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

Sourav Biswas, Quinlan Sykora

December 2022

1 Problem Introduction

We begin by introducing the concepts of cuts in graphs. Namely we define a cut C to be defined as a partition of a graph into two groups. More formally, a cut C is defined as two sets V_1 and V_2 such that

$$value(C) = \sum_{v_i \in V_1, v_j \in V_2} w_{ij} \tag{1}$$

Assuming the weight between all vertices that don't have edges is zero. In general, cuts are useful concepts when dealing with graphs. Many problem have solutions that are ultimately derived from their minimum cut. For example, maximum flows across a graph are directly related to the minimum cut that separates our initial two nodes. Minimum bisection problems are similar equivalent to finding the minimum cut, and most relevantly, the conductivity of the graph, and therefore its second eigenvalue, is also directly related to the minimum cut in a graph.

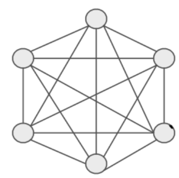
$$\frac{\nu_2}{2} \leq \Phi(G) \leq \sqrt{2\nu_2} \tag{2}$$

We furthermore observed in our first problem set how the diameter of the graph can be bounded by this conductivity. In summary, there are numerous uses of minimum cuts, and no truly flawless way to compute them.

2 Definitions

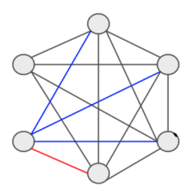
We will begin by defining some of the core concepts used in this work.

K-connectivity Firstly, we will define the notion of K-connectivity. We define a graph to be k-connected if any cut that fully partitions the graph sever at least k edges. In the context of weighted graphs, we require that the total weight of the edges severed have a total weight of k or more.



5-connected

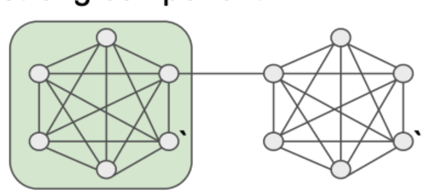
K-connected edge We extend the k-connectivity definition to individual edges in the graph. Namely, we say that an edge is k-connected if the minimum cut in the graph that includes that edge has a value of k. Essentially, the sparsest connection point that includes our given edge defines the k-connectivity of that edge.



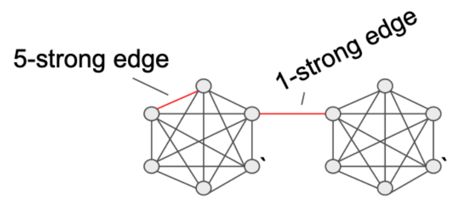
4-connected edge

K-strong component A K-strong component is defined as the maximal k-connected subgraph. This is essentially a group of k-connected vertices that are all k-connected to each other. The maximality means that we can't extend this cluster of points without including vertices that are not k-connected in the cluster. Note that the component can sometimes include the entire graph.

5-strong component



K-strong edge connectivity We define the K-strong edge connectivity (denoted as c_e) as the maximum value of k for which a strong component includes the given edge e. It can be intuitively thought of as the largest cluster that this particular edge is part of.



K-strong/K-weak edges An edge is called k-stong if it's k-strong edge connectivity c_e is equal to or larger than k. Conversely, the edge is called k-weak if it's c_e is less than or equal to k.

3 Edge Sampling

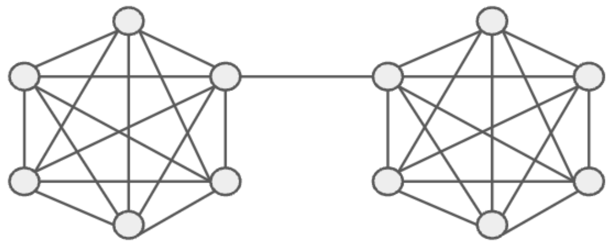
One of the first ideas that we have seen even in class is to randomly sample the edges. We showed previously that by changing the weight of the edges we can still have a high probability of approximating the minimum cut with an increasingly sparse graph. Let's begin by firstly sampling edges with the following probability:

$$p = \Theta(\frac{log(n)}{\epsilon^2 c}) \tag{3}$$

Where c is defined as the minimum cut in the graph. It has been proven in previous works [4] that by sampling according to this probability, we can effectively approximate the cuts in a graph to within a factor of ϵ

$$\epsilon = \sqrt{2(d+2)(\frac{M}{\hat{c}})ln(n)} \tag{4}$$

With a corresponding probability of success of $1 - O(n^{-d})$. However, the main issue with this sampling theorem is that we are dependent on the minimum cut not being too small. We note that in this definition, the sampling theorem only works by setting each edge to be equal to a random variable with a maximum value of M and sampling those random variables. For the edge case we can just make it equal to a Bernoulli distribution with a certain probability of being equal to 1.

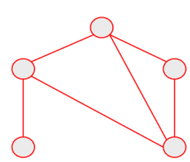


As seen in the figure above, the single edge connecting the two large clusters is effectively limiting the minimum probability that we can employ when sampling this graph. This is because the risk of that edge being sampled and prematurely cutting the graph would be too high otherwise. We therefore need a way to work around the sparsest regions of the graph when doing our sampling procedure.

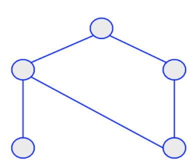
4 Sparse Certificates

Another important concept is a sparse certificate defined below:

> **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.*



Original graph G



H is a k-certificate with k=2

High level idea: A graph H with less edges than original graph G and preserves cuts of value $\leq k$. Therefore if we are interested in searching for cuts of value $\leq k$ then its more efficient to look in H instead of G due to the reduced number of edges in H.

Recall Sampling Theorem 1.14 from earlier. We can combine sampling and sparse certificates in order to find cuts in graphs.

To find the minimum cut of value c we can construct an $O(c)$ connectivity certificate which retains the same minimum cuts but reduces the edges in the graph to $O(nc)$. We can then use sampling with probability $\Theta((\log n)/c)$ to yield an $O(n \log n)$ edge graph with minimum cuts approximating the minimum cuts in the original graph. We can then use an algorithm to find minimum cuts noted in [1] to find min cuts in $\tilde{O}(n)$ time (note this is ignoring logarithmic terms).

However to find an $s-t$ cut with value v with a similar process involves sampling the $O(v)$ certificate of the original graph which reduces the edges to $\tilde{O}(nv/c)$. Note we cannot use a sparser certificate without possibly damaging the cuts we are interested in finding. Similarly we cannot lower the sampling probability otherwise the variance of cut values will become too large. Therefore we become dependent on the ratio v/c. However, we can avoid this issue by using non-uniform sampling.

5 Compression

High level idea: Need compression to pay more attention to less dense areas and less attention to clustered areas. After compressing the graph - we can run existing algorithms for max flow, sparsest cut etc.

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.*

Note that the expected value for each edge weight is 1 thus every cut has expected value after compression equal to its original value.

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 1.11 notes we can compress the graph and keep cuts within constant bounds in the new graph.

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

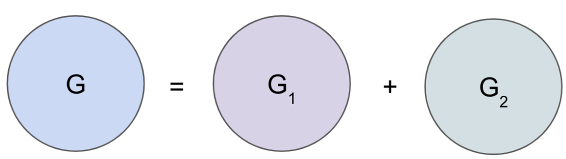
In the theorems a constant compression factor of $\rho_\epsilon = 16(d+2)(\ln n)/\epsilon^2$ is used for a given error bound ϵ.

Theorem 1.12 allows us to use ρ to get an approximation of $O(n \log n)$ for the number of edges in the compression graph.

The proof for this theorem is essentially as follows. We will seek to produce an upper and a lower bound on the total number of edges produced using this sampling policy and find that that these bounds and exponentially convergent.

We begin with the idea of showing that graphs of a certain size must have a k-strong component. More specifically, we find that any graph of size $k(n-1)$ nodes must have at least one k-strong component. The proof for it is essentially a simple use of induction.

We begin by finding the smallest graph with $k(n-1)$ nodes that does not have a k-strong component. You then take the nodes in the graph, find the smallest cut in the graph (which must be less than k) and determine that since this is the smallest graph without a k-strong component, it must have a value of less than k. We next determine that these two subgraphs created from the cut must also have less than $k(n'-1)$ nodes since otherwise they would contain the k-strong component. By adding together these components and counting up the edges, we find that we do not add back up to more than $k(n-1)$ edges, meaning there is not way of us having a graph of this size without it having a k-strong component.



# nodes	n	=	n_1	+	n_2			
# edges	$k(n-1)$	=	$< k(n_1-1)$	+	$< k(n_2-1)$	+	$< k$	

Now, we can return to our original formulation for the probability we use to sample each edge $p_e = min(1, \rho/c_e)$. Given this, we can create an inverse graph G' where each edge is the inverse weight of the k-strong edge connectivity

can create an inverse graph G' where each edge is the inverse weight of the k-strong edge connectivity.

$$weight(e', G') = c_e^{-1}, \quad \forall e' \in G \tag{5}$$

We then can calculate the expected volume of the edges in the original graph relative to the weight of the edges in this new graph

$$\mathbb{E}[Vol(G[p_e])] = \sum_{e \in E} p_e \leq \rho \sum_{e \in E} c_e^{-1} = \rho \mathbb{E}[Vol(G'[p_e])] \tag{6}$$

If this graph has a total edge weight of at most n, then this automatically gives is the total number of edges in the original graph by the above relation. We find that if the graph has more edges than n, then it must have a 1-strong component, and therefore must have a cut greater than 1. Let F be the component in the original graph that corresponds to this cut, and F' be the cut in the new graph G'. Given this fact, let $c_{e'}$ correspond to the minimum edge connectivity in this given cut. We get from that that the value of the cut $VAL(C, F) \leq c_{e'}$, and also that since this is the 1 strong cut that $VAL(C, F') > 1$. However, we can also prove that

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

Which is a contradiction and therefore there can't be a 1 strong component and the number of edges is bounded.

6 Error bounds

The next step in this proof is going to be proving that the error bounds hold for this kind of sampling. The first thing that is proven is that we can generate an initial error bound of $O(\epsilon log(m))$ where m is the total weights of all the edges in the graph. The way we will initially approach this problem is with subgraph edge sampling.

We will begin by sampling all edges that have a minimum cut that is less than 2.

$$c_e \leq 2 \tag{7}$$

This creates the first sparsest subgraph. We then repeat the process of all edges that have strong weights between 2^1 and 2^2.

$$2 \leq c_e \leq 4 \tag{8}$$

This can be repeated $log(m)$ times to encompass all edges in the graph. Each one of these groups we can effectively sample with a single probability to under-estimate the actual weight of each edge. Notice how each strong component is maintained in each of these graphs since we have all of the edges that make up each c_e strong component. Therefore we can use the previous sampling theorem to evaluate the error on each subgraph. This results in us achieving the error bound $O(\epsilon log(m))$.

Now we will prove the better error bound. We will begin by defining $r_e^{(i)} = 2^{-(j-i)}$ where j corresponds to the index of the power of 2 subgraph we are considering, namely G_i which contains all edges where $2^i \leq c_e \leq 2^{i+1}$. We can then define a graph H in such a way that for each edge we set it equal to a random variable

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

Note that this is the same format as the sampling theorem random variables. The one difference from our above attempt, is we set the edges that aren't in this "phase" equal to a small value instead of being equal to 1.

Next, we need to show that the strength of this component H of G is maintained even with this reducing of the edge weight by a factor of $r_e^{(i)}$. We begin by defining F_i to be the edges that have a strength of at least 2_i. We can use induction to show that this holds.

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

Where in this case we do reverse induction from the higher F_{i+1} and where H' is the 2^{i+1} strong component of this larger graph.

Now, we can describe the previous sampled graph using these new subgraphs, since we can depict every phase as a subtraction between two of these subgraphs. More specifically, we can combine this with the original sampling theorem to get:

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

This therefore means that by looking at the sampling in a different way we can get the error bound we were originally loooking for.

7 Finding Strong Connectivity

So now that we know that this sampling method works, we need to find a way to determine the strong connectivity of the graph. Unfortunately, this problem is fundamentally NP hard, and therefore doesn't have a nice convenient solution. However, one way we could get around this is by using a polynomial time algorithm that allows us to lower bound the strong connectivity of a graph. We could use the previously mentioned algorithm for subdividing the graph, but this results in too many edges by a factor of $log(m)$ since we again have the recursive division of the graph.

Therefore, we first consider sampling for the k-weak edges in the graph. This will later on tell us which regions of the graph are the sparsest and to more or less avoid sampling. According to a previous work [5], we can use the below algorithm to find these edges. This algorithm is able to run in $O(mlog(n))$ time.

```
procedure WeakEdges(G, k)

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

Given our above theorem on how many edges there are in k-weak graphs at most, we can find that this division into different graphs still results in $k(n-1)log(n)$ edges, which is again, not quite sparse enough for general purpose usage.

8 Graph Partitioning

One of the final proposed tools of this paper is to use a new algorithm called partition. This algorithm with output a new set of even sparser edges called the k-partition that we can use to produce the sparse graph.

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.*

We can derive such edge subsets by running the certificate algorithm seen above, and then replacing the edges that are still connected in the produced graph. When looking for the sparsest cuts, we can do this since these edges aren't actually cutting the graph and therefore aren't needed. This allows us to work with an even sparser subset of the edges in the graph itself. The algorithm can be seen below:

```
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)
```

The most important aspect of of this algorithm is that it finally allows us to operate without the dependency on the size of the graph in any way. Specifically, we find that the partition follows the following lemma:

> **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.*

Given this, we now know that we can produce the desired number of edges that allow us to compute cuts on a much sparser graphs. We can now consider all of the details together in the Estimation algorithm below.

Recall from earlier, if we have approximations $\tilde{c}_e$ for the strong connectivities c_e of edges in the graph then we could compress efficiently. The following algorithm allows us to find reasonably good lower bounds for the strong connectivities which we can then use for compression.

```
procedure Estimation(H, k)

input: subgraph H of G

if H contains edges
    E' ← WeakEdges(H, 2k)
    for each e ∈ E'
        c̃_e ← k
    for each connected component H' ⊂ H − E'
        Estimation(H', 2k)
```

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

From a high level, this algorithm recurses on components made of edges which are $2k$ strong in H and thus are $2k$ strong in G. Therefore in the next level of recursion when we set $\tilde{c_e} = 2k \leq c_e$ for edges which are $4k$ weak then we know $\tilde{c_e} = 2k \leq c_e \leq 4k$ which satisfies the lower bound we were aiming to achieve and noted in Lemma 4.5.

Thus with a call Estimation$(G, 1)$ we can estimate the strong connectivities in G.

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* $O(n \log n)$.

Lemma 4.6 states that the compressed graph using these estimates for strong connectivity has $O(n \log n)$ edges.

We can prove Lemma 4.6 by induction and show that Estimation(G, k) yields estimates $\tilde{c_e}$ where $\sum 1/\tilde{c_e} \leq 4(n-1)$. Showing this is sufficient since the number of edges then follows from Theorem 1.12.

Aside from the trivial base case of one vertex, we consider the induction case with Estimation(G, k). WeakEdges returns $4k(r-1)$ edges that split G into r components $G_1, ..., G_r$. Suppose these components have sizes $n_1, ..., n_r$ and WLOG $r > 1$. Note the recursive calls to Estimation on each G_i implies by induction we have strong connectivity estimates $\tilde{c_e}$ for each G_i where $\sum_{e \in G_i} 1/\tilde{c_e} \leq 4(n_i - 1)$.

Note for the $4k(r-1)$ edges not in the components G_i we assign their estimates $\tilde{c_e} = k$ yielding:

$$
\begin{aligned}
\sum_e 1/\tilde{c}_e &\leq \sum_i 4(n_i - 1) + \sum_{e \notin G_i} 1/k \\
&\leq \sum_i 4(n_i - 1) + 4k(r-1)/k \\
&\leq 4 \sum_i n_i - 4r + 4(r-1) \\
&\leq 4(n-1)
\end{aligned}
$$

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.*

The runtime in Lemma 4.8 is found by noting WeakEdges is called m times with a maximum $O(\log m)$ recursions since the max strong connectivity is m. Similarly for weighted graphs the maximum recursions is $O(\log nW)$ where W is the maximum edge weight.

9 Modifications for Weighted Graphs

To extend the approach to weighted graphs, a few modifications are needed. The main theorems 1.11 and 1.12 still hold for weighted graphs if weights are scaled up, rounded to integers, and edges of weight w are replaced with w copies of the same edge.

However note that the runtimes for the algorithms become unreasonable with the potentially large number of duplicated edges.

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 $(1 - 1/n)c_e \leq c'_e \leq c_e$.*

The intuition behind Lemma 5.1 is instead of considering the entire weighted graph, we can limit our focus to edges within a certain range and the strong connectivities in this graph approximate the strong connectivities in the original graph reasonably well.

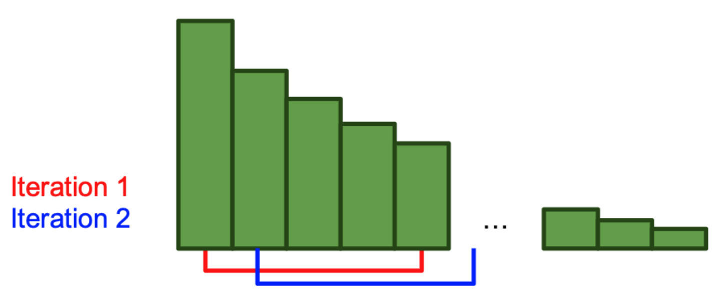
To prove Lemma 5.1 we can first prove $c'_e \leq c_e$. Proceed by contradiction and suppose after contracting we have an edge e in component H' with $c'_e > c_e$. Now consider the original component H in G which must have connectivity $\leq c_e$. Since H' was obtained by contracting all edges of weight over w^+ from H, then there is a cut with value $c_e < w^+$. Based on how H' is generated only removing edges with weights over w^+ then no edge across the cut in H is removed so it still has value c_e in H' which is a contradiction.

On the other hand, we can see the total weight of removed edges is
$\leq \binom{n}{2} w^- / n^3 < c_e/n$ which proves the lower bound $c'_e > c_e(1 - 1/n)$.

```
procedure WindowEstimation(G)

    Sort the edges in decreasing weight order
    initialize G' as an empty graph
    repeat
        W ← 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
```

We can then modify the Estimation algorithm to only focus within windows of relevant connectivity and call the previous Estimation algorithm within each window. Note that we can use a union-find data structure for adding and contracting edges. Since the window range is W/n^5 to W, each edge is in the window a maximum of 5 times. Therefore the total size of graphs passed to Estimation is $O(m)$. Therefore the runtime of WindowEstimation is $O(m \log^2 n)$



10 Applications

We can use the simplified compressed graph $G[p_e]$ with reduced edges along with existing algorithms to reduce the overall runtimes for different applications.

10.1 $s - t$ Min Cut

Note that minimum cut values are approximated in the compressed graph denoted by $VAL(C, G[p_e]) \leq (1 + \epsilon)v$. We also know that the compressed graph has $O(n \log n/\epsilon^2)$ edges. Therefore we can use a maximum flow algorithm [2] on the compressed graph $G[p_e]$ to find the minimum $s - t$ cut in $O(n^2 \log^2 n/\epsilon^2)$ time.

10.2 Sparsest Cut

To find the sparsest cut we are looking for the cut that minimizes the ratio:

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

Similar to before, we can use an existing algorithm [6] to find the sparsest cut $O(\log n)$ approximation in $O(m^2 \log m)$ time. Therefore by running this algorithm on the compressed graph we can find a similar $O(\log n)$ approximation but instead in $O(n^2 \log^3 n/\epsilon^4)$ time.

11 Future Steps

Compression is relying heavily on simplified versions of the original graph which approximate the values of cuts in the original graph. One future direction is to try and make these approximations exact. There are existing works [3] which have exact linear time solutions however this appears to only be for specific problems and will not work in the general case.

References

[1] Harold N. Gabow. "A matroid approach to finding edge connectivity and packing arborescences". In: *Journal of Computer and System Sciences, 50(2):259-273* (1995).

[2] Andrew V. Goldberg and Robert E. Tarjan. "A new approach to the maximum flow problem". In: *Journal of the ACM, 35:921-940* (1988).

[3] David R. Karger. "Minimum cuts in near-linear time". In: *In Proceedings of the 28th A CM Symposium on Theory of Computing. ACM, ACM Press* (1996).

[4] David R. Karger. "Random sampling in cut, flow, and network design problems". In: *In Proceedings of the 26th ACM Symposium on Theory of Computing, pages 648-657. ACM, ACM Press* (1994).

[5] Hiroshi Nagamochi and Toshihide Ibaraki. "Linear time algorithms for finding k-edge connected and k-node connected spanning subgraphs". In: *Algorithmic, 7:583–596* (1992).

[6] Clifford Stein Philip N. Klein and Eva Tardos. "Leighton-Rao might be practical: Faster approximation algorithms for concurrent flow with uniform capacities". In: *In Proceedings of the 22 ACM Symposium on Theory of Computing, pages 310- 321. ACM, ACM Press* (1990).