Approximating s–t Minimum Cuts in $\tilde{O}(n^2)$ Time

András A. Benczúr*
Department of Mathematics
M. I. T.

David R. Karger†
Laboratory for Computer Science
M. I. T.

Abstract

We improve on random sampling techniques for approximately solving problems that involve cuts in graphs. We give a linear-time construction that transforms any graph on n vertices into an $O(n \log n)$-edge graph on the same vertices whose cuts have approximately the same value as the original graph's. In this new graph, for example, we can run the $\tilde{O}(mn)$-time maximum flow algorithm of Goldberg and Tarjan to find an s–t minimum cut in $\tilde{O}(n^2)$ time. This corresponds to a $(1+\epsilon)$-times minimum s–t cut in the original graph. In a similar way, we can approximate a sparsest cut in $\tilde{O}(n^2)$ time.

1 Introduction

Several papers by Karger [Kar94b, Kar94a, Kar96] have recently shown that random sampling is an effective tool for problems involving cuts in graphs. A *cut* is a partition of a graph's vertices into two groups; its *value* is the number, or in weighted graphs the total weight, of edges with one endpoint in each side of the cut. Many problems depend only on cut values: the connectivity of a graph is the minimum value of a graph cut; the maximum flow that can be routed from s to t is the value of the minimum cut separating s and t; a minimum bisection is the smallest cut that splits the graph into two equal-sized pieces.

Random sampling "preserves" the values of cuts in a graph. If we pick each edge of a graph G with probability p, we get a new graph in which every cut has expected value exactly p times it value in G. The following lemma from [Kar94a] shows that all cuts are near their expected values:

Lemma 1.1 *Let G have minimum cut c. Build G' by including every edge from G with probability p. If $p > 8(\log n)/\epsilon^2 c$ then with probability $1-1/n$ every cut in G' has value within $(1\pm\epsilon)$ of its expectation.*

Thus, for example, the minimum cut of G' is close to pc. Therefore, if we are willing to settle for an approximation, we can find the minimum cut in G', which has fewer edges than G, and from it derive an approximately minimum cut of G. This scheme can be used to approximate the minimum cut in $\tilde{O}(m)$ time [Kar94a].

Unfortunately, the requirement that $p \gg 1/c$ limits the effectiveness of this scheme. It means that in a graph with m edges, we can only reduce the number of edges to m/c. Thus, when c is small, we gain little. Results can be even worse in weighted graphs.

1.1 Our Results

In this paper, we show how *nonuniform sampling* can be used to remove random sampling's dependence on c. Our main result is the following:

Theorem 1.2 *Given a graph G and an error parameter ϵ, there is a graph G' such that*

- *G' has $O(n \log n/\epsilon^2)$ edges and*
- *the value of every cut in G' is $(1\pm\epsilon)$ times the value of the corresponding cut in G.*

G' can be constructed in $O(m \log^2 n)$ time if G is unweighted and in $O(m \log^3 n)$ time of G is weighted.

It follows that for any algorithm to (even approximately) solve a cut problem, if we are willing to accept an approximate answer, we can substitute $n \log n$ for any factor of m in the running time. Our applications of this result are the following:

Corollary 1.3 *In an undirected graph, a $(1+\epsilon)$ times minimum s–t cut can be found in $\tilde{O}(n^2/\epsilon^2)$ time.*

Corollary 1.4 *In an undirected graph, a $(1+\epsilon)$ times minimum s–t cut of value v can be found in $\tilde{O}(nv/\epsilon^2)$ time.*

Corollary 1.5 *An $O(\log n)$-approximation to the sparsest cut in an undirected graph can be found in $\tilde{O}(n^2/\epsilon^2)$ time.*

These corollaries follow by applying our sampling scheme to (respectively) the maximum flow algorithm of Goldberg and Tarjan [GT88], the classical augmenting-paths algorithm for maximum flow [FF56, AMO93], and the Klein-Stein-Tardos algorithm for approximating the sparsest cut [KST90].

---

*E-mail: benczur@theory.lcs.mit.edu. Address: Laboratory for Computer Science, 545 Technology Square, Cambridge MA 02139.

Research supported by DARPA contract N00014-92-J-1799 and NSF contract 9302476-CCR.

†M.I.T. Laboratory for Computer Science, 545 Technology Square, Cambridge MA 02139. E-mail: karger@theory.lcs.mit.edu.
URL: http://theory.lcs.mit.edu/~karger.

Research supported in part by ARPA contract N00014-95-1-1246.

1.2 Our Methods

Our approach modifies the uniform sampling scheme described in Lemma 1.1. That scheme sampled all graph edges with the same probability. Here we show that a graph with many edges has "dense regions" that can be sampled sparsely (with lower probability than the general graph), ensuring that not too many edges are sampled overall. Consider a c'-connected induced subgraph of G with $c' > c$. Lemma 1.1 says that we can sample the edges of this subgraph with probability $\tilde{O}(1/c')$ without introducing significant error in the cut values. More generally, we can sample an edge with probability inversely proportional to the connectivity of *any* subgraph containing that edge.

To take advantage of this fact, we will show that almost all the edges are in components with large connectivities and can therefore be sampled with low probability—the more edges, the less likely they are to be sampled. We can therefore construct an $O(n \log n)$-edge graph that, regardless of the minimum cut value, accurately approximates all cut values.

In addition to proving that such sampling works, we give fast algorithms for identifying the dense regions and determining the correct sampling probabilities from them. This involves an extension of the *sparse certificate* technique of Nagamochi and Ibaraki [NI92b].

Our approach works for unweighted and weighted graphs. Initially, we will focus our discussion on unweighted graphs. We extend our results for weighted graphs in Section 5.

Our ideas are formalized in the following definitions and theorems.

Strong Connectivity

We first formalize the notion of subgraphs with large connectivities.

Definition 1.6 *A graph is* k-connected *if the value of each cut in G is at least k.*

Definition 1.7 *A* k-strong component *is a maximal k-connected vertex-induced subgraph.*

It follows that the k-strong components partition the vertices of a graph and that the $(k+1)$-strong components are a refinement of the partition into k-strong components.

Definition 1.8 *The* strong connectivity *of an edge e, denoted c_e, is the maximum value of k such that a k-strong component contains e. We say e is* k-strong *if its strong connectivity is k or more, and* k-weak *otherwise.*

Note that the strong connectivity of an edge differs from the standard definition of connectivity:

Definition 1.9 *The* (standard) connectivity *of an edge e is the minimum value of a cut separating its endpoints.*

Consider the graph with edges (s, v_i) and (v_i, t) for $i = 1, \ldots, n$. Vertices s and t have (standard) connectivity n but only have strong connectivity 1. An edge's strong connectivity is always less than its connectivity since a cut of value less than k separating the endpoints of an edge implies that there is no k-connected component containing both endpoints.

Compression

As was stated above, we aim to sample edges with varying probabilities. This eliminates the linear-scaling behavior of the sampled cut values. To recover this behavior, we make the following definition.

Definition 1.10 *Given a graph G and* compression probabilities *p_e for each edge e, we build a* compressed graph $G[p_e]$ *by including edge e in $G[p_e]$ with probability p_e, and giving it weight $1/p_e$ if it is included.*

Since the expected weight of any edge in the graph is 1, every cut's expected value is equal to its original value.

The Main Theorems

We now use the above definitions to describe our results. In our theorems, we use a constant *compression factor* ρ. We choose the compression factor so as to satisfy a given error bound ϵ:

$$\rho_\epsilon = 16(d+2)(\ln n)/\epsilon^2 \ .$$

Theorem 1.11 (Compression) *Given ϵ and a corresponding ρ_ϵ, for each edge e, let $p_e = \min\{1, \rho/c_e\}$. Then with probability $1 - n^{-d}$, every cut in $G[p_e]$ has value between $(1-\epsilon)$ and $(1+\epsilon)$ times its value in G.*

Theorem 1.12 *If $p_e = \min\{1, \rho/c_e\}$ then with very high probability $G[p_e]$ has $O(n\rho)$ edges.*

In particular, to achieve any constant error in cuts values, one can choose ρ to yield $O(n \log n)$ edges in the compressed graph.

1.3 Previous results

In this section, we describe some ingredients of previous work that will be used in our result.

Uniform Sampling

We begin with the edge-sampling scheme of [Kar94a].

Definition 1.13 *For an unweighted graph G, let each edge e have a random variable X_e assigned to it. We obtain a random weighted graph $G(X_e)$ by putting weight X_e on edge e.*

Note the use of parenthesis versus brackets to distinguish general sampling from compression. Note also that the special case of assigning weight 0 to an edge corresponds to not putting the edge in $G(X_e)$ at all—this means $G(X_e)$ can have fewer edges than G.

Theorem 1.14 (Sampling [Kar94a]) *Let X_e be a collection of random variables for $e \in G$ such that $0 \leq X_e \leq M$ for all e. Let $\hat{c}$ be the minimum expected value of any cut in $G(X_e)$. With probability $1 - O(1/n^d)$, every cut in $G(X_e)$ has value between $(1+\epsilon)$ and $(1-\epsilon)$ times its expected value, where $\epsilon = \sqrt{2(d+2)(M/\hat{c})\ln n}$.*

What this Sampling Theorem essentially says is that the key to a good error bound is to have no one edge contribute a significant portion to the weight of any cut. As a particular example, suppose we take a graph with minimum cut c and sample every edge with probability $p = \Theta((\log n)/\epsilon^2 c)$. This corresponds to a variable

$$X_e = \begin{cases} 1 & \text{with probability } p, \\ 0 & \text{otherwise.} \end{cases}$$

We can plug these variables into the sampling theorem with $M = 1$ and $\hat{c} = pc = \Omega(\log n)$. Thus the sampled graph will approximate all original graph cuts to within $(1 \pm \epsilon)$ (in a scaled sense).

The Sampling Theorem is used in algorithms for approximating minimum cuts and maximum flows and for solving network design problems [Kar94b, Kar94a]. It also plays an important role in a near linear time algorithm for finding minimum cuts exactly [Kar96].

Sparse Certificates

The other tool we use is *sparse certificates*.

Definition 1.15 *A* sparse k-connectivity certificate, *or simply a* k-certificate, *for an n-vertex graph G is a subgraph H of G such that*

1. *H contains at most $k(n-1)$ edges, and*
2. *H contains all edges crossing cuts of value k or less.*

It follows from this definition that if a cut has value $v \le k$ in G, then it has the same value v in H since all edges from it must be in the certificate. On the other hand, any cut of value greater than k in G has value at least k in H. Therefore, if we are looking for cuts of value less than k in G, we might as well look for them in H, since they are the same. The advantage is that H may have many fewer edges than G.

Nagamochi and Ibaraki [NI92b, NI92a] used sparse certificates in an $O(mn)$-time algorithm for computing the minimum cut in a graph. They gave an algorithm [NI92b] that constructs a sparse k-connectivity certificate in $O(m)$ time on unweighted graphs, independent of k. They also gave a weighted-graph algorithm with an $O(m + n\log n)$ running time [NI92a]. Note that in weighted graphs, we define a sparse certificate as follows: we equate an edge of weight w with a set of w unweighted edges with the same endpoints—the bound on size becomes a bound on the total weight of certificate edges.

Limitations

We can find approximate minimum cuts efficiently by combining sparse certificates with the Sampling Theorem. Given that the minimum cut is c, we construct a sparse $O(c)$-connectivity certificate with the same minimum cuts but with $O(nc)$ edges. The edges of this graph can then be sampled with probability $\Theta((\log n)/c)$, yielding an $O(n \log n)$-edge graph with minimum cuts corresponding to approximately minimum cuts in the original graph. In this sample, minimum cuts can be found in $\tilde{O}(n)$ time using Gabow's algorithm [Gab95] for minimum cuts.

Suppose we try to use the above approach to find an s–t minimum cut with value v. By sampling a sparse $O(v)$-certificate, we can reduce the number of edges to $\tilde{O}(nv/c)$. We cannot take a sparser certificate without potentially damaging the cuts we aim to find, and we cannot decrease the sampling probability without substantially increasing the variance in cut values. Thus the number of edges, and hence any improved running time to approximate an s–t minimum cut, will depend on the ratio of v to c which could be arbitrarily large. Our introduction of nonuniform sampling allows us to overcome this limitation.

2 Definitions

In the bulk of this paper, G denotes an unweighted undirected graph with n vertices and m edges; parallel edges are allowed. We also consider weighted graphs. If running times are not relevant, we can treat an edge of weight w as a set of w parallel edges with the same endpoints.

If F is a weighted graph, $\text{wt}(e, F)$ denotes the weight of edge $e \in F$. For a constant γ, γF denotes a graph with all edge weights multiplied by γ. We extend this notion when r_e is a vector over the edge set and let $r_e F$ denote a graph with $\text{wt}(e, r_e F) = r_e \text{wt}(e, F)$.

A *cut* $\mathcal{C}$ is a partition of the vertices into two subsets. The *value* $\text{VAL}(\mathcal{C}, G)$ of the cut in (a weighted or unweighted) graph G is the total weight (or number) of edges with endpoints in different subsets. If it does not lead to ambiguity, we also let G denote the vector of all cut values of a graph G, ordered canonically by vertex partitions. We therefore say that $G \le F$ if G and F have the same vertex set and the value of each cut in G is not more than that of the corresponding cut in F. For a scalar or vector v, we say that $v \in (1 \pm \epsilon)v'$ if $(1-\epsilon)v' \le v \le (1+\epsilon)v'$.

We say that an event occurs with *high probability* if its probability is $1 - O(n^{-d})$ for some parameter d. An event occurs with *very high probability* if the probability it does not occur is exponentially small in n.

Using all of these definitions, the conclusion of the Compression Theorem can be described as follows: with high probability, $G[p_e] \in (1 \pm \epsilon)G$.

3 Proofs of the Compression Theorem

We now prove Theorems 1.11 and 1.12. Recall that we compress with probabilities $p_e = \rho/c_e$ where c_e is the strong connectivity of edge e. Compression is a special case of sampling. We can use Theorem 1.14, with $M = \max\{1/p_e \mid e \in G\}$, to prove an error bound, but it is extremely weak. Our main result will be a much tighter error bound based on connectivity information encoded in the p_e-values.

3.1 Bounding the number of edges

We prove Theorem 1.12 first. We bound the *expected* number of edges by $n\rho$ (where ρ is the compression factor); the high probability result follows by a straightforward Chernoff bound [Che52]. We use the following lemma:

Lemma 3.1 *A graph with total edge weight $k(n-1)$ or more has a k-strong component (which may be the graph itself).*

Proof: Let G be a smallest counterexample with n vertices. Since in particular G is not k-connected, it must have a cut $\mathcal{C}$ of value less than k. Let us remove the edges of $\mathcal{C}$ and consider the two sides G_1 and G_2 with n_1 and $n_2 = n - n_1$ vertices respectively. Since G is a smallest counterexample and G_1 is not k-strong, G_1 must have total edge weight less than $k(n_1 - 1)$. Similarly, G_2 has edge weight less than $k(n_2 - 1)$. Adding back the fewer than k edges of $\mathcal{C}$, we see that the total edge weight of G is strictly less than $k(n_1-1) + k(n_2-1) + k = k(n-1)$, a contradiction. □

We can now prove Theorem 1.12. The expected number of edges in $G[p_e]$ is $\sum_{e \in G} p_e$; since we took $p_e = \min\{\rho/c_e, 1\}$, this value is at most $\rho \sum_{e \in G} c_e^{-1}$. Let us build a graph G' by taking each edge e of G with weight c_e^{-1}. It suffices to show that this graph has total edge weight at most n. Assume this is not the case: then by Lemma 3.1, G' has a 1-strong component F'. Let F be the corresponding subgraph of G; let e' have $c_{e'}$ minimum over $e \in F$. By the definition of $c_{e'}$, F cannot be more than $c_{e'}$-connected. Hence there is a cut $\mathcal{C}$ of F with $\text{VAL}(\mathcal{C}, F) \leq c_{e'}$. Then on one hand, $\text{VAL}(\mathcal{C}, F') > 1$; on the other hand, by the minimality of $c_{e'}$,

$$\begin{aligned}
\text{VAL}(\mathcal{C}, F') &= \sum_{e \in \mathcal{C}} c_e^{-1} \\
&\leq \sum_{e \in \mathcal{C}} c_{e'}^{-1} \\
&= c_{e'}^{-1} \text{VAL}(\mathcal{C}, F) \\
&\leq c_{e'}^{-1} c_{e'} \\
&= 1,
\end{aligned}$$

a contradiction.

3.2 A first error bound

We now give an argument that shows a slightly weaker result than Theorem 1.11. Namely, with the compression factor ρ_ϵ as above, the error bound will be $O(\epsilon \log m)$, instead of ϵ as in Theorem 1.11. The actual proof in the next subsection follows the same lines but is slightly more involved.

The main proof idea is the following. For an edge e with strong connectivity c_e, there is a c_e-strong component H of G containing e. Any cut $\mathcal{C}$ containing e must cut H, inducing a cut $\mathcal{C}'$. Let us fix $p = \rho / c_e$. If we sample each edge of H with this probability and give the included edges weight $1/p$, Theorem 1.14 applied to H (with $M = 1/p$ and $\hat{c} = c_e$) shows that the total weight of edges of $\mathcal{C}'$ changes by at most ϵ.

The problem with this approach is that the Sampling Theorem applied to H assumes that all edges in H are being sampled with the same probability ρ / c_e. Since some of the edges in H may actually be in a c'-strong component with $c' \gg c_e$, this assumption could be violated.

To get around this problem, we divide the sampling process into a series of phases: first we flip sampling coins for edges e with $c_e < 2$, then for $2 \leq c_e < 4$, etc. Clearly this division into phases does not affect the eventual outcome since we flip exactly one coin for each edge. However, in each phase all the coin flips have roughly the same bias so that the Sampling Theorem applies.

Let us decompose G into graphs graphs G_i for $i \geq 0$, such that $e \in G_i$ if $2^i \leq c_e < 2^{i+1}$. Graph G_i contains the edges whose coins we flip in the i-th phase. Let $F_i = \bigcup_{j \geq i} G_j$; then F_i is the set of all 2^i-strong components. In phase i, we flip coins for the edges of G_i while the other edges of G are left alone. To see how this affects the cuts in G, consider each 2^i-strong component H in G independently. The edges of $H \cap G_i$ are sampled with probability ρ / c_e. The edges of $H - G_i$ are left alone—equivalently, we can think of sampling them with probability 1. Formally, we can define random variables $X_e^{(i)}$ to the edges e by taking $p_e = \rho / c_e$ and setting

$$X_e^{(i)} = \begin{cases} \left\{ \begin{array}{ll} 1/p_e & \text{with probability } p_e \\ 0 & \text{otherwise} \end{array} \right\} & \text{if } e \in G_i \\ 1 & \text{if } e \notin G_i \,. \end{cases}$$

Now consider $H(X_e^{(i)})$. Theorem 1.14 applies with $\hat{c} \geq 2^i$ and $M = 2^{i+1} / \rho_\epsilon$. Thus with high probability

$$H(X_e^{(i)}) \in (1 \pm \epsilon/2) H.$$

By combining the disjoint 2^i-strong components that make up F_i, we deduce that $F_i(X_e^{(i)}) \in (1 \pm \epsilon/2) F_i$. Notice the importance of defining G_i such that all c_e in it are within a constant ratio: we must keep $\hat{c}$ and M near each other in Theorem 1.14.

The total error we incur during compression is bounded by the sum of all individual ϵ-error terms over all the coin-flipping phases. Since the maximum possible strong connectivity of any edge is m, there are at most $\log m$ phases for a total error of $\epsilon \log m$. More formally, we have:

$$\begin{aligned}
G[p_e] &= \sum_{i \leq \log m} G_i[p_e] && (1) \\
&= \sum G_i(X_e^{(i)}) \\
&= \sum (F_i(X_e^{(i)}) - F_{i+1}) \\
&= \sum F_i(X_e^{(i)}) - \sum F_{i+1} \\
&\in \sum (1 \pm \epsilon/2) F_i - \sum F_{i+1} && (2) \\
&= \sum (1 \pm \epsilon/2) \cdot (G_i \cup F_{i+1}) - \sum F_{i+1} && (3) \\
&= \sum (1 \pm \epsilon/2) G_i + \sum (1 \pm \epsilon/2) F_{i+1} \\
&\quad - \sum F_{i+1} && (4) \\
&= (1 \pm \epsilon/2) G \pm \epsilon/2 \sum F_{i+1} \\
&\in (1 \pm \epsilon/2) G \pm \epsilon/2 \sum_{i \leq \ell} G && (5) \\
&= (1 \pm \frac{\epsilon}{2} \log m) G && (6)
\end{aligned}$$

Here (1) follows by the decomposition of G into G_i, (2) by Theorem 1.14, and (3–4) by $F_i = G_i \cup F_{i+1}$. (5) follows by upper bounding the cut values in F_j by those in $G \supset F_j$. (6) follows since there are $O(\log m)$ terms in the sum.

3.3 An improved error bound

The above argument introduces an extra $\log m$ factor in the error bound; our next analysis improves on this bound and thus lets us get by with fewer sampled edges. We use the same ideas as in the previous subsection. We let G_i and F_i be as before, and bound the relative error arising by sampling from G_i separately for all values of i.

Our improvement over the previous proof is the following. There when we flipped coins for G_i, we added the edges of F_{i+1} so that we could use Theorem 1.14. The reason F_{i+1} had to be added was to embed the edges of G_i into 2^i-strong components; while F_i has strong components, $G_i = F_i - F_{i+1}$ might not. However, we show below that we can add a much lighter graph to G_i to make it 2^i-connected. Since the approximation error in a phase is proportional to the values of the cuts in that phase, adding a graph with lighter cuts will reduce the approximation error. We define this lighter graph as follows. For $e \in G_j$ and $i \leq j$, define $r_e^{(i)} = 2^{-(j-i)}$ and use $r_e^{(i)} F_{i+1}$ as the "deterministic part" added to G_i to make the Sampling Theorem work (recall that $r_e F$ denotes the graph F with the weight of edge e multiplied by r_e) The main observation is:

Lemma 3.2 *Let H be a 2^i-strong component of G. Then the subgraph $r_e^{(i)} H$ of $r_e^{(i)} F_i$ is 2^i-strong as well.*

Proof: We use downward induction on i. The claim holds for the largest value of i, when F_{i+1} is empty, since then $r_e^{(i)}F_i = F_i$. In the inductive step, let H be a 2^i-strong component. If a cut $\mathcal{C}$ of H has all of its edges in G_i, clearly $\text{VAL}(\mathcal{C}, r_e^{(i)}H) = \text{VAL}(\mathcal{C}, G_i) \geq 2^i$. In general, however, $\mathcal{C}$ may contain edges of F_{i+1} and thus of a 2^{i+1}-strong component H'. But then $\mathcal{C}$ cuts H'; therefore by induction, $\text{VAL}(\mathcal{C}, r_e^{(i+1)}H') \geq 2^{i+1}$. Hence

$$\begin{aligned}
\text{VAL}(\mathcal{C}, r_e^{(i)}F_i) &\geq \text{VAL}(\mathcal{C}, r_e^{(i)}F_{i+1}) \\
&\geq \text{VAL}(\mathcal{C}, \tfrac{1}{2}r_e^{(i+1)}F_{i+1}) \\
&\geq \tfrac{1}{2}\text{VAL}(\mathcal{C}, r_e^{(i+1)}H') \\
&\geq \tfrac{1}{2}2^{i+1}.
\end{aligned}$$

□

We now proceed to prove Theorem 1.11 as in the previous subsection. For each fixed i, define

$$X_e^{(i)} = \begin{cases} \left\{ \begin{array}{ll} 1/p_e & \text{with probability } p_e \\ 0 & \text{otherwise} \end{array} \right\} & \text{if } e \in G_i \\ r_e^{(i)} & \text{if } e \notin G_i \end{cases}$$

Observe that $G_i = r_e^{(i)}G_i = r_e^{(i)}F_i - r_e^{(i)}F_{i+1}$ while $G_i[p_e] = F_i(X_e^{(i)}) - r_e^{(i)}F_{i+1}$.

Lemma 3.3 *With high probability,* $F_i(X_e^{(i)}) \in (1 \pm \epsilon/2)r_e^{(i)}F_i$.

Proof: Let H be a 2^i-strong component of F_i. Let us apply Theorem 1.14 for $H(X_e^{(i)})$. We get $M \leq 2^{i+1}/\rho_e$ and $\hat{c} \geq 2^i$ by Lemma 3.2. Plugging in we find that the error bound is $\sqrt{2(d+2)(M/\hat{c})\ln n} = \epsilon/2$. The claim for F_i follows by summing cut values over all 2^i-strong components. □

Now the proof is completed by the same method as the previous subsection, by using Lemma 3.3 with the new random variables $X_e^{(i)}$:

$$\begin{aligned}
G[p_e] &= \sum_{i \leq \log m} G_i[p_e] \\
&= \sum F_i(X_e^{(i)}) - \sum r_e^{(i)}F_{i+1} \\
&\in \sum (1 \pm \epsilon/2)r_e^{(i)}F_i - \sum r_e^{(i)}F_{i+1} \\
&= \sum (1 \pm \epsilon/2) \cdot (r_e^{(i)}G_i \cup r_e^{(i)}F_{i+1}) \\
&\quad - \sum r_e^{(i)}F_{i+1} \\
&= \sum (1 \pm \epsilon/2)G_i + \sum (1 \pm \epsilon/2)r_e^{(i)}F_{i+1} \\
&\quad - \sum r_e^{(i)}F_{i+1} \\
&= (1 \pm \epsilon/2)G \pm (\epsilon/2)\sum r_e^{(i)}F_{i+1} \\
&= (1 \pm \epsilon/2)G \pm (\epsilon/2)\sum_{j \geq 1} G_j \sum_{1 \leq i < j} r_e^{(i)} \qquad (7) \\
&\in (1 \pm \epsilon)G \qquad (8)
\end{aligned}$$

Here Equation 7 follows by noting that $F_j = \sum_{i \geq j} G_i$ and then exchanging the order of the summation. Equation 8 follows by noting that $r_e^{(i)} = 2^{-(j-i)}$ for $e \in G_j$, implying

$$\sum_{1 \leq i < j} r_e^{(i)} = \sum_{1 \leq i < j} 2^{-(j-i)} < \frac{1}{2}.$$

This completes the proof of Theorem 1.11.

4 Finding strong connectivities

We could efficiently compress graphs if we could efficiently find the strong connectivities of edges. To be able to use the Compression Theorem, we do not even require the exact values c_e. For example, if we have estimates $\tilde{c}_e = \Theta(c_e)$, the Compression Theorem remains true with a constant-factor change in the expected number of edges and the error bound.

Unfortunately, even to quickly approximate strong connectivities is an open question. However, we now show that it is possible to find reasonably good lower bounds $\tilde{c}_e \leq c_e$. If we set $p_e = \rho/\tilde{c}_e \geq \rho/c_e$ in the Compression Theorem we only reduce the variances of our edge variables and therefore get no worse an error bound than with the correct c_e values. In this modified setting, we can no longer use Theorem 1.12: we must re-prove that the expected number of edges in the compressed graph $G[p_e]$ is $O(n\rho)$. A Chernoff bound then proves that the bound holds with very high probability.

Our basic plan is the following. Lemma 3.1 says that any graph with $k(n-1)$ or more edges has a k-strong component. It follows that at most $k(n-1)$ edges are k-weak (that is, have strong connectivity less than k). For otherwise the subgraph consisting of those edges would have a k-strong component, a contradiction. For each value $k = 1, 2, 4, 8, \ldots, m$, we will find a set of $k(n-1)$ edges containing all the k-weak edges (note that every edge is m-weak). We set $\tilde{c}_e = k$ for all edges in this set, thus establishing lower bounds for which the Compression Theorem works. The expected number of edges sampled under this basic scheme would be

$$\sum_{i=0}^{\log m} 2^i(n-1)(\rho/2^i) = O(\rho n \log m).$$

We will eventually describe a more sophisticated scheme that eliminates the factor of $\log m$.

Our first goal is to identify the k-weak edges. An obvious tool is a *sparse connectivity certificate*. Recall that this certificate contains $k(n-1)$ edges that include all edges with (standard) connectivity less than k—that is, all edges that cross a cut of value less than k. Nagamochi and Ibaraki [NI92b] give an algorithm `Certificate` that finds sparse certificates in $O(m)$ time on unweighted graphs and $O(m + n\log n)$ time on weighted graphs.

Unfortunately, although a sparse k-certificate contains all edges with *standard* connectivity less than k, it need not contain all edges with *strong* connectivity less than k, since some such edges might not cross any cut of value less than k. We must therefore perform some extra work. In Figure 1 we give an algorithm `WeakEdges` for identifying edges with $c_e < k$. It uses the Nagamochi-Ibaraki `Certificate` algorithm as a subroutine.

Theorem 4.1 `WeakEdges` *outputs a set containing all the* k*-weak edges of* G.

```
procedure WeakEdges(G, k)
  do log_2 n times
    E' ← Certificate(G, 2k)
    output E'
    G ← G − E'
  end do
```

Figure 1: Procedure `WeakEdges` for identifying $c_e < k$

Proof: First suppose that G has no k-strong components, i.e. that $c_e < k$ for all edges. Then by Lemma 3.1, there are at most $k(n-1)$ edges in G; hence at least half of the vertices have at most $2k$ incident edges (which define a cut of value at most $2k$). In a iteration of the loop in `WeakEdges`, these vertices become isolated after removing the sparse certificate edges. We have shown that in a single loop iteration half of the non-isolated vertices of G become isolated; hence in $\log_2 n$ rounds we isolate *all* vertices of G. Thus all the edges of G are output by `WeakEdges`.

In the general case, let us obtain a new graph H by contracting each k-strong component of G to a vertex. Any sparse $2k$-certificate of G contains the edges of a sparse $2k$-certificate of H as well. Thus by the previous paragraph, all edges of H are output by `WeakEdges`. But these are all the k-weak edges of G. □

4.1 Sparse partitions

The above algorithm can clearly be implemented via $O(\log n)$ calls to the Nagamochi-Ibaraki `Certificate` algorithm. It follows that it runs in $O(m \log n)$ time on unweighted graphs and outputs a set of at most $k(n-1)\log n$ edges. In this section, we eliminate the $\log n$ factor in this approach by finding edge sets that are "sparser" than the Nagamochi–Ibaraki certificate.

The first observation we use is that a given k-certificate E' may contain edges that are inside a connected component of $G - E'$. The edges in $G - E'$ do not cross a cut of value at most k (since they are connected by a path of edges that do not cross such a cut), so the same holds for any edge of E' whose endpoints are connected by a path in $G - E'$. We can therefore remove any such edge from E' and put it back in G.

We can find the resulting reduced edge set by contracting all edges not in E', yielding a new graph G'. But now observe that any edge crossing a cut of value at most k in G also crosses such a cut in G' since we contract no edge that crosses a small cut. Thus we can find all such edges via a certificate in G'. Since G' has fewer vertices, the certificate has fewer edges. We can iterate this procedure until all edges in the certificate cross some cut of value at most k or until G' becomes a single vertex. In the latter case, the original graph is k-connected, while in the former, if the current contracted graph has n' vertices, it has at most $k(n'-1)$ edges. This motivates the following definition:

Definition 4.2 *A* sparse k-partition*, or* k-partition*, is a set E' of edges of G such that*

1. *E' contains all edges crossing cuts of value k or less in G, and*
2. *If $G - E'$ has r connected components, then E' contains $2k(r-1)$ edges.*

In fact, the construction just described yields a graph with at most $k(r-1)$ edges, but we have relaxed the definition to $O(k(r-1))$ edges to allow for an efficient construction.

Procedure `Partition` in Figure 2 outputs a sparse partition. It uses the Nagamochi–Ibaraki `Certificate` algorithm and obtains a new graph G' by contracting those edges not in the certificate. It repeats this process until the graph is sufficiently sparse.

```
procedure Partition(G, k)

input: An n-vertex m-edge graph G

  if m/n < 2k then
      output the edges of G
  else
      E' ← Certificate(G, k)
      G' ← contract all edges of G − E'
      Partition(G', k)
```

Figure 2: `Partition` finds low-connectivity edges

Theorem 4.3 `Partition` *outputs a sparse partition in $O(m)$ time on unweighted graphs and $O(m \log n)$ time on weighted graphs.*

Proof: Correctness is clear since no weak edge is ever contracted; we need only bound the running time. Note that at a single recursion level the work done is just that of the call to `Certificate` which takes linear time [NI92b]. Now suppose $m/n > 2k$. We find a sparse connectivity certificate with $m' \le kn$ edges and then contract the graph to n' vertices. If $n' > n/2$ then in the following iteration we will have $m'/n' < (kn)/(n/2) \le 2k$ and the algorithm will terminate. It follows that the number of vertices halves in every recursive call except the last.

Suppose G is unweighted. At each recursive call, the number of edges is at most k times the number of vertices—thus the number of edges halves in each recursive call. It follows that $T(m,n) = O(m) + T(m/2, n/2) = O(m)$.

If G is weighted, we must use the $O(m + n\log n)$-time algorithm of [NI92a] to find sparse certificates. Furthermore, we cannot ensure that the number of edges halves, but only that the total weight of edges halves. However, the vertex-reduction argument still applies. Therefore, $T(m,n) = O(m + n\log n) + T(m, n/2) = O(m \log n)$. □

Lemma 4.4 *If* `Partition` *is used instead of* `Certificate` *in a call to* `WeakEdges`(G, k)*, then algorithm* `WeakEdges` *runs in $O(m \log n)$ time on unweighted graphs and $O(m \log^2 n)$ time on weighted graphs and returns a set of at most $4k(r-1)$ edges that partitions G into r connected components.*

Proof: The running time is clear from the previous lemma. To prove the edge bound, consider a particular connected component remaining in a particular iteration of `WeakEdges`. A call to `Partition` returns a set of $4k(s-1)$ edges that breaks that component into s subcomponents (the multiplier 4 arises from the fact that we look for a $2k$-partition). That is, it uses at most $4k(s-1)$ edges to increase the number of connected components by $s-1$. We can therefore charge $4k$ edges to each of the new components that gets created. Accumulating these charges over all the calls to `Partition` shows that if

WeakEdges outputs $4k(r-1)$ edges then those edges must split the entire graph into at least r components. □

4.2 Assigning Estimates

We now give an algorithm Estimation for estimating strong connectivities in Figure 3. We use subroutine WeakEdges to find a small edge set containing all edges e with $c_e < k$ but replace the Nagamochi-Ibaraki Certificate implementation with our algorithm Partition to reduce the number of output edges.

procedure Estimation(H, k)

input: subgraph H of G

if H contains edges
  $E' \leftarrow$ WeakEdges$(H, 2k)$
  for each $e \in E'$
    $\tilde{c}_e \leftarrow k$
  for each connected component $H' \subset H - E'$
    Estimation$(H', 2k)$

Figure 3: Procedure Estimation for assigning $\tilde{c}_e$-values

Lemma 4.5 *After a call to* Estimation$(G, 1)$*, the labels* $\tilde{c}_e$ *satisfy* $\tilde{c}_e \le c_e$.

Proof: Consider any recursive call Estimation$(H', 2k)$. This call occurred because H' was one component produced by WeakEdges$(H, 2k)$. By the correctness of WeakEdges, every edge in H' is $2k$-strong in H, and therefore is certainly $2k$-strong in G. It is therefore legitimate to set $\tilde{c}_e = 2k \le c_e$ for any $e \in H'$. Correctness of Estimation follows by induction. □

Lemma 4.6 *Assume that in procedure* WeakEdges*, procedure* Certificate *is replaced by* Partition*. Then the values* $\tilde{c}_e$ *output by* Estimation$(G, 1)$ *are such that the expected number of edges in* $G[\rho/\tilde{c}_e]$ *is* $O(n \log n)$.

Proof: We prove the following by induction on n: regardless of k, for any n-vertex graph G, Estimation(G, k) assigns values $\tilde{c}_e$ such that $\sum 1/\tilde{c}_e \le 4(n-1)$. The expected number of edges then follows as in Theorem 1.12.

The base case of a single vertex is trivial. For the inductive step, consider a call to Estimation(G, k). We first call WeakEdges(G, k), which returns a set of $4k(r-1))$ edges that partition G into r connected components $G_1, \ldots, G_r$ of sizes $n_1, \ldots, n_r$. Let us assume without loss of generality that $r > 1$, since otherwise Estimation simply recurses on the same graph without assigning any $\tilde{c}_e$ values until this is so. We now recursively call Estimation on the graphs G_i. By induction, these calls assign values $\tilde{c}_e$ to the edges of G_i such that

$$\sum_{e \in G_i} 1/\tilde{c}_e \le 4(n_i - 1)$$

Meanwhile, we assign value $\tilde{c}_e = k$ to each of the at most $4k(r-1)$ edges not in one of the G_i. It follows that the assignment of $\tilde{c}_e$ values satisfies

$$\begin{aligned} \sum_e 1/\tilde{c}_e &\le \sum_i 4(n_i - 1) + (4k(r-1))/k \\ &= (4\sum_i n_i) - 4r + 4(r-1) \\ &= 4(n-1) \end{aligned}$$

□

In summary, we have given the necessary construction for approximating strong connectivities:

Lemma 4.7 Estimation *determines values* $\tilde{c}_e \le c_e$ *such that* $G[\rho_\epsilon/\tilde{c}_e]$ *has* $O(n \log n / \epsilon^2)$ *edges and approximates all cut values to within* $(1 \pm \epsilon)$ *with high probability.*

Lemma 4.8 Estimation *runs in* $O(m \log^2 n)$ *time on an unweighted graph and in* $O(m \log^2 n \log nW)$ *time on a graph with maximum edge weight* W.

Proof: Each level of recursion of Estimation calls subroutine WeakEdges on graphs of total size m. An unweighted graph has maximum strong connectivity m and therefore has $O(\log m)$ levels of recursion; in a weighted graph the number of levels of recursion is $O(\log nW)$. □

5 Changes in weighted graphs

Now we describe how to extend our results to weighted graphs. The main theorems (Theorems 1.11 and 1.12) hold for multigraphs (graphs with parallel edges). Hence we can apply these theorems to weighted graphs by scaling all weights up, rounding them to integers, and replacing an edge of weight w be w parallel copies of the same edge. Notice that the error-bounding theorems are independent of the number of edges in G (which may become huge in this process).

On the algorithmic side, things are less simple. First of all, the simple m-step procedure of selecting each edge with probability p_e is no longer polynomial in n. This minor problem is solved by several linear-time algorithms for sampling weighted graphs [Kar94a]—for example, one can sample from the binomial (or Poisson) distribution that each edge of the weighted graph induces when each unit of weight is thought of as an individual edge.

The main difficulty in weighted graphs is in the estimation procedure for c_e. If the total edge weight M is polynomial, the solution is straightforward. Nagamochi and Ibaraki give an $O(m + n \log n)$-time weighted-graph implementation of their Certificate algorithm [NI92a] that we can use to implement Estimation in $O(m \log n)$ time. However, the $\tilde{c}_e$ values may now be as large as M, meaning that $O(\log M)$ levels of recursion will be required in Partition. To deal with this problem, we show how to localize our computation of strong connectivities to a small "window" of relevant connectivity values.

Lemma 5.1 *Suppose we contract all edges of* G *with weight exceeding* w^+ *and delete all edges with weight less than* w^-/n^3 *to get a new graph* G'. *Suppose* e *has strong connectivity* c_e *in* G*, with* $w^- < c_e < w^+$. *Then its strong connectivity* c'_e *in* G' *satisfies* $(1 - 1/n)c_e \le c'_e \le c_e$.

Proof: We first prove $c'_e \le c_e$. Suppose that after we contract the specified edge, we find edge e in a component H' with connectivity $c'_e > c_e$. Consider the preimage H of that component in G—it must have connectivity at most c_e. The claim is that contracting all edges of weight exceeding w^+ in H yields H' with connectivity c'_e. Since H has connectivity at most c_e,

procedure `WindowEstimation`(G)

> Sort the edges in decreasing weight order
> initialize G' as an empty graph
> **repeat**
> $W \leftarrow$ maximum remaining edge weight
> add every edge of weight greater than W/n^5 to G'
> Call `Estimation`(G')
> Contract every G' edge of weight exceeding W/n
> **until** no edges remain

Figure 4: `WindowEstimation` for weighted graphs

there must be a cut of value at most $c_e < w^+$ in H. Since only edges of weight exceeding w^+ are contracted, no edge across the small cut is contracted, so that H' still has connectivity at most c_e—a contradiction. Therefore, $c'_e < c_e$.

To prove that $c'_e > c_e(1 - 1/n)$, just note that the total weight of deleted edges is at most $\binom{n}{2} w^-/n^3 < c_e/n$. □

We now apply the above "windowing" lemma in our algorithms. We begin with a simple rough estimate for the strong connectivity values. Suppose we construct a maximum spanning tree (MST) for G. Let d_e be the minimum weight of an edge on the MST-path between the endpoints of e. The quantities d_e can be determined in linear time using an MST *verification* algorithm [DRT92]. Clearly, $c_e \geq d_e$. However, if we remove all edges of weight d_e or greater, then we disconnect the endpoints of e. There are at most $\binom{n}{2}$ such edges, so the weight removed is at most $n^2 d_e$. Therefore, $c_e \leq n^2 d_e$. This gives us an initial factor of n^2 estimate $d_e \leq c_e \leq n^2 d_e$.

For convenience we now delete every edge e whose weight is less than d_e/n^3. Since the minimum cut separating the endpoints of e is at least d_e, this deletion can change the value of any cut by a relative factor of at most $1/n$—negligible in our approximation algorithms. So we can assume that every edge e has weight at least $d_e/n^2 \geq c_e/n^4$.

Now assume that the maximum edge weight in G is W, and that our goal is to find the strong connectivity of every edge e with $c_e > W/n$. If we consider the subgraph of G made up only of edges with weight at least W/n^5, we do not significantly change any of the strong connectivities we care about. Thanks to the deletions of the previous paragraph, every edge e with $c_e > W/n$ will have weight at least W/n^5 and will therefore be present in our subgraph. In the new graph, we can run algorithm `Estimation` with a starting $\tilde{c}_e$ value of W/n and therefore finish in $O(m \log^2 n)$ time. This will assign correct approximate values $\tilde{c}_e$ to any edge e with $c_e \geq W/n$.

Given that all strong connectivities exceeding W/n have now been correctly assigned, we now contract all edges with weight exceeding W/n. This gives us a new graph whose maximum edge weight is W/n. We can repeat the previous paragraph's algorithm in the new graph, then contract more edges. Each time, we reduce the maximum edge weight by a factor of n. Eventually, we will have contracted all the graph edges and will have assigned all the $\tilde{c}_e$ values. This approach is detailed in our algorithm `WindowEstimation` in Figure 4.

Lemma 5.2 *Procedure* `WindowEstimation` *can be implemented to run in $O(m \log^2 n)$ time.*

Proof: The contractions in `WindowEstimation` can be implemented using a standard union-find data structure [CLR90]. Each time an edge is contracted, a union is called on its endpoints. Each time an edge is added, find operations can identify its endpoints. Therefore, the additions and contractions of edges do not affect the running time. Instead, the running time is determined by the repeated calls to `Estimation`. Note, however, that the "window" of considered edges ranges from W/n^5 to W, and that each iteration of the loop reduces W by a factor of n. Therefore, every edge is in the window at most 5 times. It follows that the total size of graphs passed to `Estimation` is $O(m)$. The claimed time bound follows. □

6 Applications

We now prove the application corollaries in the introduction.

6.1 Minimum s–t cuts.

Let us fix a pair of vertices s and t. Let $\hat{v}$ be the value of a minimum cut separating s from t in the compressed graph $G[p_e]$. We show that the minimum s–t cut value v in G is within $(1 \pm 3\epsilon)\hat{v}$. By Theorem 1.11, with high probability the s–t minimum cut $\mathcal{C}$ in G has has $\mathrm{VAL}(\mathcal{C}, G[p_e]) \leq (1+\epsilon)v$. Thus $\hat{v} \leq (1+\epsilon)v$. Furthermore, with high probability every cut of G with value exceeding $(1+3\epsilon)v$ in G will have value at least $(1-\epsilon)(1+3\epsilon) \geq (1+\epsilon)v$ in $G[p_e]$ and therefore will not be the minimum cut of $G[p_e]$.

We can find an approximate value $\hat{v}$ of the minimum s–t cut (and an s–t cut with this value) by computing a maximum flow in the $O(n \log n/\epsilon^2)$-edge graph $G[p_e]$. The maximum flow algorithm of Goldberg and Tarjan [GT88] has a running time of $O(nm \log(n^2/m))$ which decreases to a running time of $O(n^2 \log^2 n/\epsilon^2)$ after compression. Alternatively, the classical augmenting path algorithm [FF56, AMO93] for finding a flow of value v can be used to find an s–t cut of value at most $(1+\epsilon)v$ in time $O(nv \log n/\epsilon^2)$.

6.2 Sparsest cuts

A *sparsest cut* $(C|V-C)$ of a graph G minimizes the ratio

$$\frac{\mathrm{VAL}(C \mid V-C)}{|C||V-C|}.$$

It is $\mathcal{NP}$-hard to find the value of a sparsest cut. To find an α-approximate value of a sparsest cut, we use the approach of the previous subsection: we compute a β-approximate sparsest cut in the compressed graph $G[p_e]$. This cut is then an $\alpha = (1+\epsilon)\beta$-approximate sparsest cut of G.

An algorithm of Klein, Stein and Tardos [KST90] finds an $O(\log n)$-approximation to a sparsest cut in $O(m^2 \log m)$ time. By running their algorithm on $G[p_e]$, we will find an $O(\log n)$-approximate sparsest cut in $O(n^2 \log^3 n/\epsilon^4)$ time.

7 Conclusion

We have given new, stronger applications of random sampling to problems involving cuts in graphs. The natural open question is whether these approximation algorithms can be made exact. A partial affirmative answer was given in [Kar94a], but it only gives a useful speedup for graphs with large minimum cuts. More recently, sampling has led to an exact linear-time algorithm for minimum cuts [Kar96]; however, the techniques used there appear to be specialized to that particular problem.

References

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[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.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[DRT92] Brandon Dixon, Monika Rauch, and Robert E. Tarjan. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM Journal on Computing*, 21(6):1184–1192, 1992.

[FF56] Lester R. Ford, Jr. and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[Gab95] Harold N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. *Journal of Computer and System Sciences*, 50(2):259–273, April 1995. A preliminary version appeared in STOC 1991.

[GT88] Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum flow problem. *Journal of the ACM*, 35:921–940, 1988.

[Kar94a] David R. Karger. Random sampling in cut, flow, and network design problems. In *Proceedings of the 26^{th} ACM Symposium on Theory of Computing*, pages 648–657. ACM, ACM Press, May 1994. Submitted to *Mathematics of Operations Research*.

[Kar94b] David R. Karger. Using randomized sparsification to approximate minimum cuts. In *Proceedings of the 5^{th} Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 424–432. ACM-SIAM, January 1994. Arlington, VA.

[Kar96] David R. Karger. Minimum cuts in near-linear time. In *Proceedings of the 28^{th} ACM Symposium on Theory of Computing*. ACM, ACM Press, May 1996. Philadelphia, PA.

[KST90] Philip N. Klein, Clifford Stein, and Éva Tardos. Leighton-Rao might be practical: Faster approximation algorithms for concurrent flow with uniform capacities. In *Proceedings of the 22^{nd} ACM Symposium on Theory of Computing*, pages 310–321. ACM, ACM Press, May 1990.

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

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