Using Randomized Sparsification to Approximate Minimum Cuts

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

October 29, 1993

Abstract

We introduce the concept of randomized sparsification of a weighted, undirected graph. Randomized sparsification yields a sparse unweighted graph which closely approximates the minimum cut structure of the original graph. As a consequence, we show that a cut of weight within a $(1+\epsilon)$ multiplicative factor of the minimum cut in a graph can be found in $O(m + n(\log^3 n)/\epsilon^4)$ time; thus any constant factor approximation can be achieved in $\tilde{O}(m)$ time. Similarly, we show that a cut within a multiplicative factor of α of the minimum can be found in $\mathcal{RNC}$ using $m + n^{2/\alpha}$ processors. We also investigate a parametric version of our randomized sparsification approach. Using it, we show that for a graph undergoing a series of edge insertions and deletions, an $O(\sqrt{1+2/\epsilon})$-approximation to the minimum cut value can be maintained at a cost of $\tilde{O}(n^{\epsilon+1/2})$ time per insertion or deletion. If only insertions are allowed, the approximation can be maintained at a cost of $\tilde{O}(n^{\epsilon})$ time per insertion.

1 Introduction

1.1 Minimum Cuts. This paper studies the min-cut problem. Given a graph with n vertices and m (possibly weighted) edges, we wish to partition the vertices into two non-empty sets so as to minimize the number or total weight of edges crossing between them. Throughout this paper, the graph is assumed to be connected because otherwise the problem is trivial. We also require that all edge weights be non-negative, because otherwise the problem is $\mathcal{NP}$-complete by a trivial transformation from the maximum-cut problem [GJ79, page 210]. The problem actually has two variants: in the *s-t min-cut problem* we require that two specified vertices s and t be on opposite sides of the cut; in what we call the *min-cut problem* there is no such restriction.

Particularly on unweighted graphs, solving the min-cut problem is sometimes referred to as finding the *connectivity* of a graph; that is, determining the minimum number of edges (or minimum total edge weight) that must be removed to disconnect the graph.

Throughout this paper, we will focus attention on an n vertex, m edge graph with minimum cut value c. The fastest presently known algorithm for finding minimum cuts in weighted undirected graphs is the Recursive Contraction Algorithm (RCA) of Karger and Stein [KS93]; it runs in $O(n^2 \log^3 n)$ time. An algorithm by Gabow [Gab91] finds the minimum cut in an unweighted graph in time $O(m + c^2 n \log(n/c))$, where c is the value of the minimum cut. It is thus faster than the RCA on unweighted graphs with small minimum cuts $(c < \sqrt{n})$.

1.2 New Results. This paper studies algorithms for *approximating* the minimum cut. To this end, we make the following definition:

DEFINITION 1.1. *An* α-approximation to the minimum cut, *or more concisely an* α-minimal cut, *is a cut whose weight is within a multiplicative factor of* α *of the minimum cut. An* α-approximation algorithm *is one which finds an* α*-minimal cut in every input graph.*

In this paper, we give a collection of minimum cut approximation algorithms. They are based on a randomized algorithm for taking a weighted undirected graph and constructing a sparse unweighted graph, or *skeleton*, which closely approximates the minimum cut information of the original graph. Finding a minimum cut in the skeleton gives information about the minimum cut in the original graph. Because the skeleton is sparse and unweighted, fast specialized minimum cut algorithms (such as Gabow's) can be applied.

We use graph skeletons in a new sequential approximation algorithm for minimum cuts. For any $\epsilon > 0$, the algorithm finds a $(1+\epsilon)$-minimal cut in $O(m + n(\log^3 n)/\epsilon^4)$; thus, in particular, if ϵ is any constant it finds a $(1+\epsilon)$-approximation in $O(m + n\log^3 n)$

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

time. Another consequence is an $\mathcal{RNC}$ α-approximation algorithm which uses $m + n^{2/\alpha}$ processors.

We also consider what can be viewed as a parametrized construction of skeletons. This lets us derive an alternative $\mathcal{RNC}$ $(\sqrt{1+2/\epsilon})$-approximation algorithm which uses $O(mn^{\epsilon})$ processors. Although this processor bound is dominated by the previous one for all interesting approximation values, it is interesting because the cut is found by minimum spanning tree computations. More importantly, it yields algorithms which *dynamically* maintain a $(\sqrt{1+2/\epsilon})$-approximation to the minimum cut value as edges are inserted in and deleted from a graph; the cost per update is $O(n^{\epsilon})$ if only insertions are allowed, and $O(n^{\epsilon+1/2})$ if both insertions and deletions are permitted.

For simplicity, we will generally focus on unweighted graphs. However, all these techniques extend to weighted graphs, and we will occasionally note small changes which need to be made for them.

1.3 Related Work. Recently, Matula [Mat93] gave a $(2+\epsilon)$-approximation algorithm for the minimum cut. It runs in $O(m)$ time so long as ϵ is a constant.

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 has it as well. The advantage is that since the certificate is sparse, the property can be verified more quickly. The skeleton is a sparse *approximate* certificate. Sparse certificates have already been used successfully in several areas. Eppstein *et al* [EGIN92] give sparsification techniques which improve the running times of dynamic algorithms for numerous graph problems such as connectivity, bipartitioning, and and minimum spanning trees.

Our algorithms make use of a particular sparse certificate we now define:

Definition 1.2. *A* sparse k-connectivity certificate *for G is a subgraph of G containing at most kn edges such that any cut of value at most k in the original graph has the same value in the certificate. (For weighted graphs the total* weight *in the certificate is at most kn.)*

Nagamochi and Ibaraki [NI92b, NI92a] give an algorithm which finds a sparse k-connectivity certificate in $O(m)$ time on unweighted graphs and $O(m + n\log n)$ time on weighted graphs. Their algorithm plays a central role in Matula's approximation algorithm. Gabow [Gab91] uses it also, improving the running time of his minimum cut algorithm from $O(cm)$ to $O(m + c^2 n)$ on undirected graphs. Cheriyan, Kao, and Thurimella [CKT93] give a parallel sparse k-connectivity certificate algorithm which runs on $\tilde{O}(k)$ time[1] using m processors. They use it to improve the processor bounds for parallel k-vertex connectivity algorithms.

2 Randomized Sparsification

The main tool used in this paper is *randomized sparsification.* Given the input graph, which for the time being we assume unweighted, we construct a new sparse graph, called a *skeleton,* on the same vertices such that every small cut in the original graph corresponds to a small cut in the new graph, and vice versa.

To construct this graph we use random sampling. Given a graph G, we construct a skeleton $G(p)$ on the same vertices by including each edge of G independently with probability p. Since this graph has the same vertices, there is an obvious correspondence between a cut in the skeleton and the cut in G which has the same vertex partition. Intuitively, a cut with k crossing edges in G will have about pk of its edges sampled and included in $G(p)$; thus a cut of value k in G should correspond to a cut of value pk in $G(p)$. If this correspondence in values really happened, then the minimum cut in $G(p)$ would have value pc, and the corresponding cut in G would be a (minimum) cut of value c.

Unfortunately, since $G(p)$ is a random graph, this exact proportionality does not hold. While it is true that the $G(p)$ cut corresponding to the minimum cut of G will have minimum *expected* value in $G(p)$, it does not follow that this cut will have minimum *actual* value. So we aim for an approximation. What we would like to say is that it is extremely unlikely that any large cut of G will correspond to a small cut in $G(p)$. Thus the cut in G which corresponds to a minimum cut in $G(p)$ would likely be approximately minimal. This is certainly true for any *particular* cut of G. However, since there are so many cuts, some of them will deviate widely from their expected values; there is thus no *a priori* reason to assume that the minimum cut in the sampled graph will correspond to even a small cut in the original graph. However, the special structure of graphs makes the argument work. We apply the following lemma.

Lemma 2.1. ([Kar93]) *In an undirected graph, the number of α-minimal cuts is less than $n^{2\alpha}$.*

We also use the Chernoff bound to argue that the large cuts are so unlikely to induce a minimum cut in the sampled graph that despite their number they can be ignored.

[1] The notation $\tilde{O}(f)$ denotes $O(f \operatorname{polylog}(n))$

LEMMA 2.2. ([CHE52]) *Let X be a Bernoulli random variable on n trials with success probability p and expected number of successes $\mu = np$. Then*

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

Lemma 2.1 says that the number of cuts within an α factor of the minimum increases exponentially with α. On the other hand, Lemma 2.2 says that the probability one such cut corresponds to a sufficiently small $G(p)$-cut to cause trouble decreases exponentially with α. Combining these two lemmas and balancing the exponential rates will give us the following result:

THEOREM 2.1. *Let G be any graph with minimum cut c and let $p = k/c$, where $k = 6(\ln n)/\epsilon^2$. Then the probability that the value of some cut in $G(p)$ has value more than $(1+\epsilon)$ times or less than $(1-\epsilon)$ times its expected value is $O(1/n)$.*

Proof. Consider any particular cut of value αc. By the Chernoff bound, the probability that its corresponding cut in $G(p)$ does not have value in the desired bounds of the lemma is at most $P(\alpha) = e^{-\epsilon^2\alpha pc/2}$. Now let $f(\alpha)$ be the number of cuts of value αc in G. Then the probability that the lemma fails to hold is at most the probability that some minimum cut exceeds its bounds, namely

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

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 2.1: Defining $F(x) = \sum_{\alpha\leq x} f(x)$, we require that $F(x) \leq n^x$. In fact, we will give the adversary extra freedom and allow him 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

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

COROLLARY 2.1. *Under the above construction, but with $k = 54(\ln n)/\epsilon^2$, with high probability the minimum cut in $G(p)$ will correspond to a $(1+\epsilon)$-minimal cut of G.*

Proof. Changing k corresponds to dividing ϵ by 3. Applying the Chernoff bound to the minimum cut edges, the probability is high that at most $(1+\epsilon/3)pc$ of the minimum cut edges will be sampled. Thus $G(p)$ has minimum cut at most $(1+\epsilon/3)pc$. On the other hand, Theorem 2.1 proves that with high probability any cut of value exceeding $pc(1+\epsilon/3)/(1-\epsilon/3) < (1+\epsilon)pc$ will have at least $(1+\epsilon/3)pc$ of its edges sampled and thus cannot correspond to a minimum cut in $G(p)$.

The above results obviously generalize to integer weighted graphs if we treat an edge of weight w as a collection of w unweighted edges. Non-integral edge weights can be treated similarly if we scale and round appropriately [Kar93].

Skeletons can also be seen as a generalization of *expanders* [AKS87]. Just as expanders approximate the expected cut values in the complete graph, skeletons approximate the expected cut values in arbitrary graphs. This motivates us to ask whether it is possible to deterministically construct skeletons, as is the case for expanders [GG81]. Furthermore, just as the expander of [GG81] has constant degree, it may be possibile to deterministically construct a skeleton with a constant minimum cut, rather than the $\Omega(\log n)$ minimum cut produced by the randomized construction.

3 Sequential Algorithms

We now give sequential algorithms for constructing $G(p)$ and show how they easily yield an approximation algorithm. For unweighted graphs, constructing a sparse certificate appears relatively simple: examine edges one at a time, flipping a coin with the appropriate bias to decide whether each edge should be included. However there are two drawbacks. The first is that if we wish to claim that this takes linear time, we must assume that a sample from the probability distribution can be generated in $O(1)$ time. Furthermore, it does not extend immediately to weighted graphs. We may therefore wish to apply the slightly more involved algorithm in Figure 1.

This algorithm, given k from Theorem 2.1, generates a skeleton with minimum expected cut exceeding k. Note that the theorem assumes the minimum expected cut is exactly k; however, if it is larger, it only increases the degree of accuracy in our approximation. For simplicity, we continue to focus on unweighted graphs; however, the following algorithm generalizes to weighted graphs without any change in running time. For simplicity we also assume $\epsilon < 1$.

1. Use Matula's linear time algorithm to estimate the minimum cut c to a factor of 3. The estimate c' has $c < c' < 3c$.
2. Use Nagamochi and Ibaraki's algorithm [NI92b] to construct a sparse $2c'$-connectivity certificate in linear time.
3. Choose $6nk$ edges at random from the certificate.

Figure 1: Constructing a Skeleton

To prove correctness, note that the sparse certificate produced in Step 2 has the same $(1+\epsilon)$-minimal cuts as the original graph did. Furthermore, it contains at most $6nc$ edges. As $6nk$ edges are chosen randomly from among at most $6nc$, the probability that any one edge is selected exceeds k/c, as desired. Since the number of edges selected is fixed, there is a slight negative correlation between the events of two different edges being selected; however, this dependence is negligible and has no effect on the analysis (details will be in the full paper).

Since the $6kn$ edges can be selected in $O(m + kn \log n)$ time, even under the pessimistic assumption that generating random variates requires $\Theta(\log n)$ time per variate, the running time of this entire procedure is $O(m + kn \log n)$.

This technique generalizes easily to weighted graphs. Further details regarding the random selection of (weighted) edges may be found in [Kar93, KS93].

Given the above tools, constructing a sequential approximation algorithm for the minimum cut is straightforward:

THEOREM 3.1. *A $(1+\epsilon)$-minimal cut in an undirected graph can be found with high probability in $O(m + (n\log^3 n)/\epsilon^4)$ time.*

Proof. Take $k = \Theta((\log n)/\epsilon^2)$ and use the randomized sparsification algorithm to construct a skeleton with $O(kn)$ edges and minimum cut $c = \Theta(k)$. Theorem 2.1 proves that a minimum cut in this skeleton corresponds to an approximately minimal cut of the desired accuracy, so we simply use Gabow's $O(m + c^2 n \log n)$ time algorithm to find it; this takes $O(m + n\log^3 n/\epsilon^4)$ time.

4 Parallel Algorithms

We now present parallel approximation algorithms. The approach is as before: generate a sparse skeleton and use a sparse-graph algorithm to quickly find a minimum cut in the skeleton. It is easy to construct a skeleton in parallel using m processors: simply assign one processor to each edge, and have it flip an appropriately biased coin to decide whether or not to include the edge. This method can be extended to weighted graphs using the techniques of [Kar93]. However, using the skeleton in approximation is somewhat more complicated, because there is no parallel version of Gabow's minimum cut algorithm. To solve this problem, we develop algorithms for approximating the minimum cut in a sparse graph. By applying this algorithm to the skeleton of a graph G, we find an approximately minimal cut in the skeleton. Theorem 2.1 shows that this approximately minimal cut corresponds to an approximately minimal cut of G.

As in the sequential case, constructing a skeleton of the proper density requires a constant factor estimate of the minimum cut. We can simulate this estimate by performing a binary search on the range of possible cut values. We decrease or increase our estimate as the cut approximation algorithm does or does not find a cut of roughly the estimated value. In the case of weighted graphs, we use a linear processor $\mathcal{NC}$ algorithm of [Kar93] to estimate the minimum cut to within a factor of n^2. This ensures that even in the weighted case, $O(\log n)$ iterations of the cut algorithms we present will suffice to home in on the actual cut value. Thus from now on we assume that a constant factor estimate of the cut value is given.

4.1 A $(2+\epsilon)$-Approximation.

As a first step, we give a $(2+\epsilon)$-approximation algorithm. Since we already know how to construct a skeleton, it is sufficient to find an approximation algorithm which works well on sparse graphs. We parallelize Matula's $(2+\epsilon)$-approximation algorithm. His algorithm is described in Figure 2.

1. Given graph G, compute its minimum degree d.
2. Compute a sparse $\left(\frac{d}{2+\epsilon}\right)$-connectivity certificate.
3. Contract all the non-certificate edges, and recursively find an approximate minimum cut in the contracted graph.
4. Return the smaller of d and the result of Step 3.

Figure 2: Matula's Algorithm

To see why this algorithm works, consider two cases. If the minimum cut exceeds $d/(2+\epsilon)$, then d is a $(2+\epsilon)$ approximation to the minimum cut. If the minimum cut is less than $d/(2+\epsilon)$, then the certificate will contain all the minimum cut edges. Thus contracting the non-certificate edges will preserve the minimum cut, so the smaller graph has the same minimum cut as the original and we can argue correctness by induction.

To understand the time bounds, observe that the original graph has at least $dn/2$ edges (since the minimum degree is d) while the sparse certificate contains only $dn/(1+\epsilon/2)$ edges. Thus Step 3 eliminates an $\Omega(\epsilon)$ fraction of the edges in the graph; therefore the depth of the recursion is $\tilde{O}(1/\epsilon)$ and the amount of work performed at each level of the recursion decreases geometrically.

Now observe that the central element of the algorithm is the call to a sparse certificate subroutine. Furthermore, it is easy to implement the other steps of the algorithm in $\mathcal{NC}$ using a linear number of processors. Thus to parallelize Matula's Algorithm we need only show how to find sparse certificate in parallel. This is where we take advantage of the minimum cut being small. We use the ($\mathcal{NC}$) sparse certificate algorithm of [CKT93]. Given an input parameter k, this algorithm uses m CRCW processors, runs in $O(k \log n)$ time, and finds a certificate of G which has at most kn edges but preserves all cuts of value less than or equal to k. Using this algorithm as the black box for Matula's algorithm allows us to deduce the following result:

THEOREM 4.1. *In a graph with minimum cut c, a $(2+\epsilon)$-minimal cut can be found in $\mathcal{NC}$ in $\tilde{O}(c/\epsilon)$ time using m processors.*

To extend this approach to arbitrary graphs, given ϵ, we construct a skeleton of $\tilde{O}(n/\epsilon^2)$ edges and minimum cut $\tilde{O}(\epsilon^{-2})$ which, by Theorem 2.1, approximates cuts to within a $(1+\epsilon/4)$ factor. We then run the $(2+\epsilon/2)$-approximation algorithm just mentioned. By Theorem 2.1, such a cut will correspond to a $(2+\epsilon/2)(1+\epsilon/4) \approx (2+\epsilon)$-minimal cut. We have shown:

THEOREM 4.2. *In any weighted undirected graph, a $(2+\epsilon)$-approximation to the minimum cut can be found in $\mathcal{RNC}$ in $\tilde{O}(\epsilon^{-3})$ time using a linear number of processors.*

5 Parallel $(1+\epsilon)$-Approximation

We now give an approximation algorithm with better bounds. The bottleneck in the approximation algorithm we just described is the use of Matula's algorithm on the skeleton, which forces the approximation factor to exceed 2. We therefore develop a $(1+\epsilon)$-approximation algorithm for the case where the minimum cut is small, and then use that instead of Matula's algorithm to approximate the minimum cut in the skeleton. As in the last section, this immediately yields a $(1+\epsilon)$-approximation algorithm for arbitrary cut values.

We proceed to modify the Recursive Contraction Algorithm (RCA) of [KS93] to take advantage of the combination of sparse certificates and a willingness to approximate. Recall that the RCA uses the more primitive Contraction Algorithm of [Kar93]. The Contraction Algorithm, denoted $CA(G,k)$, takes a graph G of n vertices and m edges, and using m processors in $\mathcal{RNC}$ returns a contraction of G to k vertices such that with probability $\Omega((k/n)^2)$, the contracted graph has the same minimum cut as the original.

There are two reasons the RCA requires n^2 processors. One derives from the Contraction Algorithm: since its success probability is $\Theta(n^{-2})$, it is necessary to perform $\Omega(n^2)$ trials to get a high success probability. This forces us to do $\Omega(n^2)$ work even after the RCA reduces the amount of work per trial to nearly a constant. The second reason derives from the RCA: as it performs recursive calls, it is impossible to bound the number of edges in the graphs which are its arguments—thus the best bound which can be put on the number of edges in an r vertex graph is r^2, and this forces us to allocate r^2 processors even to examine the input.

In order to reduce the processor cost, we have to solve both of the just mentioned problems. Randomized sparsification is used to solve the second problem, ensuring that the arguments in the recursive calls are sparse. However, in order to solve the problem of the success probability, it is necessary to modify the Contraction Algorithm, accepting less accuracy in the result in return for a higher probability of getting one.

5.1 Modifying the Contraction Algorithm.

We assume familiarity with [Kar93, KS93]. Since we are willing to settle for an approximation, we can halt the Contraction Algorithm as soon as we find a nearly optimal cut. This allows us to improve the analysis by making the assumption that no such small cut has yet been found. In particular, we assume that the minimum cut value c is known and apply the variant of the Contraction Algorithm in Figure 3.

```
repeat until two vertices remain or the average
    degree is less than αc
    choose an edge uniformly at random
    contract its endpoints
```

Figure 3: The Approximate Contraction Algorithm

This algorithm demonstrates that a willingness to approximate increases the success probability of the Contraction Algorithm, as we see in the following lemma:

LEMMA 5.1. *If an n vertex graph is contracted to k vertices, then with probability $\Omega((k/n)^{2/\alpha})$, the contraction will either preserve the minimum cut in the contracted graph or reveal an α-minimal cut.*

Proof. If the algorithm terminates because the average degree is less than αc, it means that some intermediate vertex has degree less than αc. This vertex in turn defines a cut of value less than αc. If the algorithm does not terminate for this reason, then it runs until two vertices remain. Following the analysis of [Kar93, Theorem 2.1] and using the fact that at all times the average degree exceeds αc, we deduce that the probability that the two vertices define a minimum cut is $\Omega(n^{-2/\alpha})$. Similarly, if we stop when k vertices remain, we deduce that the minimum cut survives with probability $\Omega((k/n)^{2/\alpha}))$.

A parallel version of the Contraction Algorithm is given in [Kar93]. To use the variant of Figure 3, we must slightly modify this parallel algorithm in order to keep track of the average degree during contraction, and stop if we encounter a vertex of sufficiently small degree. However, we defer the discussion of this change to Section 5.3 and first discuss the use of the modified algorithm in the final approximation algorithm.

5.2 Modifying the Recursive Algorithm. Having modified the Contraction Algorithm CA to increase its success probability, we proceed to modify the RCA to reduce the amount of work it does. Recall that the modified version of $CA(G, k)$ will either output a cut of value at most αc or ensure that the minimum cut survives the contraction with probability $\Omega(n^{-2/\alpha})$.

Figure 4 gives the modified algorithm RCA for finding an α-minimal cut in a graph with minimum cut k. The algorithm uses the previously mentioned (deterministic) sparse certificate algorithm of [CKT93]. That algorithm runs in $O(\alpha c)$ time and finds a sparse (αc)-connectivity certificate containing $O(\alpha c n)$ edges. Recall that any cut of value exceeding αc in G corresponds to a cut of value exceeding αc in the certificate.

```
Algorithm RCA(G, n)

input A graph G of size n.
if n = 2
then examine the implied cut of the original graph
else  Find an (αc)-connectivity certificate G' of G
      repeat twice
            G'' ← CA(G', n/2^{α/2})
            RCA(G'', n/2^{α/2})
```

Figure 4: The Modified Algorithm

As a consequence of Lemma 5.1, if we are looking for an α-minimal cut and contract an n-vertex graph to $n/2^{\alpha/2}$ vertices, we have a 50% chance of either finding an α-minimal cut or preserving the minimum cut of the original graph. Since in the algorithm we perform this experiment twice, we expect that one of the experiments will succeed. The remainder of the proof of correctness follows [KS93]. The necessary addition is to observe that the sparse certificate algorithm used here does not affect the minimum cut.

We now consider the processor bounds. Since we run the sparse certificate algorithm before calling RCA recursively, we can be sure by induction that at all levels of the recursion, whenever RCA is called with a graph of n vertices, that graph has $O(cn)$ edges. It follows that the calls to the sparse certificate algorithm and to CA will require only cn processors. This gives the following recurrence for the processor cost (details of its derivation can be found in[KS93]):

$$T(n) = 2(cn + T(n/2^{\alpha/2})).$$

This recurrence solves to $T(n) = \tilde{O}(cn^{2/\alpha})$. The recursion depth is $O(\log n)$, and the time spent at each level of the recursion is $\tilde{O}(c)$ (dominated by the sparse certificate construction). This gives the following:

Lemma 5.2. *An α-minimal cut of a graph with minimum cut c can be found in $\tilde{O}(c)$ time with high probability using $m + cn^{2/\alpha}$ processors.*

As in Section 4.1, we can use this sparse graph algorithm to approximate the minimum cut in an arbitrary graph by approximating the minimum cut in its sparse skeleton. This yields the following theorem:

Theorem 5.1. *For any constant α, an α-minimal cut of any graph can be found in $\mathcal{RNC}$ with high probability using $m + n^{2/\alpha}$ processors.*

Neither a sparse graph nor the willingness to approximate can in itself give a faster algorithm. Even if the graph is sparse, using the RCA to find the minimum cut exactly requires $\Omega(n^2)$ processors. On the other hand, if we are willing to approximate but fail to sparsify, the fact that we are recursively calling CA on dense graphs means that it may require $\Omega(n^2)$ processors (see [KS93] for details).

5.3 Tracking the Degree. The Contraction Algorithm is parallelized in [Kar93]. We modify this parallelization in order to keep track of the average degree; this in turn lets us implement the algorithm in Figure 3 in parallel. As in [Kar93], we simulate the contraction by generating a random permutation of the edges and then contracting edges in order of the permutation. As we consider the sequence of contractions induced by the permutation, let epoch i denote the period when $n - i$ vertices remain in the graph. Our goal is to determine the average degree at each epoch, which is equivalent to determining the number of edges which exist at each epoch. This in turn is easy if we determine, for each edge, the epoch until which it survives.

As a first step towards making this determination, we identify the *contracted edges*. These are the at most $n-2$ edges which are actually chosen for contraction, distinguished from the edges which disappear when their endpoints are merged by some other contraction. Given the edge permutation, a particular edge is contracted if and only if all the edges preceding it fail to connect its endpoints. If we assign ranks to the edges based on the permutation order, and construct a minimum spanning tree (MST) based on the ranks, then the MST edges are precisely the edges satisfying this property. The MST can be found in $\mathcal{NC}$ using m processors [AS87b, JM92, CL93]. Furthermore, the order of the MST edges in the permutation determines the order of contractions: the first MST edge causes the first contraction, and therefore initiates the first epoch. The second MST edge initiates the second epoch, and so on. Label each MST edge based on this order.

This labeling gives us the information we need to determine until which epoch each other edge survives. An edge e survives until epoch i if and only if the edges contracted before epoch i, namely the MST edges with labels less than i, fail to connect e's endpoints. Thus, the epoch in which e disappears is simply the label of the largest labeled edge on the MST path between the endpoints of e. We have thus reduced our problem to the following: given the minimum spanning tree, compute for each edge the largest edge label on the path between its endpoints. This problem (essentially that of minimum spanning tree verification) can be solved in $\mathcal{NC}$ using $O(m)$ processors [AS87a].

If we determine that in some epoch the average degree fell below αc, it is simple to perform the contraction up to that epoch ([Kar93]) and then find the minimum degree vertex in the partially contracted graph; this yields a cut of the desired value. We have therefore shown how to implement the Approximate Contraction Algorithm of Figure 3.

6 Parametric Sparsification

We now consider an alternative approach to sparsification. Rather than fixing a probability p and using the sparse graph $G(p)$ to estimate the minimum cut in G, we estimate the minimum cut in G by determining the value of p for which $G(p)$ is sparse. We then show how this technique can be applied to give parallel and dynamic algorithms for approximating the minimum cut value. We will be using a weaker version of Theorem 2.1 from [Kar93].

THEOREM 6.1. *For any constant $a > 1$, if each edge of a graph is removed with probability $p = n^{-2a/c}$, then the probability that the graph becomes disconnected is at least p^c and at most n^2p^c.*

This theorem can be seen as a variation on our previous randomized sparsification theorem. While that theorem stated that it was the small cuts which were most likely to be small under sparsification, this theorem says that it is the small cuts which are most likely to vanish entirely under (more intense) sparsification.

The theorem suggests the following idea for approximating minimum cuts using only connectivity tests. The theorem gives the disconnection probability as a function of c and can therefore be inverted to give c as a function of the disconnection probability. This lets us estimate c by causing random edge failures and observing the resulting disconnection probability.

More precisely, given the graph G, fix some $\epsilon < 1$ and identify the value p such that killing each edge with probability p causes the graph to become disconnected with probability $n^{-\epsilon}$. Then use the theorem to estimate the minimum cut as follows. The theorem says that

$$p^c < n^{-\epsilon} < n^2p^c.$$

Since we know all other quantities, we can solve for c to deduce

$$\epsilon \ln_{1/p} n < c < (2+\epsilon)\ln_{1/p} n$$

Thus if we take c to be the geometric mean of its two bounds, the error in approximation can be at most $\sqrt{1+2/\epsilon}$.

The difficulty in this approach is in determining the correct value of p to cause disconnection at the appropriate probability. Note that it is insufficient to simply try a small number of different possible values of p, since the fact that p is exponentiated by c means that a small variation in p can cause a tremendous change in the disconnection probability. The same problem makes binary search among p-values infeasible: c bits of accuracy would be needed. The problem becomes particularly challenging when we need to solve it in a dynamic fashion. Therefore we slightly reformulate Theorem 6.1. Given a weighted graph G, define $w(G)$ to be the minimum weight edge in a maximum spanning tree of G.

COROLLARY 6.1. *If each edge in a graph G with minimum cut c is assigned an edge weight chosen uniformly at random from the unit interval, then*

$$p^c < \Pr[w(G) < p] < n^2p^c.$$

Proof. The value $w(G)$ is just the largest weight such that if all edges of weight less than $w(G)$ are removed from G then G remains connected. Thus $\Pr[w(G) < p]$ is just the probability that G becomes disconnected if we remove all edge of weight p or less. However, assigning uniform edge weights and then deleting all edges of weight at most p is equivalent to deleting each edge with probability p, so Theorem 6.1 applies.

This allows the following variation on the minimum cut approximation scheme recently described. Given a graph G, perform $N = \tilde{O}(n^\epsilon)$ independent trials of assigning random edge weights and computing $w(G)$. Let p be the $(n^{-\epsilon}N)^{th}$ smallest $w(G)$ value encountered. Standard Chernoff bound arguments show that with high probability, p is such that $\Pr[w(G) < p] = \Theta(n^{-\epsilon})$. Now apply the bounds in the corollary to determine an approximation for c. In this version we only have a constant factor approximation to the "disconnection probability" $\Pr[w(G) < p]$ (*i.e.*, we know that $p^c < \Theta(n^{-\epsilon}) < n^2p^c$). However, since the computation of c involves taking logarithms this becomes and *additive* $o(1)$ error in the value of c. We thus have a $(\sqrt{1+2/\epsilon})$-approximation for the minimum cut value.

We can implement this scheme in parallel by using one of numerous linear processor maximum spanning tree algorithms to compute the n^ϵ values $w(G_i)$. This yields the following result:

THEOREM 6.2. *A $(\sqrt{1+2/\epsilon})$-approximation to the value of the minimum cut can be computed in $\mathcal{RNC}$ using mn^ϵ processors.*

7 Dynamic Approximation

Another power of parametric sparsification is that we can make it dynamic. Eppstein *et al* [EGIN92] give an algorithm for dynamically maintaining a maximum spanning tree of a graph in $\tilde{O}(\sqrt{n})$ time per edge insertion or deletion (coincidentally, their algorithm is also based on a graph sparsification technique). They give another algorithm which maintains a maximum spanning tree under insertions only in $\tilde{O}(1)$ time per insertion. We use this as follows. Given an unweighted graph G, we maintain $\tilde{O}(n^\epsilon)$ copies G_i of G, each with randomly assigned edge weights, and dynamically maintain the maximum spanning trees of the G_i. When an edge is added to our graph G, we add the edge to each G_i, assigning it an independently chosen random weight in each. When an edge is deleted, we delete it from each of the G_i. It is easy to modify the dynamic minimum-spanning tree algorithms to maintain the values $w(G_i)$ with no additional overhead. Thus after each update, we simply inspect the values $w(G_i)$ in order to estimate the minimum cut. By choosing constants appropriately, we can ensure a polynomially small probability that our analysis will fail at any particular step in the update sequence. It follows that over any polynomial length sequence of updates, we have a high probability of the analysis being correct at all points in the sequence. If we now plug in the time bounds for the dynamic maximum spanning tree algorithms, we deduce time bounds for dynamically approximating the minimum cut.

THEOREM 7.1. *The minimum cut of an unweighted graph can be dynamically approximated with high probability to within a $\sqrt{1+2/\epsilon}$ factor over any polynomial length sequence of edge insertions and deletions in $\tilde{O}(n^{\epsilon+1/2})$ time.*

THEOREM 7.2. *The minimum cut of an unweighted graph can be dynamically approximated with high probability to within a $\sqrt{1+2/\epsilon}$ factor over any polynomial length sequence of edge insertions in $\tilde{O}(n^\epsilon)$ time.*

It should be noted that in these dynamic algorithms an important but natural assumption is that the adversary determining the update sequence is not aware of any of the random bits used by the algorithm.

These algorithms generalize to weighted graphs. All that is necessary is to treat an edge of weight w as a set of w parallel edges. It is even possible to keep the running time from depending on the magnitudes of the weights: the uniformly generated random variables are replaced with exponential random variables whose parameters depend on the weights of the edges. The approach is similar to that found in [Kar93].

8 Conclusion

This paper has extended the work of [Kar93] which shows a strong connection between the minimum cut in a graph and its behavior under edge failures. We have demonstrated that random edge failures tend to "preserve" the minimum cut information of a graph. This yields a linear time sequential approximation algorithm for minimum cuts, as well as a parallel algorithm which uses few processors. By parametrizing the random edge failures, we have have developed efficient dynamic approximation algorithms for the problem, and demonstrated a intriguing connection between minimum cuts and random minimum spanning trees which may be of further interest. An important question is whether better approximation results can be achieved by combining our k-connected skeletons with the minimum spanning tree approach; we conjecture that an investigation of the threshold for k-connectivity in a randomly weighted graph will yield even better information than the (minimum spanning tree) threshold for 1-connectivity. In a similar vein, randomized sparsification means that dynamically approximating connectivity can be reduced to dynamically maintaining exact connectivity in $O(\log n)$-connected graphs.

References

[AKS87] M. Ajtai, J. Komlós, and E. Szemerédi. "Deterministic simulation in logspace". In *Proceedings of the 19^{th} Annual ACM Symposium on Theory of Computing*, pages 132–140, 1987.

[AS87a] Noga Alon and Baruch Schieber. "Optimal preprocessing for answering online product queries". Technical report, Tel Aviv University, 1987.

[AS87b] Baruch Awerbuch and Y. Shiloach. "New connectivity and MSF algorithms for shuffle-exchange network and PRAM". *IEEE Transactions on Computers*, 36(10):1258–1263, October 1987.

[Che52] 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.

[CKT93] Joseph Cheriyan, Ming Yang Kao, and Ramki Thurimella. "Scan-first search and sparse certificates: An improved parallel algorithm for k-vertex connectivity". *SIAM Journal on Computing*, 22(1):157–174, February 1993.

[CL93] Ka Wong Chong and Tak Wah Lam. "Connected components in $O(\log n \log\log n)$ time on the EREW PRAM". In *Proceedings of the 4^{th} ACM-SIAM Symposium on Discrete Algorithms*, January 1993.

[EGIN92] David Eppstein, Zvi Galil, Giusseppe F. Italiano, and Amnon 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, October 1992.

[Gab91] Harold 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.

[GG81] O. Gabber and Z. Galil. "Explicit construction of linear-sized superconcentrators". *Journal of Computer and System Sciences*, 22:407–420, 1981.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[JM92] Donald B. Johnson and Panagiotis Metaxas. "A parallel algorithm for computing minimum spanning trees. In *Proceedings of the 4^{th} Annual ACM-SIAM Symposium on Parallel Algorithms and Architectures*, pages 363–372, June 1992.

[Kar93] David 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, January 1993.

[KS93] David R. Karger and Clifford 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.

[Mat93] 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, January 1993.

[NI92a] Hiroshi Nagamochi and Toshihde Ibaraki. "Computing edge connectivity in multigraphs and capacitated graphs". *SIAM Journal of Discrete Mathematics*, 5(1):54–66, February 1992.

[NI92b] Hiroshi Nagamochi and Toshihde Ibaraki. "Linear time algorithms for finding k-edge connected and k-node connected spanning subgraphs. *Algorithmica*, 7:583–596, 1992.