorithm Las Vegas, we repeat the two algorithms until each demonstrates the other's correctness. ■

An intriguing open question is whether this combination of an approximate cut and flow together can be used to identify an actual maximum flow more quickly than the exact algorithm previously described.

4 Global Minimum Cuts

We now show how sampling can be used to find the (global) minimum cut of a graph in $\tilde{O}(m\sqrt{c})$ time, where c is the value of the minimum cut. Our technique is similar to the one presented previously for finding maximum flows, and is again based on the divide and conquer packing paradigm of [14]. The change is that instead of using the standard augmenting paths technique to pack paths, we use a matroid augmentation technique developed by Gabow [7] to pack spanning trees.

Building on the work of Edmonds [3], Gabow defines a *complete k-intersection* to be a set of k disjoint spanning trees satisfying certain technical conditions. The key fact is that a graph has connectivity k if and only if it contains a complete k-intersection. Thus the minimum cut is simply the maximum value c for which a complete c-intersection exists.

Gabow's min-cut algorithm uses a subroutine called the Round Robin Algorithm (RR). This algorithm takes as input a graph G with a complete k-intersection. In $O(m\log(n^2/m))$ time, it either returns a complete $(k+1)$-intersection or proves that the minimum cut is k by returning an cut of value k. RR can therefore be seen as a cousin of the standard augmenting path algorithm for maximum flows: instead of augmenting by a path, it augments by a spanning tree.

Gabow's algorithm for finding a minimum cut is to repeatedly call RR until it fails. The number of calls needed is just the value c of the minimum cut; thus the total running time of his algorithm is $O(cm\log(n^2/m))$.

Lemma 4.1 *A minimum cut of value c can be found in $\tilde{O}(m\sqrt{c})$ time (LV).*

Proof: Our goal is to find a maximum complete intersection. We solve this problem as we did the max-flow problem. Randomly partition the edges into two groups, creating two undirected graphs. Recursively find maximum complete intersections in each of the two subgraphs. Their union forms a complete intersection in the original graph. Use RR to augment this complete intersection to a maximum one.

The running time is analyzed exactly as for the max-flow algorithm. By Theorem 3.1, each of the two subproblems has minimum cut $c/2 - \tilde{O}(\sqrt{c})$ and thus contains a complete $(c/2-\tilde{O}(\sqrt{c}))$-intersection with high probability. Putting these two complete intersections together yields a complete $c-\tilde{O}(\sqrt{c})$ intersection. Thus after solving the subproblems we need call RR only $\tilde{O}(\sqrt{c})$ times, using $\tilde{O}(m\sqrt{c})$ time, in order to finish. As with maximum flows, this cleanup step dominates the running time. ■

The improved time for computing a complete c-intersection has other ramifications in Gabow's work [8]. He presents other algorithms for which computing a maximum complete intersection is the computational bottleneck. He presents an algorithm for computing a compact *m-tree* representation of all minimum cuts, and shows notes that this representations can be converted to the older $O(n)$-space cactus representation [2] in linear time. He also gives an algorithm for finding a minimum set of edges to add to augment the connectivity of a graph from c to $c+\delta$. In both of these algorithms, computing the minimum cut forms the bottleneck in the running time.

Corollary 4.2 *The cactus and m-tree representations of all minimum cuts in an undirected graph can be constructed in $\tilde{O}(m\sqrt{c})$ time (LV).*

Corollary 4.3 *A minimum set of edges augmenting the connectivity of a graph from c to $c+\delta$ can be computed in $\tilde{O}(m+n(c^{3/2}+\delta c+\delta^2))$ time (LV).*

4.1 Approximating the Minimum Cut

Our faster min-cut algorithm can be plugged into the result of [15]. There we presented a Monte Carlo algorithm to find a $(1+\epsilon)$-minimal cut in $\tilde{O}(m+n/\epsilon^4)$ time. That algorithm used Gabow's min-cut algorithm; if we instead use the new, faster one, we improve the running time of the $(1+\epsilon)$-approximation algorithm from $\tilde{O}(m+n/\epsilon^4)$ to $\tilde{O}(m+n/\epsilon^3)$.

We can also use the Round Robin based algorithm to approximately verify a minimum cut. The algorithm of [15], as well as the improvement just described, is Monte Carlo since there is no guarantee that the value of the min-cut approximation will be correct. However, we can use approximately maximal complete intersections to verify the approximation just as we used approximate maximum flows to verify approximate s-t minimum cuts.

Lemma 4.4 *A $(1-\epsilon)$-approximation to the maximum complete intersection can be found in $O(m/\epsilon)$ time (MC).*

Proof: Given ϵ and its corresponding p, divide the graph in $1/p$ pieces, find a maximum complete intersection in each of the pieces independently, and add the intersections. The analysis proceeds exactly as in the approximate max-flow algorithm of Section 3.3. ∎

Corollary 4.5 *A $(1+\epsilon)$-minimal cut can be found in $\tilde{O}(m/\epsilon + n/\epsilon^3)$ time (LV).*

Proof: As in Corollary 3.6, the combination of a cut of value $(1+\epsilon/2)c$ and a complete $(1-\epsilon/2)c$-intersection brackets the minimum cut between these two bounds. ∎

5 Weighted Graphs

We now investigate the changes that occur when we apply our flow and tree packing algorithms to weighted graphs. We model an edge of weight w as a collection of w unweighted edges and apply the unweighted graph algorithm. Our s-t cut approximation algorithm has roughly the same running time as in the unweighted case: an algorithm of [15] allows us to generate an $O(nv/\epsilon^2 c)$-edge p-skeleton in linear time (linear in the number of edges, not the total edge weight); we can then use it to find the approximate cut in $O(m+n(v/c)^2/\epsilon^3)$ time as before. The $O(mv/\epsilon^3 c^2)$ bound no longer follows, since it need no longer be the case that skeleton has only pm edges.

We can also adapt the max-flow algorithms. If we directly simulated the unweighted graph algorithm, we would simulate the random partitioning of the edges into two groups by generating a binomial distribution for each weighted edge in order to determine how much of its weight went to each of the two subgraphs. To avoid having to generate such complicated distributions, we return to Theorem 2.1 and use the following approach. If w is even, assign weight $w/2$ to each group. If w is odd, then assign weight $\lfloor w/2 \rfloor$ to each group, and flip a coin to decide which group gets the remaining single unit of weight. Since the minimum expected cut ($\hat{c}$ of Theorem 2.1) which results in each half is still $c/2$, we can deduce as in unweighted case that little augmentation need be done after the recursive calls.

We have described the change in implementation, and correctness is clear, but we have to change the time bound analysis. We add a new parameter and analyze the algorithm in terms of the number of edges m, the minimum cut c, the desired flow value v, and the *total weight* W of edges in the graph. Note the two subgraphs that we recurse on have average weight roughly $W/2$. In order to contrast with previous scaling techniques, we also use the *average edge weight* $U = W/m$ which is no more than the maximum edge weight.

Lemma 5.1 *The minimum cut of value c can be found in $\tilde{O}(m\sqrt{cU})$ time (LV).*

Proof: Using our divide and conquer recursion with Gabow's Round Robin Algorithm gives the following recurrence:

$$\begin{aligned} T(m,c,W) &= 2T(m,c/2,W/2) + m\sqrt{c} \\ T(m,c,m) &= m\sqrt{c}, \end{aligned}$$

showing that the total running time is $\tilde{O}(\sqrt{mcW}) = \tilde{O}(m\sqrt{cU})$. ∎

Corollary 5.2 *In a weighted graph, a $(1+\epsilon)$-minimal cut can be found in $\tilde{O}(m\sqrt{U}/\epsilon)$ time.*

A similar result can be derived if we use the same algorithm to find flows, replacing Gabow's Round Robin Algorithm with standard augmenting paths.

Corollary 5.3 *A maximum flow of value v can be found in $\tilde{O}(mv\sqrt{U/c})$ time (LV).*

Corollary 5.4 *A $(1-\epsilon)$-maximal flow of value v can be found in $\tilde{O}(mv\sqrt{U}/\epsilon c)$ time.*

6 Other Cut Problems

6.1 Parallel Flow Algorithms

In the s-t min-cut problem the need for the final "cleanup" augmentations interferes with the development of efficient $\mathcal{RNC}$ algorithms for the problems, because there are no good parallel reachability algorithms for directed graphs. However, we can still take advantage of some of the divide and conquer technique in a partial parallelization of the problem. Khuller and Schieber [17] give an algorithm for finding disjoint s-t paths in undirected graphs. It uses a subroutine which augments a set of k disjoint s-t paths to $k+1$ if possible, using $\tilde{O}(k)$ time and kn processors. This lets them find a flow of value v in in $\tilde{O}(v^2)$ time using vn processors. We can speed up this algorithm by applying the randomized divide and conquer technique we used for maximum flows. Finding the final augmentations after merging the results of the recursive calls is the dominant step in the computation, and requires $\tilde{O}(v^2/\sqrt{c})$ time using vn processors. Thus we decrease the running time of their algorithm by by an $\tilde{O}(\sqrt{c})$ factor, without changing the processor cost.

6.2 Edge Separators

Klein, Stein, and Tardos [19] give a fast concurrent flow algorithm which they use to improve the running time of Leighton and Rao's [21] approximation algorithm for the $\mathcal{NP}$-complete *quotient cut problem.* Defining the *quotient* of a cut (A, B) of value v to be $v/(|A||B|)$, the quotient cut problem is to find a cut of minimum quotient. Their algorithm runs in $O(m^2 \log m)$ time, and finds a cut with quotient within an $O(\log n)$ factor of the optimum. Consider a skeleton of the graph which approximates cuts to within a $(1\pm\epsilon)$ factor. Since the denominator of a cut's quotient is unchanged in the skeleton, the quotients in the skeleton also approximate their original values to within a $(1 \pm \epsilon)$ factor. It follows that we can take $p = \tilde{O}(1/c)$ and introduce only a constant factor error in the approximation.

Theorem 6.1 *An $O(\log n)$-approximation to the minimum quotient cut can be computed in $O((m/c)^2 \log m)$ time (MC).*

6.3 Orienting a Graph

Given an undirected graph, the *k-connected orientation problem* is to find an assignment of directions to the edges such that the resulting directed graph is k-connected. Gabow [9] cites a theorem of Nash-Williams [26] showing that a solution exists if and only if the input graph is $2k$-connected, and also gives a submodular-flow-based algorithm for finding the orientation in $O(kn^2(\sqrt{kn}+k^2\log(n/k)))$ time. We have the following result:

Lemma 6.2 *A $(k-O(\sqrt{k\log n}))$-connected orientation of a $2k$-connected graph can be found in linear time.*

Proof: Orient each edge randomly with probability 1/2 in each direction. A minor adaptation of Theorem 2.1 shows that with high probability, for each cut, there will be at least $k - O(\sqrt{k\log n})$ edges oriented in each direction. In other words, every directed cut will have a value exceeding the desired one. ∎

Using this randomly oriented graph as a starting point in Gabow's algorithm also allows us to speed up that algorithm by a factor of $\tilde{O}\sqrt{k}$.

6.4 Integral Multicommodity Flows

Suppose we are given an unweighted graph G and a multicommodity flow problem with k source-sink pairs (s_i, t_i) and demands d_i. Let c_i be the value of the s_i-t_i minimum cut and suppose that $\sum d_i/c_i \leq 1$. Then it is obvious that there is a fractional solution to the problem: divide the graph into k new graphs G_i, giving a d_i/c_i fraction of the capacity of each edge to graph G_i. Then the s_i-t_i minimum cut of G_i has value exceeding d_i, so commodity i can be routed in graph G_i. There has been some interest in the question of when an *integral* multicommodity flow can be found [6, 23]. Sampling lets us find an integral solution, and find it faster, if we have some slack. Rather than assigning a fraction of each edge to each graph, assign each edge to a graph G_i with probability proportional to d_i/c_i. We now argue as for the flow algorithms, that given the right conditions on c, each graph G_i will be able to integrally satisfy the demands for commodity i. Thus k max-flow computations will suffice to route all the commoditites. In fact, in an unweighted graph, if m_i is the number of edges in G_i, we have that $\sum m_i = m$, so that the max-flow computations will take a time of $O(\sum m_i n) = O(mn)$ time. Various results follow; we give one as an example:

Lemma 6.3 *Suppose that each $d_i \geq \log n$, and that $\sum c_i/d_i \leq 1/2$. Then an integral multicommodity flow satisfying the demands can be found in $O(mn)$ time.*

6.5 Approximate Gomory-Hu Trees

The Gomory-Hu tree [12] is a well known data structure of size $O(n)$ which allows queries as to the minimum cut between any pair of graph vertices to be answered in $O(1)$ time. It is constructed by performing n max-flow computations. If we are willing to settle for approximately correct answers to the cut queries, then it is sufficient for us to construct the Gomory-Hu tree of a skeleton. This yields the following, based on the fact that every s-t minimum cut has value at most $n-1$ and on the use of Corollary 3.4:

Corollary 6.4 *Given an undirected graph, a $(1+\epsilon)$-approximate Gomory-Hu tree can be constructed for it in $O(m(1+(n/(\epsilon^2 c))^2))$ time (MC).*

7 Network Design

We now use Theorem 2.1 to analyze randomized rounding algorithms for network design problems. Due to space limitations, we focus here on a restricted instance of network design called the *minimum k-edge-connected subgraph problem.* Given an input graph, the goal is to find a k-connected (connectivity denotes edge-connectivity) subgraph of the input graph that contains the minimum number of edges. We assume that the input graph is k-connected, since otherwise there is no solution. The problem is $\mathcal{NP}$-complete even for $k = 2$ [5]. Williamson et al [29, 11] have recently given powerful new approximation algorithms for network design problems. In the course of their discussion, they observe that the optimum *fractional* solution to our problem, in which each edge is to be assigned a weight between 0 and 1 such that the resulting weighted graph is k-connected with a minimum total weight, can be solved on polynomial time using interior point algorithms even though the number of constraints is exponential (see [10] for details).

For the specific case of the k-connected subgraph problem, it is easy to achieve an approximation ratio of two (a different 2-approximation algorithm extending to weighted graphs is given in [18]). Given the input graph, suppose we find and delete a spanning forest. Then we will have deleted at least one edge from each cut of the graph. If we find and delete a spanning forest of the remaining graph, we will have deleted 2 edges from each cut. If we do this k times, we will have deleted at least k edges from each cut. Therefore, the set of deleted edges forms a k-connected subgraph of G. Since we delete at most n edges per spanning forest, the deleted graph will have at most kn edges. In fact, we can find the k disjoint forests in $O(m)$ time using Nagamochi and Ibaraki's sparse certificate algorithm (discussed in the introduction). Now note that every k-connected graph has minimum degree k and thus has at least $kn/2$ edges. Thus the kn-edge graph is a 2-approximation to the optimum graph.

We improve this result with randomized rounding:

Theorem 7.1 *A* $(1+O(\sqrt{(\log n)/k}))$*-approximation to the minimum* k*-connected subgraph can be found in polynomial time (LV).*

Proof: As noted in [10], we can use linear programming to find a fractional solution F which assigns a weight $0 \leq p_e \leq 1$ to each edge e. By definition, F has minimum cut k. Suppose the solution has total weight W (which must exceed $kn/2$ to give minimum degree k). Clearly W is a lower bound on the number of edges in the integral solution. Treating the weights p_e as probabilities, we build a subgraph H by including edge e with probability p_e. Since F is the "expected value" of H in the sense of Theorem 2.1 and has minimum cut k, Theorem 2.1 says that H has minimum cut $k - O(\sqrt{k \log n})$ with high probability. By the Chernoff bound, the number of edges in H is likely to be little more than W. As in the 2-approximation algorithm, we can increase H to a k-connected graph by finding $O(\sqrt{k \log n})$ disjoint spanning forests among the edges not already used. The total number of edges in the graph is therefore at most $W + O(n\sqrt{k \log n}) = W(1 + O(\sqrt{(\log n)/k}))$. ■

Our technique extends to many other network design problems. For example, in the variant where the edges have arbitrary costs and the problem is specified by giving a connectivity requirement between k and $2k$ for every pair of vertices, combining randomized rounding with the algorithm of Williamson et al lets us improve their approximation ratio from $O(\log k)$ to $O(\log \log n)$. If it is permissible to use graph edges more than once, our approximation ratio improves to $1 + O((\log n)/k)$, even in the generalization in which the edges have varying costs.

8 Conclusion

This work has demonstrated the effectiveness of a new tool for solving problems involving cuts. We have extended the work of [15], showing how random sampling tends to "preserve" all cut information in a graph. This suggests that we might want to try to reformulate other problems in terms of cuts so that the random sampling methods can be applied to them.

One result of this approach has been to reduce large max-flow and min-cut problems on undirected graphs to small max-flow and min-cut problems on directed graphs. Our techniques are in a sense "meta-algorithms" in that improved cut or flow algorithms which are subsequently developed may well be accelerated by application of our technique. In particular, our exact algorithms running times are dominated by the time needed to perform "cleaning up" augmenting path computations; any improvement in the time to compute a sequence of augmenting paths would translate immediately into an improvement in our algorithm's running time. One way to get such an improvement might be to generalize our sampling theorems to the case of directed graphs.

A different way to interpret skeletons is in the context of expanders. Skeletons of the complete graph are expanders. What we have shown is that skeletons in any graph have a sort of expansion property in that the number of edges crossing a cut is as much as could be expected given the sampling probability. This raises the natural question of whether skeletons, like expanders, can be constructed deterministically.

Another interesting goal would be to reformulate other network problems in terms of cuts so that the sampling theorems could be applied.

References

[1] H. Chernoff. "A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations". *Annals of Mathematical Statistics*, 23:493–509, 1952.

[2] E. A. Dinitz, A. V. Karzanov, and M. V. Lomonosov. "On the structure of a family of minimal weighted cuts in a graph". In A. A. Fridman, editor, *Studies in Discrete Optimization*, pages 290–306. Nauka Publ., 1976.

[3] J. Edmonds. "Minimum partition of a matroid into independents subsets". *Journal of Research of the National Bureau of Standards*, 69:67–72, 1965.

[4] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig. "Sparsification—a technique for speeding up dynamic graph algorithms". In *Proceedings of the* 33^{rd} *Annual Symposium on Foundations of Computer Science*, pages 60–69, Oct. 1992.

[5] K. P. Eswaran and R. E. Tarjan. "Augmentation problems". *SIAM Journal on Computing*, 5:653–665, 1976.

[6] L. R. Ford, Jr. and D. R. Fulkerson. *Flows in Networks*. Princeton Univ. Press, Princeton, NJ, 1962.

[7] H. N. Gabow. "A matroid approach to finding edge connectivity and packing arborescences". In *Proceedings of the* 23^{rd} *Annual Symposium on Theory of Computing*. ACM Press, May 1991.

[8] H. N. Gabow. "Applications of a poset representation to edge connectivity and graph rigidity". In *Proceedings of the* 32^{nd} *Annual Symposium on Foundations of Computer Science*, pages 812–821, Oct. 1991.

[9] H. N. Gabow. "A framework for cost-scaling algorithms for submodular flow problems". In *Proceedings of the* 34^{th} *Annual Symposium on Foundations of Computer Science*, pages 449–458, Nov. 1993.

[10] H. N. Gabow, M. X. Goemans, and D. P. Williamson. "An efficient approximation algorithm for the survivable network design problem". In *Proceedings of the Third MPS Conference on Integer Programming and Combinatorial Optimization*, pages 57–74, 1993.

[11] M. X. Goemans, A. Goldberg, S. Plotkin, D. Shmoys, E. Tardos, and D. Williamson. "Improved approximation algorithms for network design problems". In *Proceedings of the* 5^{th} *ACM-SIAM Symposium on Discrete Algorithms*, pages 223–232, Jan. 1994.

[12] R. E. Gomory and T. C. Hu. "Multi-terminal network flows". *Journal of the Society of Industrial and Applied Mathematics*, 9(4):551–570, Dec. 1961.

[13] D. R. Karger. "Global min-cuts in $\mathcal{RNC}$ and other ramifications of a simple mincut algorithm". In *Proceedings of the* 4^{th} *ACM-SIAM Symposium on Discrete Algorithms*, pages 21–30, Jan. 1993.

[14] D. R. Karger. "Random sampling in matroids, with applications to graph connectivity and minimum spanning trees". In *Proceedings of the* 34^{th} *Annual Symposium on Foundations of Computer Science*, 1993.

[15] D. R. Karger. "Using randomized sparsification to approximate minimum cuts". In *Proceedings of the* 5^{th} *ACM-SIAM Symposium on Discrete Algorithms*, Jan. 1994.

[16] D. R. Karger and C. Stein. "An $\tilde{O}(n^2)$ algorithm for minimum cuts". In *Proceedings of the* 25^{th} *Annual ACM Symposium on Theory of Computing*, pages 757–765, 1993.

[17] S. Khuller and B. Schieber. "Efficient parallel algorithms for testing connectivity and finding disjoint s-t paths in graphs". *SIAM Journal on Computing*, 20(2):352–375, Apr. 1991.

[18] S. Khuller and U. Vishkin. "Biconnectivity approximations and graph carvings". In *Proceedings of the* 24^{th} *Annual ACM Symposium on Theory of Computing*, pages 759–770, May 1992.

[19] P. N. Klein, C. Stein, and E. Tardos. "Leighton-Rao might be practical: Faster approximation algorithms for concurrent flow with uniform capacities". In *Proceedings of the* 22^{nd} *Annual Symposium on Theory of Computing*, pages 310–321. ACM, May 1990.

[20] P. N. Klein and R. E. Tarjan. "A linear-time algorithm for minimum spanning tree". In *Proceedings of the* 26^{th} *ACM Symposium on Theory of Computing*, 1994. To appear.

[21] T. Leighton and S. Rao. "An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms". In *Proceedings of the* 29^{th} *Annual Symposium on Foundations of Computer Science*, pages 422–431. IEEE, Oct. 1988.

[22] D. W. Matula. "A linear time $2+\epsilon$ approximation algorithm for edge connectivity". In *Proceedings of the* 4^{th} *ACM-SIAM Symposium on Discrete Algorithms*, pages 500–504, Jan. 1993.

[23] H. Nagamochi and T. Ibaraki. On max-flow min-cut and integral flow properties for multicommodity flows in directed networks. *Information Processing Letters*, 31:279–285, 1989.

[24] H. Nagamochi and T. Ibaraki. "Computing edge connectivity in multigraphs and capacitated graphs". *SIAM Journal of Discrete Mathematics*, 5(1):54–66, Feb. 1992.

[25] H. Nagamochi and T. Ibaraki. "Linear time algorithms for finding k-edge connected and k-node connected spanning subgraphs. *Algorithmica*, 7:583–596, 1992.

[26] C. S. J. A. Nash-Williams. "Well-balanced orientations of finite graphs and unobtrusive odd-vertex-pairings". In W. T. Tutte, editor, *Recent Progress in Combinatorics*, pages 133–149. Academic Press, 1969.

[27] P. Raghavan and C. Thompson. "Probabilistic construction of deterministic algorithms: Approximate packing integer programs". *Journal of Computer and System Sciences*, 37(2):130–43, Oct. 1988.

[28] R. E. Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics, 1983.

[29] D. Williamson, M. X. Goemans, M. Mihail, and V. V. Vazirani. "A primal-dual approximation algorithm for generalized steiner problems". In *Proceedings of the* 25^{th} *Annual ACM Symposium on Theory of Computing*, pages 708–717, May 1993.