Graph Sparsification Techniques

Andrew Smith
Advisor: Dan Spielman
December 14, 2007

Abstract

We consider the problem of graph sparsification. Given a graph $G$, we
would like to produce a sparse graph $G'$ that maintains certain properties
of $G$. For our purposes, we would like our sparsified graph to approxi-
mately preserve the quantity $x^T L x$ for all $x \in \mathbb{R}^n$, where $L$ is the Lapla-
cian matrix of our graph. This construction is useful in solving symmetric
diagonally-dominant linear systems quickly.

The Benczúr-Karger algorithm uses random sampling to create a spar-
sified graph $G'$ that approximately preserves all cut values of the original
graph $G$. We first describe the algorithm and discuss its properties. We
then analyze a natural simplification of the algorithm. We also simulate
the algorithm on various graphs, and determine how well it preserves the
quantity $x^T L x$ for all $x \in \mathbb{R}^n$. While the algorithm does not guarantee
that this quantity will be preserved in the sparsifier, we find that this
quantity is preserved in practice on several graphs.

1 Introduction

Given a graph $G$, we would like to construct a sparse graph $G'$ from a graph
$G$ that preserves certain properties of $G$. This is the problem of Graph Sparsi-
fication.

Consider $L$, the $n$ by $n$ Laplacian matrix of a graph $G$ (with $n$ vertices). We
would like to construct a sparse graph $G'$ from $G$ with Laplacian $L'$, such that

$$(1 - \epsilon) x^T L' x \leq x^T L x \leq (1 + \epsilon) x^T L' x \quad \forall x \in \mathbb{R}^n$$

The above property makes such a sparsified graph useful in solving diagonally-dominant linear systems quickly [5].

There are algorithms that exist which sparsify graphs as above. Spielman
and Teng wrote an algorithm to sparsify graphs to $O(n \log^{O(1)} n/\epsilon^2)$ edges in
time $O(m \log^{O(1)} m)$. Nikhil Srivastava wrote an algorithm to sparsify graphs to
$O(n \log n/\epsilon^2)$ edges in time $O(m \log^{O(1)} n)$, using effective resistances to sample
edges of the original graph. [5, 6] However, the constants in the running times
of these algorithms make it prohibitively difficult to solve linear systems quickly.
We will consider the Benczúr-Karger algorithm, which uses non-uniform random sampling to sparsify graphs. The Benczúr-Karger algorithm sparsifies graphs to $O\left(n \log n/\epsilon^2\right)$ edges that maintain approximately the same cut values. It runs in time $O(m \log^3 n)$. Note that Benczúr-Karger sparsifications are guaranteed only to maintain cut values, which is not the notion of sparsification we are looking for. In particular, they are not necessarily useful in solving linear systems quickly. However, the values of cuts are closely related to the quantity $x^T L x$ (the quantity we would like to preserve). Consider a cut $c(S, \bar{S})$, and the characteristic vector $x$:

$$x(i) = \begin{cases} 
1, & v_i \in S \\
0, & v_i \in \bar{S}
\end{cases}$$

We then have:

$$x^T L x = \sum_{e=(v_i,v_j) \in E} \omega_e (x_i - x_j)^2 = \sum_{e \in (S, \bar{S})} \omega_e = c(S, \bar{S})$$

Since each characteristic vector $x$ represents a subset $S \subseteq V$ (and therefore a cut in $G$), the Benczúr-Karger algorithm approximately preserves the quantity $x^T L x$ for all $x \in \{0, 1\}^n$. However, there are cases where it fails to preserve $x^T L x$ for vectors $x \notin \{0, 1\}^n$.

We will first provide background information on how the Benczúr-Karger algorithm works. We will then analyze a natural simplification of Benczúr-Karger briefly mentioned in their paper. We will conclude by examining how well the Benczúr-Karger algorithm approximates our notion of sparsification in practice.

## 2 The Benczúr-Karger algorithm

The Benczúr-Karger algorithm is described in [1]. The algorithm uses non-uniform random sampling on a graph $G$ to produce a graph $G'$ with the following properties:

1. $G'$ has $O(n \log n/\epsilon^2)$ edges
2. All cuts in $G$ are approximated in $G'$ within a factor of $1 \pm \epsilon$ with high probability

The algorithm works for unweighted and weighted graphs. For each edge $e \in E$, it computes a probability $p_e$ and a weight $u_e$. $p_e$ is the probability edge $e$ is included in graph $G'$, and $u_e = 1/p_e$ is the weight edge $e$ is assigned in graph $G'$ if it is included. The Benczúr-Karger algorithm finds these sampling probabilities in $O(m \log^3 n)$ time for weighted graphs.

We will now explain what these probabilities are, and why they result in a sampled graph that satisfies properties 1 and 2.
2.1 Random sampling in graphs

We will first explain how random sampling helps us build sparsified graphs.

**Lemma 1.** Consider any random sampling scheme for a graph $G$, where each edge is included in $G'$ with some probability $p_e$, and given weight $1/p_e$ if included.

Sampling according to any such scheme preserves cut values in expectation.

**Proof.** Let $X_e$ be a random variable for each edge, which is $w_e = 1/p_e$ with probability $p_e$ and 0 otherwise. Let $C$ be any cut in graph $G$.

$$E[\text{val}_{G'}(C)] = E\left[\sum_{e \in C} X_e\right] = \sum_{e \in C} E[X_e] = \sum_{e \in C} \frac{1}{p_e} p_e = \text{val}_G(C)$$

Lemma 1 says that the expected value of a cut in $G'$ is its value in $G$. However, with arbitrary probabilities $p_e$, properties 1 and 2 might be violated. If the inclusion probabilities for each edge are too big, $G$ can have more than $\Theta(n \log n/\epsilon^2)$ edges. If the probabilities are too small, the value of some cuts in $G'$ vary widely from their expectations (with non-negligible probability).

The following lemma will be helpful in proving that cut values are concentrated around their expectation (property 2).

**Lemma 2.** Consider any sampling scheme where each edge $e$ is sampled with probability $p_e$. Let $X_e = 1$ with probability $p_e$, and 0 otherwise. Consider any cut $C$, and let $X_C = \sum_{e \in C} X_e$.

If $E[X_C] = \Omega\left(\ln n/\epsilon^2\right)$, then $X_C \in (1 \pm \epsilon)E[X_C]$ with high probability.

**Proof.** By the Chernoff bound,

$$Pr[X_C \notin (1 \pm \epsilon)E[X_C]] \leq 2e^{-\frac{\epsilon^2E[X_C]}{3}}$$

We want to preserve cut values to within an arbitrary factor of $1 \pm \epsilon$ with high probability ($1 - 1/n^d$ for some constant $d$). We therefore need to ensure that in our sampling scheme, $E[X_C]$ is sufficiently large for any cut $C$:

$$2e^{-\frac{\epsilon^2E[X_C]}{3}} \leq \frac{1}{n^d} \iff E[X_C] = \Omega\left(\frac{\ln n}{\epsilon^2}\right)$$

Therefore, any sampling scheme for which $E[X_C] = \Omega(\ln n/\epsilon^2)$ results in $X_C$ being concentrated around its expectation.
We will now attempt to design a scheme that picks $p_e$ probabilities that satisfy properties 1 and 2. Consider a simple scheme that samples each edge in graph $G$ with some constant probability $\rho$ (and assigns each included edge weight $1/\rho$). By lemma 1, this scheme preserves cut values in expectation for all values of $\rho$.

We would like to determine for what values of $\rho$ are cuts concentrated around their expected value. We will use lemma 2. Let $C$ be a single cut in $G$ of value $c$. Let $X_e$ be a random variable for each edge, which is 1 with probability $\rho$ and 0 otherwise. Let $X_C = \sum_{e \in C} X_e$. In our uniform scheme, we have $E[X_C] = \rho c$.

To use lemma 2, we need

$$E[X_C] = \rho c \geq \Omega \left( \frac{\ln n}{\epsilon^2} \right) \Rightarrow \rho = \Omega \left( \frac{\ln n}{\epsilon^2 c} \right)$$

Thus, in order to guarantee that any $X_C$ is concentrated around its expectation, we need to set $c$ in the above expression for $\rho$ equal to the minimum cut value of $G$ (so the conditions for Lemma 2 hold for any particular cut).  

We can now apply lemma 2 and declare that $X_C$ is highly concentrated around the mean $E[X_C]$ for each cut $C \in G'$. Note that the value of a cut $C$ in $G'$ is $X_C/\rho$, as each edge has weight $1/\rho$ in $G'$. Since the cut values are constant multiples of $X_C$, the cut values are also highly concentrated around their expectations.

We have therefore shown that our scheme approximates any particular cut in the original graph.

The problem with this scheme is that if the minimum cut is low, property 1 is not satisfied. The number of edges would be too high to suit our purposes:

$$E \left[ \sum_{e \in G} X_e \right] = \rho m = \frac{m \ln n}{\epsilon^2 c}$$

This will get us our desired edge bound in property 1 only for a graph whose minimum cut $c = \Omega(n)$. For example, consider the dumbbell graph. The minimum cut in this graph is 1, so our sparsified graph will include every edge of the original graph (far from our goal of $O(n \log n/\epsilon^2)$ edges).

We can achieve a better sparsification with a different kind of sampling scheme. The key to overcoming the problem above is to use non-uniform sampling. In the dumbbell graph, the only cut with a small value is the minimum cut (passing through the single edge connecting the two complete graphs). Every other cut has $\Omega(n)$ edges. We will therefore consider the resulting graph

---

1 We will be proving that our scheme satisfies a relaxed version of property 2, where any particular cut is approximated within a factor of $1 \pm \epsilon$. This will later be extended to all cuts simultaneously.

2 Note that if our sampling probability is calculated to be greater than 1 for a particular edge, we simply make it 1 for that edge.
after sampling the middle edge with probability \( p_e = 1 \), and all other edges with probability \( p_e = \sigma_e = \Theta(ln n/\epsilon^2 n) \). We will (as always) let the weight of each sampled edge be the inverse of the probability it was sampled with, so that cut values are preserved in expectation (by lemma 1).

We will again consider the random variable \( X_C = \sum_{e \in C} X_e \), where \( X_e \) is 1 with probability \( p_e \). Under this non-uniform sampling scheme for the dumbbell graph, it is still true that \( X_C \) remains close to its expectation for any cut \( C \). For the minimum cut \( C_m \), \( X_{C_m} = 1 \) with probability 1, which is its expectation. For any other cut \( C \), \( E[X_C] \) is still large enough under our non-uniform sampling scheme to apply Lemma 2:

\[
E[X_C] = \sum_{e \in C} \rho_e \geq \Theta \left( \frac{\ln n}{\epsilon^2 n} \right) \geq \Theta \left( \frac{\ln n}{\epsilon^2 n} \right) n = \Theta \left( \frac{\ln n}{\epsilon^2} \right)
\]

Thus, by Lemma 2, \( X_C \in (1 \pm \epsilon)E[X_C] \). However, while we have shown that \( X_C \) is concentrated around its expectation, it is not obvious how to use that concentration result to prove that the cut values themselves are concentrated. Recall that in our sampled graph, we need to weight each included edge \( e \) by \( 1/p_e \) to ensure the cuts are identical in expectation. Thus, the actual value of a cut in our sampled graph is \( \sum_{e \in C} X_e/p_e \). We only proved that \( \sum_{e \in C} X_e \) was concentrated about its mean; this does not immediately apply to the former sum, as is not a constant multiple of the latter.

This was not a problem for our uniform scheme. For that scheme, we had that the value of cut \( C \) in our sampled graph was \( \sum_{e \in C} X_e/p_e = 1/\rho \sum_{e \in C} X_e = X_C/\rho \). Thus, our concentration result for \( X_C \) immediately applied to the cut value (a constant multiple of \( X_C \)). In our non-uniform scheme, however, our varying weights assigned to the edges in our sampled graph \( (1/p_e) \) make this analysis fail, since the value of cut \( C \) in our sampled graph is no longer a constant multiple of \( X_C \). The problem in our case is the middle edge; it is sampled with a different probability \( p_e \) than all the others (and thus has a different weight \( 1/p_e \)).

We will remedy this problem by decomposing our sampled dumbbell graph into two different graphs with constant weights: one with all edges, and one with all edges except the middle edge. Let these two graphs be \( G'_0 \) and \( G'_1 \). We will weight all edges in \( G'_0 \) with weight 1, and all edges in \( G'_1 \) with weight \( (1/\sigma_e - 1) \).

All edges in \( G'_0 \) have the same weight (1), so we can now apply the concentration result for \( X_C \) to the values of the cuts: cut \( C \) has value \( \sum_{e \in C} X_e = X_C \). Thus, cuts in \( G'_0 \) are concentrated around their expectation in \( G'_0 \).

Likewise, all edges in \( G'_1 \) have the same weight \( (1/\sigma_e - 1) \). We sampled each edge in \( G'_1 \) with probability \( \Theta(ln n/\epsilon^2 n) \). Since each cut in \( G'_1 \) has value \( \Omega(n) \), we get that \( E[X_C] \geq \Theta(ln n/\epsilon^2 n) \Omega(n) = \Omega(ln n/\epsilon^2) \). Thus, we can also apply lemma 2 to \( G'_1 \), which tells us that \( X_C \) is concentrated about its mean for all cuts \( C \). As with \( G'_0 \), the fact that all edges in \( G'_1 \) have the same weight means we can immediately apply this concentration result to cut values in \( G'_1 \); the value of a cut in \( G'_1 \) is \( \sum_{e \in C} X_e (1/\sigma_e - 1) = (1/\sigma_e - 1)X_C \), which is just a
constant multiple of $X_C$.

We have therefore shown that cut values in $G'_0$ and $G'_1$ are concentrated about their expectations. We can now show that cuts in $G'$ (our original sampled graph) are concentrated around their expectations. By lemma 1, these expectations are simply the values of the cuts in $G$ (our original graph).

Consider a cut $C$ in $G'$. Using our decomposition above, we have the following:

$$\text{val}_{G'}(C) = \text{val}_{G'_1}(C) + \text{val}_{G'_0}(C)$$

(1)

To see this, note that any non-middle edge $e$ belonging to the cut $C$ has weight $1/\sigma_e$ in $G'$. Furthermore, edge $e$ contributes value $(1/\sigma_e - 1) + 1 = 1/\sigma_e$ to the right hand side. Likewise, the middle edge has weight 1 in $G'$, and contributes $0 + 1 = 1$ to the right hand side.

We have now shown that $\text{val}_{G'_1}(C)$ and $\text{val}_{G'_0}(C)$ are both concentrated around their means. We can therefore conclude by (1) that $\text{val}_{G'}(C)$ is concentrated about its mean, which is $\text{val}_G(C)$ (by lemma 1). This shows that any particular cut in $G'$ is concentrated about its value in $G$ mean after sampling.

Our non-uniform sampling scheme gives us an edge bound of

$$E\left[\sum_{e \in G} X_e\right] = \sigma_e(m - 1) + 1 = \Theta\left(\frac{m \ln n}{\epsilon^2 k_e}\right)$$

$$= \Theta\left(\frac{n \ln n}{\epsilon^2}\right)$$

since $m/n = \Theta(n)$. This is precisely the edge bound we are looking for in property 1.

### 2.2 The Benczúr-Karger sampling scheme

The Benczúr-Karger algorithm generalizes this example to all graphs. Let $k_e$ be the greatest minimum cut of any maximal induced subgraph containing edge $e$. Benczúr and Karger denote this quantity as the strong connectivity, or strength, of edge $e$. For example, the strong connectivity of the middle edge in the dumbbell graph is 1, and the strong connectivity of every other edge in the graph is $n/2 - 1$.

Benczúr-Karger samples edges in $G$ according to probabilities $p_e = \Theta\left(\frac{\ln n}{\epsilon^2 k_e}\right)$, and assigns each included edge weight $1/p_e$ in our sampled graph $G'$.

We will now show that sampling with these probabilities results in a graph that approximately preserves cut values. The following proof is a summary of Benczúr-Karger’s proof in [1].

#### 2.2.1 Proof that cut values are approximately preserved

As in the previous example, we will decompose $G$ into different graphs $G_1, ..., G_m$. $G_i$ refers to the graph before sampling, and $G'_i$ refers to the graph after sampling. We will prove that cuts in each graph $G'_i$ are concentrated about their
means. This will then imply that for each cut $C$, $val_{G'}(C)$ is concentrated about its mean, which is simply $val_G(C)$ by lemma 1.

Since there are $m$ edges, there are at most $m$ distinct values of $k_e : k_1 < k_2 < ... < k_m$. Each value of $k_e$ corresponds to a sampling probability $\sigma_i = \Theta(\ln n/\epsilon^2 k_i)$. Each $G_i$ described above will only include edges with strength at least $k_i$, and each edge of $G_i$ will be assigned the weight $(1/\sigma_i - 1/\sigma_{i-1})$. We will now prove that each cut in each $G_i$ is concentrated about its mean.

Consider some cut $C$ in $G_1$. Since every edge in $G_1$ has strength at least $k_1$, all $e \in C$ have strength at least $k_1$. We thus have

$$E[X_C] = \sum_{e \in C} p_e = \sum_{e \in C} \Theta\left(\frac{\ln n}{\epsilon^2 k_e}\right) \geq \sum_{e \in C} \Theta\left(\frac{\ln n}{\epsilon^2 k_1}\right)$$

$$\geq \Theta\left(\frac{\ln n}{\epsilon^2 k_1}\right) k_1$$

$$= \Theta\left(\frac{\ln n}{\epsilon^2}\right)$$

Note that (3) follows from the fact that every cut in $G'_1$ must have at least $k_1$ edges, by definition of strong connectivity.

We can now apply lemma 2 to get that $X_C$ is concentrated about its mean in $G'_1$. Because each edge has the same weight $1/\sigma_1$ in $G'_1$, the value of any cut $C$ in $G'_1$ is simply $1/\sigma_1 X_C$. Thus, the value of each cut in $G'_1$ is highly concentrated about the mean (being a constant multiple of $X_C$).

Now, consider $G'_i$ for arbitrary $i$, which contains all edges of strength at least $k_i$. The strength of each edge $e \in G'_i$ is the same as its strength in $G$, as follows. Since edge $e \in G_i$, $k_e \geq k_i$. By definition of strong connectivity, there exists some subgraph of $G$ whose minimum cut is $k_e \geq k_i$. Such a subgraph must also be contained in $G_i$; if this were not the case, that subgraph would have to have edges of strength $< k_i \leq k_e$. This contradicts the definition of $k_e$ as the maximum value of $k$ such that a maximal induced $k$-connected component contains $e$.

Because edge strengths are preserved in our decomposition, we can apply the same argument (2 - 4) to each $G'_i$ to see that $E[X_C] = \Omega(\ln n/\epsilon^2)$ for each cut in $G'_i$. By lemma 2, $X_C$ is therefore concentrated around its expectation. Since the weight of each edge in $G'_i$ is constant $(1/\sigma_i - 1/\sigma_{i-1})$, the value of each cut $C$ in $G'_i$ is $(1/\sigma_i - 1/\sigma_{i-1}) X_C$, which is also concentrated around its expectation.

We have now shown that all cuts in $G'_i$ are concentrated around their expectations, for all $i$. We now observe that

$$val_{G'}(C) = \sum_{i=1}^m val_{G'_i}(C)$$

To see this, note that any edge $e \in C$ with strength $k_i$ has weight $1/\sigma_i$ in $G'$. Such an edge contributes $\sum_{j=1}^i (1/\sigma_j - 1/\sigma_{j-1}) = 1/\sigma_i - 1/\sigma_0 = 1/\sigma_i$ to the right hand side, since this edge does not appear in any $G'_j$ for $j > i$.  

7
Since \( \text{val}_{G'}(C) \) is the sum of quantities concentrated around their expectations (with high probability), \( \text{val}_{G'}(C) \) itself is concentrated around its expectation, which is just \( \text{val}_G(C) \).\(^3\) This proves that each cut in \( G' \) approximates the value of the corresponding cut in \( G \) within a factor of \( (1 \pm \epsilon) \), with high probability.

Note that property 2 calls for every cut to be approximated within a factor of \( (1 \pm \epsilon) \) with high probability. The concentration result in Lemma 2 only implies that each particular cut is approximated within this factor with high probability. However, Karger proved in a previous paper that our sampling scheme still satisfies this stronger criterion:

**Sampling Theorem.** Let \( G \) be an unweighted graph. Assume that we generate an unweighted sampled graph \( G' \) by sampling each edge \( e \) of \( G \) with probability \( p_e \). Then with probability \( 1 - O(1/n^d) \), every cut is approximated to within a factor of \( \delta = \sqrt{2(d+2)(\ln n)/c} \), where \( c \) is the minimum expected value over all cuts.

Since we assume in Lemma 2 that \( E[X_C] = \Theta \left( \frac{\ln n}{c^2} \right) \), we can set \( c \) equal to this quantity. With the appropriate constant factor \( 2(d+2) \), this gives us \( \delta = \epsilon \) in the above theorem. The above theorem therefore indicates that every cut in \( G' \) is highly concentrated around its expectation. The value of any cut \( C \) in \( G' \) is simply \( \sum_{e \in C} X_e = X_C \) (since \( G' \) is unweighted). This means that the sampling lemma implies that for all cuts \( C \), \( X_C \) is highly concentrated around its expectation, which matches the stronger concentration result required to satisfy property 2. Thus, our analysis of the Benczúr-Karger algorithm holds with the sampling theorem in place of lemma 2, and the values of all cuts are therefore approximated with high probability.

While the above proof was for unweighted graphs, any weighted graph is equivalent to an unweighted multigraph with \( w_e \) copies of edge \( e \) (in the sense that all cuts have the same value). The above proof also holds for unweighted multigraphs, though the algorithm used to obtain the strong connectivities is slightly different to avoid exponential blowup with large weights.

### 2.2.2 Proof of edge bound in property 1

The Benczúr-Karger sampling probabilities also satisfy property 1 as follows. Consider the quantity \( \sum_{e \in E} w_e/k_e \). We will show that this quantity is \( O(n) \).

Consider any component of \( G \) whose minimum cut \( C \) has \( k \) edges. Every \( e \in C \) has strong connectivity at least \( k \) (by definition). This means that

\(^3\)We proved that \( \text{val}_{G'}(C) \) was within a factor of \( (1 \pm \epsilon) \) with probability \( 1 - 1/n^d \). This means that the probability that \( \text{val}_{G'}(C) \) is highly concentrated around the mean for all \( i \) is at most \( 1 - 1/n^{d-2} \) by the union bound (as there are at most \( m \leq n^2 \) graphs \( G'_i \)). We can fine-tune \( d \) to any constant we desire by adjusting the constant factor in our sampling probabilities. This means that \( \text{val}_{G'}(C) \) is concentrated around the mean for all \( i \) with high probability, which implies the same for \( \text{val}_G(C) \) (their sum).

\(^4\)For a proof, see [2].
\[ \sum_{e \in C} w_e / k_e \leq \sum_{e \in C} w_e / k \leq k / k = 1. \]

We can repeat this analysis up to \( n - 1 \) times, which breaks \( G \) into \( n \) connected components. With \( n \) connected components, there are no edges remaining. This means that

\[ \sum_{e \in E} w_e / k_e \leq n - 1 \]

Let \( X = \sum_{e \in E} X_e \), where \( X_e \) is 1 with probability \( p_e \) and 0 otherwise. The total expected number of edges sampled is

\[ E[X] = \sum_{e \in E} p_e = \frac{\ln n}{c^2} \sum_{e \in E} \Theta \left( \frac{w_e}{k_e} \right) = O \left( \frac{n \ln n}{c^2} \right) \]

By the Chernoff bound,

\[ \Pr[X \notin (1 \pm \delta) E[X]] \leq 2e^{-\frac{\delta^2 E[X]}{3}} \leq 2e^{-\frac{n \log n \delta^2}{\epsilon^2}} \leq 2e^{-\frac{n \log n \delta^2}{\epsilon^2}} \]

which is exponentially small in \( n \) for any \( \delta = \Omega \left( \frac{\epsilon}{\sqrt{\log n}} \right) \).

### 2.3 Obtaining strong connectivities

The Benczúr-Karger algorithm obtains approximations of the strong connectivity \( (k_e) \) of each edge. These approximations need to be lower bounds, since sampling with upper bounds risks damaging the cut values. Using lower bounds instead of exact values can only increase the sampling probabilities of each edge, improving upon the condition required to invoke the sampling theorem. The algorithm uses lower bounds tight enough such that the number of edges sampled is still \( O(n \log n / \epsilon^2) \), as required in property 1.

To calculate these approximations of strong connectivity, the algorithm obtains a small set of edges \( S_k \) that contain all edges of strong connectivity \( k \) or less. Once we have the ability to produce such a set, we can simply assign \( k_e^* = k / 2 \) to all edges in \( S_k \setminus S_{k/2} \). These are lower bounds, since any edge in \( S_k \setminus S_{k/2} \) by definition has strong connectivity at least \( k / 2 \). Note that the tightness of the lower bounds (and therefore the resulting number of edges) is directly determined by how small the set \( S_k \) is that we generate. If \( S_k \) is small enough to only contains edges of strong connectivity \( k \) or less, then the lower bounds we obtain are within a factor of 2 of the actual strong connectivities. This would result in an edge bound that is at most a factor of 2 larger than we would have if we calculated the strong connectivities exactly. This would be ideal. Unfortunately, computing such a tight set \( S_k \) would take too long for our purposes.
On the other hand, we cannot use an $S_k$ that contains too many edges of strong connectivity greater than $k$, as we set $k^*_e = k$ for all edges in $S_k \setminus S_k/2$. If $S_k$ had too many edges of strong connectivity greater than $k$, the fact that we are setting $k^*_e = k/2$ for all of these edges would make our lower bounds too loose (and result in too many edges in our sparsified graph).

The idea behind the Benczúr-Karger algorithm is to obtain a small enough $S_k$ such that the resulting lower bounds result in a sparsifier with $O(n \log n/\epsilon^2)$ edges. The Benczúr-Karger algorithm uses the Nagamochi-Ibaraki algorithm to come up with a preliminary candidate for $S_k$. It then reduces the number of edges in this set by contracting certain edges that clearly have large connectivity, and repeats the process on the resulting set. Eventually, $S_k$ is small enough to be useful in computing relatively tight lower bounds on $k^*_e$, and we set $k^*_e = k/2$ for all edges in $S_k \setminus S_k/2$. The algorithm does this for $k = 1, 2, 4, 8, ..., m$ for unweighted graphs, thereby obtaining approximations for the strong connectivities of every edge. It then samples each edge with probability $\Theta(\ln n/k^*_e \epsilon^2)$ to obtain the sparsified graph.\(^5\)

### 2.4 The Nagamochi-Ibaraki algorithm

The Benczúr-Karger algorithm uses as a subroutine another algorithm by Nagamochi and Ibaraki. The algorithm is presented in [3] and [4]. The algorithm computes a sparse-certificate $G_k$ of any graph $G$ and any value $k$ that satisfies the following properties:

1. $G_k$ has at most $k(n - 1)$ edges
2. $\lambda(x, y, G_k) \geq \min\{\lambda(x, y, G), k\}$

where $\lambda(x, y, G)$ is the value of the minimum $x$-$y$ cut in $G$. In particular, all edges of local edge-connectivity (standard connectivity) $k$ or less appear in $G_k$. This is guaranteed by property 2, as for $e = (x, y)$, if $\lambda(x, y, G) \leq k$, then $\lambda(x, y, G_k) \geq \lambda(x, y, G)$. This implies $e \in G_k$, since every cut separating $x$ and $y$ includes edge $e$. This property is useful for computing the strong connectivity of each edge, since it gives us a small subset of edges including all edges of standard connectivity at most $k$. Much of the work of the Benczúr-Karger algorithm is using this subroutine to produce a small set $S_k$ of edges including all edges of strong connectivity at most $k$.

For unweighted graphs, the Nagamochi-Ibaraki algorithm works by creating a partition $E_1, E_2, \ldots, E_m$. It first finds a spanning forest $F_1$ and sets $E_1 = F_1$. Then, it removes $F_1$ from $G$ and finds a new spanning forest $F_2$ on $G \setminus F_1$, and sets $E_2 = F_2$. It continues this process up to $E_m$. Once the partition is computed, the resulting sparse certificate is $G_k = \cup_{i=1}^k E_i$.

\(^5\)For weighted graphs, the algorithm computes $S_k$ for $k = 1, 2, 4, 8, ..., nW$, where $W$ is the weight of the largest edge, and sets $k^*_e = k$ for all edges in $S_k \setminus S_k/2$. It then samples each edge with probability $\Theta(\ln nW/k^*_e \epsilon^2)$. If $W$ is not polynomial in $n$, improvements can be made to cut the resulting factor of $\log nW$ in the running time to $\log n$. This results in a total running time of $O(m \log^3 n)$. 

10
For each edge \( e \), define its height \( h_e \) to be the value of \( i \) such that \( E_i \) contains \( e \). Note that for \( e = (x, y) \), \( h_e \) is a lower bound on \( \lambda(x, y, G) \). This is because if \( \lambda(x, y, G) = k < h_e \), then edge \( e \) must be in \( G_k = \bigcup_{i=1}^{k} E_i \), contradicting the fact that \( e \in E_{h_e} \).

The Nagamochi-Ibaraki algorithm can be modified to directly handle weighted graphs. For purposes of our analysis, we will only consider graphs with integer weights. In this case, a weighted graph is simply an unweighted multigraph (with each edge of weight \( w_e \) replaced by \( w_e \) parallel edges). The height \( h_e \) of edge \( e = (x, y) \) is now defined to be the maximum value of \( i \) such that \( E_i \) contains a parallel edge of the form \( (x, y) \). The height \( h_e \) is still a lower bound on the local-edge-connectivity of \( e \), since all edges of the form \( (x, y) \) have the same local-edge connectivity (which means we can apply the lower-bound argument above to the edge \( (x, y) \in E_i \) with maximum \( i \)).

### 3 Simplification of Benczúr-Karger

Recall that the Benczúr-Karger algorithm uses the Nagamochi-Ibaraki algorithm to come up with a value \( k_e^* \) for each edge \( e \) with weight \( w_e \), and samples each edge \( e \) with probability \( p_e = O(w_e/k_e^*) \). The \( k_e^* \) values for each edge \( e \) are similar to the \( h_e \) values returned by the Nagamochi-Ibaraki algorithm. While \( h_e \) is a lower bound on the standard edge-connectivity of edge \( e \), \( k_e^* \) is a lower bound on the strong edge-connectivity of edge \( e \), denoted by \( k_e \).

The following is a natural question: can we sample graph \( G \) with probabilities \( O(w_e/h_e) \) instead of \( O(w_e/k_e^*) \)? This would greatly improve the running time of our algorithm. Benczúr-Karger runs in time \( O(m \log^3 n) \). Much of this time is spent finding small sets with all edges of strong connectivity \( k \) or less. On the other hand, obtaining the \( h_e \) values only requires one run of the Nagamochi-Ibaraki algorithm, which takes time \( O(m + n \log n) \). This is essentially linear, since if a graph had \( O(n \log n) \) edges it would not need sparsification. Benczúr and Karger were unable to prove that property 2 holds for \( p_e = O(w_e/\lambda_e) \), where \( \lambda_e \) is the standard connectivity of edge \( e \). However, they were also unable to come up with any counterexamples where property 2 does not hold after sampling with those probabilities (i.e., where values of cuts are not approximately equal after random sampling). This motivates our simplification, since \( h_e \) is a lower bound on \( \lambda_e \).

We concern ourselves with property 1. We prove the following:

**Theorem 1.** For unweighted graphs, sampling with \( p_e = \tilde{O}\left(\frac{1}{h_e}\right) \) results in a graph \( G' \) that has \( O\left(\frac{n \log^2 n}{h_e^2}\right) \) edges with high probability.

**Theorem 2.** For weighted graphs, sampling with \( p_e = \tilde{O}\left(\frac{w_e}{h_e}\right) \) results in a graph \( G' \) that has \( \max\left(O\left(\frac{n \log n \log W}{h_e^2}\right), |E|\right) \) edges with high probability, where \( W \) is the largest weight of any edge.

---

6 Sampling with probability \( \tilde{O}(p_e) \) implies a hidden factor that is \( \Theta(\log n/\epsilon^2) \).
We also prove a converse of theorem 2, which shows that theorem 2 is essentially tight for graphs with large weights:

**Theorem 3.** There are weighted graphs for any $|V|$ such that sampling with $p_e = \tilde{O}\left(\frac{w_e}{h_e}\right)$ results in a graph $G'$ that includes every edge of $G$ with high probability.

This leads us to conclude that sampling with probabilities $\tilde{O}\left(\frac{w_e}{h_e}\right)$ is probably not a good idea for weighted graphs, since the proportion of edges included in $G'$ can be arbitrarily high.

### 3.1 Theorem 1

Consider sampling with probability $\tilde{O}\left(\frac{1}{h_e}\right)$. First, note that for any $i$, $|E_i| \leq (n - 1)$ in the Nagamochi-Ibaraki algorithm (since $E_i$ is a spanning forest for each $i$). We thus have the following:

$$\sum_{e \in G} \frac{1}{h_e} \leq \sum_{i=1}^{m} \frac{n - 1}{i} \leq (n - 1)H_m \leq (n - 1)H_{n^2} = O(n \log n)$$

(5) comes from the fact that all edges have height at most $m$, and that at most $(n - 1)$ edges (a maximal spanning forest) can share a single height.

Let $X_e$ be 1 if edge $e$ is included in $G'$, and 0 otherwise. Note that the number of edges in $G'$ is simply $X = \sum_{e \in G} X_e$. We thus have the following:

$$E \left[ \sum_{e \in G} X_e \right] = \sum_{e \in G} E[X_e] = \ln n \sum_{e \in G} \frac{1}{h_e} \leq \frac{n \log^2 n}{\epsilon^2}$$

By the Chernoff bound,

$$Pr[X \notin (1 \pm \delta)E[X]] \leq 2e^{-\frac{\delta^2 E[X]}{3}} \leq 2e^{-\frac{n \log^2 n \delta}{\epsilon^2}}$$
which is exponentially small in \( n \) for any \( \delta = \Omega \left( \frac{\epsilon}{\log n} \right) \).

### 3.2 Theorem 2

Consider a weighted graph \( G \) as an unweighted multigraph \( H \). Note that the total number of edges in this multigraph is at most \( \frac{nW(n-1)}{2} \), where \( W \) is the maximum edge weight. Since each height can contain at most \( (n-1) \) edge weight, the maximum height assigned to any edge is at most \( \frac{nW(n-1)}{2}/(n-1) = \frac{nW}{2} \). Using the same analysis as in Theorem 1, we get

\[
\sum_{e \in H} \frac{1}{h_e} \leq \sum_{i=1}^{nW} \frac{n-1}{i} = (n-1)H_{\frac{nW}{2}} = O(n \log (nW))
\]

But there is a difference between the way the Nagamochi-Ibaraki algorithm handles our graph as a weighted graph and as an unweighted multigraph. As an unweighted multigraph, it assigns a different height to each parallel edge \((x, y)\), since two parallel edges cannot be part of the same spanning forest. As a weighted graph, it assigns a single height to weighted edge \((x, y)\). So we cannot immediately apply our analysis of unweighted graphs to the corresponding unweighted multigraph.

However, the single height assigned to \((x, y)\) in the weighted graph is the \textit{maximum} height assigned to any of the parallel edges \((x, y)\) in the multigraph. This means that \( \frac{1}{h_e} \) for edge \( e \in G \) is at most \( \frac{1}{h_{e'}} \) for a parallel copy \( e' \in H \) of \( e \in G \). We therefore have

\[
\sum_{e \in G} \frac{w_e}{h_e} = \sum_{e \in G} \sum_{i=1}^{nW} \frac{1}{h_e} \leq \sum_{e' \in H} \frac{1}{h_{e'}} = O(n \log (nW))
\]

as above.

Therefore, by the same analysis in theorem 1, the number of edges in our sampled graph is \( \max \left( O \left( \frac{n \log n \log(nW)}{\epsilon^2} \right), |E| \right) \) with high probability.

### 3.3 Theorem 3

We will describe a family of graphs for which sampling with probabilities \( p_e = \tilde{O}(w_e/h_e) \) results in every edge being included with high probability. For reference, we reproduce the Nagamochi-Ibaraki algorithm for weighted graphs as Algorithm 1.

Let \( \alpha \) be a constant. Consider the complete graph on \( n \) vertices with weights defined as follows:

- \( w_{1,i} = \alpha^0 \) for \( 1 < i \leq n \)
- \( w_{2,i} = \alpha^1 \) for \( 2 < i \leq n \)
- \( w_{3,i} = \alpha^2 \) for \( 3 < i \leq n \)

...
Algorithm 1 Nagamochi-Ibaraki Algorithm [4]

Require: \( G = (V, E) \), edge weights \( w_e \) \( \forall e \in E \)

Ensure: \( h_e > 0 \) \( \forall e \in E \)

1: Label all nodes \( v \in V \) and all edges \( e \in E \) as unscanned
2: \( r_v \leftarrow 0 \) \( \forall v \in V \); \( q_e \leftarrow 0 \) \( \forall e \in E \)
3: while there exist unscanned vertices do
4: Pick unscanned node \( v \in V \) with largest \( r_v \)
5: for all unscanned edges \( e = (v, u) \) do
6: \( h_e \leftarrow r_u + w_e \)
7: \( r_u \leftarrow r_u + w_e \)
8: Mark edge \( e \) as scanned
9: end for
10: Mark vertex \( v \) as scanned
11: end while

\[ w_{k,i} = \alpha^{k-1} \text{ for } k < i \leq n \]

We will assume that if there is a tie in \( r_v \) values at step 1 of the while loop, the lowest-numbered vertex is picked.

We will show by induction that

1. Step 4 always picks vertex \( k \) in the \( k \)th iteration of the while loop
2. At the end of the \( k \)-th iteration of the while loop, \( r_v = \sum_{j=0}^{k-1} \alpha^j \) for all unscanned \( v \in V \).

Initially, all \( r_v = 0 \). The algorithm thus picks vertex 1 in step 4. The algorithm will then set \( r_v = 1 \) for all unscanned \( v \) (since \( v_1 \) is adjacent to all \( v \in V \), and because \( w_{1,i} = \alpha^0 = 1 \) for all \( 1 < i \leq n \)).

We now assume by induction that at the beginning of the \( k \)th iteration of the while loop, \( r_v = \sum_{j=0}^{k-2} \alpha^j \) for all unscanned \( v \in V \), and that \( v_1, v_2, \ldots, v_{k-1} \) have been scanned. The algorithm picks the lowest numbered unscanned vertex \( v_k \) (since all unscanned \( r_v \) values are the same by the induction hypothesis). This proves part 1 of the induction hypothesis.

The unscanned edges adjacent to \( v_k \) are \( (k, i) \), \( k < i \leq n \). Thus, vertices \( k+1 \) through \( n \) are set to \( w_{(k,i)} + r_i = \alpha^k + \sum_{j=0}^{k-2} \alpha^j = \sum_{j=0}^{k-1} \alpha^j \). But vertices \( k+1 \) through \( n \) are precisely all of the unscanned vertices. This proves part 2 of the induction hypothesis.

This means that that on the \( k \)th iteration of the while loop, \( h_{(k,i)} \) is set to \( w_{(k,i)} + r_i = \alpha^{k-1} + \sum_{j=0}^{k-2} \alpha^j \) (by the above proof) = \( \sum_{j=0}^{k-1} \alpha^j \). This gives us
the following for all $i < j$:

\[
\begin{align*}
\frac{w_{(i,j)}}{h_{(i,j)}} &= \frac{\sum_{j=0}^{i-1} \alpha^{i-1}}{\alpha^{i-1} - 1} \\
&= \frac{\alpha^i - 1}{\alpha - 1} \\
&= \frac{1 - \frac{1}{\alpha}}{1 - \frac{1}{\alpha^i}} \\
&\geq 1 - \frac{1}{\alpha}
\end{align*}
\]

Thus, the probability of picking each edge is at least $1 - 1/\alpha$. This means that the probability of picking all edges is at least

\[
\left(1 - \frac{1}{\alpha}\right)^{\binom{n}{2}} \geq 1 - \frac{\binom{n}{2}}{\alpha} \geq 1 - e^{-n}
\]

if we set $\alpha = \left(\frac{n}{2}\right) e^n$. (6) follows from the fact that $(1 - c)^k \geq 1 - ck$.

4 Performance of Benczúr-Karger algorithm for sparsification

Benczúr and Karger proved that their sampling scheme approximately preserved all cut values (and thus the quantity $x^T L x$ for all $x \in \{0, 1\}^n$). For sparsifications to be useful in solving linear systems quickly, such graphs need to preserve the quantity $x^T L x$ for all $x \in \mathbb{R}^n$.

4.1 Theoretical performance

It is not immediately obvious that the Benczúr-Karger algorithm does not also succeed in this type of sparsification. However, there are indeed examples on which the Benczúr-Karger algorithm fails in this regard. One such example is shown in figure 1 (similar to the example mentioned in [5]). In this example, there are $b$ sets of $a$ vertices arranged in a circle, and there is a complete bipartite graph between every two adjacent sets of $a$ vertices (with one edge in the middle connecting two vertices of non-adjacent sets). All edges have unit weight.

Let $x$ be the vector that assigns the blue numbers in the graph to each vertex in its corresponding set. Note that all edges in the complete bipartite graphs $e = (u, v)$ have $(x_u - x_v)^2 = 1$, while the middle edge $e' = (u', v')$ has $(x_{u'} - x_{v'})^2 = b^2/4$. If we let $b = a^3$, we get a graph of $m$ edges and $n$ vertices, with $n = b^{4/3}$ and $m = b^{5/3} + 1$ (which means $m = n^{5/4} + 1$). If $L$ is the Laplacian of this graph, we get
Figure 1: Graph that the Benczúr-Karger algorithm fails to sparsify

\[ x^T L x = \sum_{e=(u,v) \in E} (x_u - x_v)^2 = b^2 + b^2/4 \]

The strong connectivity of the middle edge is the connectivity of the whole graph, which is \(2a = 2n^{1/4}\). Thus, if we want constant error in cut values, the middle edge is discarded with probability \(1 - \Theta \left( \frac{\ln n}{n^{1/4}} \right) \).

With the middle edge discarded, we have \(x^T L' x \leq b^{5/3}\), which gives us

\[
\frac{x^T L x}{x^T L' x} \geq \frac{b^{5/3} + b^2/4}{b^{5/3}} = 1 + \frac{b^{1/3}}{4} = 1 + \frac{n^{1/4}}{4}
\]
which grows arbitrarily high with \( n \). Thus, with probability \( 1 - \Theta(\ln n/n^{1/4}) \), the Benczúr-Karger algorithm fails to sparsify the above graph.

### 4.2 Practical performance

While the Benczúr-Karger algorithm does not give any reasonable theoretical guarantee for our required notion of sparsification, it would still be of interest if the algorithm correctly sparsified graphs in practice. If we are using our sparsifications to solve linear systems quickly, even an algorithm that does not guarantee an accurate sparsification is still useful. We can first attempt to solve the linear system with the sparsification. We can then immediately tell whether our solution (approximately) satisfies \( Ax = b \). If it does not, we can then try a conventional linear system solver. Since the conventional solver would presumably be much slower, a failed attempt at sparsification would not dominate the the time it takes to solve the system, and it could greatly improve the running time on certain systems if it sparsifies the system (graph) correctly.

#### 4.2.1 Modification to algorithm

In section three, we analyzed a simplification to the Benczúr-Karger algorithm that ran in time \( O(m+n \log n) \). In this simplification, we sampled each edge with probability inversely proportional to a lower bound on the standard connectivity of the edge (as opposed to a lower bound on the strong connectivity). We found that the simplification did not suit our needs; it resulted in too many edges being included in the sparse graph. However, even with our analysis, it is still possible that the values of cuts are approximately preserved using lower bounds on the standard connectivity of each edge. Indeed, Benczúr and Karger could not find any counterexamples where cuts were not preserved using standard connectivity in place of strong connectivity.

With this in mind, we note that there is a minor simplification we can make to the algorithm that does not increase the number of edges included in the sparsified graph. This simplification reduces the running time by a factor of \( O(\log n) \) (to \( O(m \log^2 n) \)).

To compute \( S_k \) (a small set of edges including all edges of strong connectivity \( k \) or less), the Benczúr-Karger algorithm first uses the Nagamochi-Ibaraki algorithm to get a sparse-\( k \) certificate. It then pares down this sparse-\( k \) certificate to get an even sparser set. The problem is that the sparse-\( k \) certificate output by the Nagamochi-Ibaraki algorithm (and the resulting sparser set) is only required to contain edges of standard connectivity less than \( k \). For \( k \) to be a valid lower bound on the strong connectivity, \( S_k \) needs to contain all edges of strong connectivity less than \( k \). To rectify this problem, the Benczúr-Karger algorithm repeats this process \( \log n \) times, each time on all edges not in one of the sparser sets previously computed.

If we can using standard connectivity instead of strong connectivity still preserves cut values (conjectured above but unproven), we can immediately reduce the running time by a factor of \( \log n \) by only running the above process.
once to get each $S_k$. As argued in section 2.4, a sparse-$k$ certificate already includes all edges of standard connectivity $k$ or less. Note that this modification can only make our resulting graph sparser, as we are computing smaller $S_k$ sets. As argued in section 2.3, this can only produce lower sampling probabilities.

Our experiments below use this simplification to construct sparsifiers.

### 4.2.2 Performance on random graphs

We first examine the performance of the Benczúr-Karger algorithm on random graphs from the Erdős-Rényi $G(n, p)$ distribution. Recall that the algorithm produces graphs with $O(n \log n/\epsilon^2)$ edges when we sample with probability $p = \Theta(\ln n/k^* \epsilon^2)$. In practice, the algorithm produces graphs with at most $8cn \ln n$ edges if we sample with probability $c \ln n/k^*$. This means that in order to have meaningful results, we must experiment on graphs with at least $8c \ln n$ edges for each value of $c$ we sample with.

For $c \leq 4$, sampling with $p = 8/\sqrt{n}$ on graphs with large $n$ results in graphs where $8c \ln n < m$.

Let $L$ be the Laplacian of our original graph, and $L'$ be the Laplacian of our sparsified graph. In order to measure how the Benczúr-Karger algorithm sparsifies graphs according to our notion of sparsification, we would like to determine the maximum values of $a$ and $b$ (over all $x \in \mathbb{R}^n$) such that

$$ax^T L'x \leq x^T L x \leq b x^T L'x$$

Therefore, $a$ and $b$ are the following two quantities:

$$a = \max_{x \in \mathbb{R}^n} \frac{x^T Lx}{x^T L'x} \quad \text{and} \quad b = \max_{x \in \mathbb{R}^n} \frac{x^T L'x}{x^T Lx}$$

These quantities give us the relative error; we would like these quantities to be as close to 1 as possible. We measure these two quantities by solving the generalized eigenvalue problems $Lx = \lambda L'x$ and $L'x = \lambda' Lx$, and taking the maximum $\lambda$ and $\lambda'$ in each case. [5] proves that $\lambda$ and $\lambda'$ are equivalent to $a$ and $b$.

In our experiment, we generate a random unweighted graph according to $p = 8/\sqrt{n}$ and sparsify it with Benczúr-Karger. We perform several trials of this form for different values of $n$ (the number of vertices) and $c$ (the constant in our sampling probabilities).

Table 1 shows the results of this experiment. In Table 1, $c$, $a$, and $b$ are the quantities described above. $n$ is the number of vertices in the original (and sparsified) graph; $m$ and $m'$ are the number of edges in the original and sparsified graphs.

This data shows that the Benczúr-Karger algorithm does reasonably well on the above family of random graphs, considering the low $c$ values. The Benczúr-Karger algorithm guarantees that all cuts are approximated within a factor of

\[\text{See [5], section 2.1.}\]
Table 1: Results of Benczúr-Karger on random graphs

<table>
<thead>
<tr>
<th>c</th>
<th>n</th>
<th>Trial</th>
<th>m</th>
<th>m’</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>400</td>
<td>1</td>
<td>31855</td>
<td>5947</td>
<td>1.939019</td>
<td>1.595079</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>31788</td>
<td>5922</td>
<td>2.152610</td>
<td>1.588191</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>31838</td>
<td>5924</td>
<td>2.067598</td>
<td>1.632596</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td></td>
<td></td>
<td>2.053076</td>
<td>1.605289</td>
</tr>
<tr>
<td></td>
<td>600</td>
<td>1</td>
<td>58647</td>
<td>11505</td>
<td>1.744769</td>
<td>1.478684</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>58799</td>
<td>11780</td>
<td>1.697673</td>
<td>1.490435</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>58708</td>
<td>11675</td>
<td>1.763658</td>
<td>1.760081</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td></td>
<td></td>
<td>1.735367</td>
<td>1.576400</td>
</tr>
<tr>
<td></td>
<td>800</td>
<td>1</td>
<td>90395</td>
<td>18802</td>
<td>1.691632</td>
<td>1.409160</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>90957</td>
<td>19033</td>
<td>1.948453</td>
<td>1.505423</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>90680</td>
<td>18871</td>
<td>1.912678</td>
<td>1.505792</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td></td>
<td></td>
<td>1.850921</td>
<td>1.473458</td>
</tr>
<tr>
<td>2</td>
<td>400</td>
<td>1</td>
<td>32231</td>
<td>12175</td>
<td>1.597489</td>
<td>1.335151</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>32133</td>
<td>11877</td>
<td>1.483373</td>
<td>1.418213</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>32062</td>
<td>12019</td>
<td>1.521207</td>
<td>1.309035</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td></td>
<td></td>
<td>1.534023</td>
<td>1.354133</td>
</tr>
<tr>
<td></td>
<td>600</td>
<td>1</td>
<td>58574</td>
<td>23218</td>
<td>1.410923</td>
<td>1.273125</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>58229</td>
<td>23316</td>
<td>1.454310</td>
<td>1.316283</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>58554</td>
<td>23432</td>
<td>1.427786</td>
<td>1.290741</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td></td>
<td></td>
<td>1.431006</td>
<td>1.293383</td>
</tr>
<tr>
<td></td>
<td>800</td>
<td>1</td>
<td>90132</td>
<td>37574</td>
<td>1.387573</td>
<td>1.273116</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>90307</td>
<td>37671</td>
<td>1.367290</td>
<td>1.255708</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>90263</td>
<td>37584</td>
<td>1.373726</td>
<td>1.269888</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td></td>
<td></td>
<td>1.376196</td>
<td>1.265270</td>
</tr>
<tr>
<td>4</td>
<td>400</td>
<td>1</td>
<td>32077</td>
<td>24011</td>
<td>1.173130</td>
<td>1.140062</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>31923</td>
<td>24000</td>
<td>1.161748</td>
<td>1.128298</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>31908</td>
<td>24004</td>
<td>1.159952</td>
<td>1.148836</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td></td>
<td></td>
<td>1.164943</td>
<td>1.139065</td>
</tr>
<tr>
<td></td>
<td>600</td>
<td>1</td>
<td>58741</td>
<td>46878</td>
<td>1.134174</td>
<td>1.112224</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>58565</td>
<td>46999</td>
<td>1.132394</td>
<td>1.126891</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>58652</td>
<td>46993</td>
<td>1.158734</td>
<td>1.116450</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td></td>
<td></td>
<td>1.141767</td>
<td>1.118522</td>
</tr>
<tr>
<td></td>
<td>800</td>
<td>1</td>
<td>90187</td>
<td>75497</td>
<td>1.112751</td>
<td>1.100198</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>90618</td>
<td>75633</td>
<td>1.118031</td>
<td>1.091561</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>90434</td>
<td>75375</td>
<td>1.126335</td>
<td>1.096541</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td></td>
<td></td>
<td></td>
<td>1.119039</td>
<td>1.096100</td>
</tr>
</tbody>
</table>

1 ± ε if we sample with probabilities 15 ln n/k∗ ϵ². This guarantee means that for c = 1, 2, and 4, the algorithm guarantees our cuts are approximated within a factor of 1 ± ε for ϵ = 3.87, 2.739, and 1.936 respectively. Our results beat
these guarantees for not only cuts, but \( x^T L x \) for all \( x \in \mathbb{R}^n \). In particular, sampling with \( c = 4 \) resulted in errors small enough to be acceptable in solving the corresponding linear system.

However, the usefulness of these results is limited by the low values of \( c \) we tested with. For these graphs in particular, testing with \( c = 8 \) or higher would result in nearly every edge being picked, since \( 8cn \log n \) would exceed the number of edges in the original graph. Even sampling with \( c = 4 \) resulted in a sparsified graph with about three quarters of the original number of edges. While this is an improvement over the original graph, it is unlikely to significantly speed up solving the corresponding linear system.

To see how the Benczúr-Karger does for larger values of \( c \), we examine how the algorithm works for larger, non-random graphs. We run our algorithm for different values of \( c \) on the following 4 weighted graphs:

- **Airplane P5**: Airplane image segmentation graph, to the 5th power
- **Airplane P6**: Airplane image segmentation graph, to the 6th power
- **Grid P5**: Grid graph, to the 5th power
- **Grid P6**: Grid graph, to the 6th power

Our results appear in table 2.

**Table 2: Results of Benczúr-Karger on non-random graphs**

<table>
<thead>
<tr>
<th>Graph</th>
<th>( c )</th>
<th>( m' )</th>
<th>( a )</th>
<th>( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Airplane P5</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( n = 1376 )</td>
<td>1</td>
<td>22320</td>
<td>2.2255</td>
<td>1.7686</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>32441</td>
<td>1.3399</td>
<td>1.3725</td>
</tr>
<tr>
<td>( m = 84192 )</td>
<td>4</td>
<td>42066</td>
<td>1.204</td>
<td>1.1402</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>51225</td>
<td>1.1107</td>
<td>1.104</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>58600</td>
<td>1.0643</td>
<td>1.0542</td>
</tr>
<tr>
<td><strong>Airplane P6</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( n = 1376 )</td>
<td>1</td>
<td>30484</td>
<td>1.5955</td>
<td>1.6536</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>42803</td>
<td>1.3105</td>
<td>1.2719</td>
</tr>
<tr>
<td>( m = 123475 )</td>
<td>4</td>
<td>54140</td>
<td>1.2066</td>
<td>1.1667</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>64976</td>
<td>1.1619</td>
<td>1.1057</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>75257</td>
<td>1.0428</td>
<td>1.0364</td>
</tr>
<tr>
<td><strong>Grid P5</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( n = 1000 )</td>
<td>1</td>
<td>17943</td>
<td>1.2579</td>
<td>1.2783</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>23490</td>
<td>1.1406</td>
<td>1.1314</td>
</tr>
<tr>
<td>( m = 44772 )</td>
<td>4</td>
<td>30947</td>
<td>1.0897</td>
<td>1.0635</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>37113</td>
<td>1.0256</td>
<td>1.026</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>41413</td>
<td>1.0091</td>
<td>1.0089</td>
</tr>
<tr>
<td><strong>Grid P6</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( n = 1000 )</td>
<td>1</td>
<td>15424</td>
<td>1.5689</td>
<td>1.3757</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>23356</td>
<td>1.2496</td>
<td>1.2277</td>
</tr>
<tr>
<td>( m = 63512 )</td>
<td>4</td>
<td>31654</td>
<td>1.0884</td>
<td>1.1113</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>38763</td>
<td>1.0522</td>
<td>1.0527</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>47893</td>
<td>1.0272</td>
<td>1.0237</td>
</tr>
</tbody>
</table>
Figure 2 and Figure 3 indicate that Benczúr-Karger correctly sparsified these graphs, with error directly correlated with the constant we use in our sampling probabilities. The results here again beat the performance guarantee of the algorithm for all $x^T Lx \in \mathbb{R}^n$. In addition, the edge bounds for these graphs are low enough to make the sparsifiers useful in solving linear systems. To obtain a graph with 20% error in $x^T Lx$ or less, we can use $c = 4$ for Airplane P5, Airplane P6, and Grid P6, and $c = 2$ for Grid P5. The resulting graphs give over a 50% reduction in edges for the former three graphs, and a 66% reduction in the latter graph.

![Figure 2: Plot of $a$ vs. $c$ for the above graphs](image)

5 Conclusion

We examined the Benczúr-Karger algorithm in the context of sparsifying graphs. We analyzed a natural simplification of the algorithm briefly mentioned in their original paper, and find that it does not suit our needs for sparsification. We also empirically tested the Benczúr-Karger algorithm to see how it sparsifies certain graphs in practice. We found that even though graphs can be constructed that Benczúr-Karger does not sparsify correctly, the algorithm does correctly sparsify certain graphs. It correctly sparsified the graphs we tested it with, which included random graphs, algebraically-constructed graphs, and image segmentation graphs. The graphs we used are clearly not representative of graphs in general. However, our results suggest that it may be worthwhile to check the performance of Benczúr-Karger on graphs corresponding to certain families of
linear systems prior to solving them, as it could greatly speed up the process if it is successful.

6 Future work

It would be useful to test Benczúr-Karger on graphs corresponding to linear systems that often show up in practice. This would give us a better sense of how useful sparsifying with Benczúr-Karger is to solving linear systems in practice.

We can also further examine the Benczúr-Karger algorithm theoretically. While there exists examples where the algorithm fails for our purposes, there may be a way to modify the algorithm to handle these examples with high probability. In particular, we may be able to use Nikhil Srivastava’s result in [6] to relate the probabilities that Benczúr-Karger computes to effective resistances of edges. The strong connectivities themselves are not directly proportional to effective resistances. This can be seen by looking at the graph in figure 1, with $a = 1$. The effective resistance of the middle edge is $\Theta(n)$, while the effective resistance of every other edge is $O(1)$. On the other hand, the strong connectivities of all edges in the graph are $O(1)$ (as the minimum cut of any connected subgraph is 2).

However, it may still be possible to use the result in [6] to come up with a modification to the Benczúr-Karger algorithm that preserves $x^T L x$ for all graphs. Even if this is not possible, it could be the case that a good heuristic

Figure 3: Plot of $b$ vs. $c$ for the above graphs
relating to effective resistances, combined with the Benczúr-Karger algorithm, would yield a better heuristic for sparsification.

7 References


