Seminar on Algorithms for Database Systems - Student Report

Graph Sparsification in the Semi-streaming Model[1]

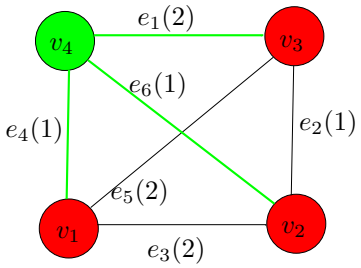
Demjan Grubic

Introduction

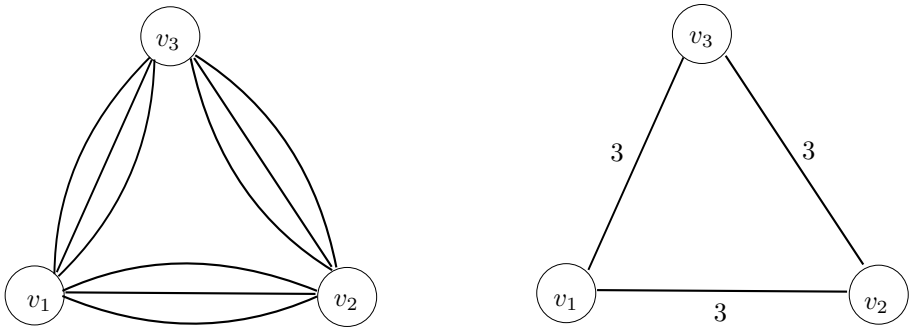
For the start, let's first explain what semi-streming model means and what is graph sparsification. After that we will see some application of graph sparsification and why is it important.

Semi-stremining model. The title of the paper is "Graph Sparsification in the Semi-streaming Model". First, we should note that it says semi-streaming and not streaming. In graph problems there is a linear space lower bound for even the simple problems such as determining the connectedness of a graph. In other words, we have to store at least all vertices of the graph. Because of that, this paper uses semi-streaming model, which means that we can store all vertices in memory, but we don't have enough memory to store all edges. Also, we are given edges one by one in arbitrary (and possibly adversarial) order. If we take a look at social network for example, the number of vertices (people) could be significantly lower than the number of edges in that graph, which means this model has a practical application.

Cut. In a graph $G(V, E)$, a cut is a partition of the vertices of a graph into two disjoint subsets. Any cut determines a cut-set, the set of edges that have one endpoint in each subset of the partition. We can represent a cut by two sets A and B, where $A, B \subset V, A \cup B = V$ and $A \cap B = \emptyset$, and a cut-set contains those edges that have one endpoint in A and other one in B. Value of the cut is a sum of weights of edges in cut-set. If the graph is unweighted, we assume that every edge has a weight 1. Throughout the paper, authors use different representation of a cut. The cut can be given by cut-set. In other word, by the set of edges that are in a cut-set. For a value of a cut C in a graph G we will use a notation $VAL(C, G)$. In the image below, a cut-set is given by green edges (weights of edges are in parenthesis) so we have a cut $C = \{e_1, e_4, e_6\}$ and $VAL(C, G) = 4$.



Sparsification. Let's take a look at two graphs in the image below.



What could we say about those graphs? Let's denote graph on the left side with G and graph on the right side with H. Graphs have same number of vertices, but H has less edges. Note that if you take any cut, value of that cut in G will be equal to the value of cut in H. This is idea behind sparsification. We want to sparsify graph in such a way that number of edges in a new graph is significantly less than in original graph, but keeping values of every cut inside some interval around value in original graph.

Definition 1. *H is a sparsification of a graph G if and only if for every cut C it holds $(1-\epsilon)VAL(C,G) \leq VAL(C,H) \leq (1+\epsilon)VAL(C,G)$. If H is a sparsification of a graph G we write that as $H \in (1 \pm \epsilon)G$.*

Goal. Our goal is to build an one pass algorithm, since we can't store all edges, which gives sparsification of the input graph. One pass algorithm means we don't have to go multiple times though streming data.

Natural question arises, why do we need this? Many useful algorithms are based on values of a cut, and one of the most popular is max flow algorithm. Also we can check connectivity of the graph with cuts. One more example involving huge graphs is image segmentation using graph cuts, where in one version of it we connect every pair of pixels with weighted edge and then use max flow - min cut to figure out what pixels are in the background and what is foreground.

Another question is why do we need such an algorithm in semi-streaming model. This paper was published in 2009 and before that we already knew quite many algorithms that produce sparsification of a given graph. All those other algorithms have to keep all edges in memory which isn't possible in some cases such as mentioned image segmentation. The ability of the algorithm to retain the most relevant information in main memory has been deemed critical.

Algorithms

Let's start with the simple algorithm.

Simple algorithm. We are given graph G and we build sparsified graph H. Add every edge from G in H with probability p and assign that edge a weight $1/p$ (with probability $(1-p)$ we don't add that edge to H). Now, consider any cut $C = \{e_{i_1}, e_{i_2}, \cdots, e_{i_l}\}$ in G. Let's denote set $\{i_1, i_2, \cdots, i_l\}$ with S. Value of cut C in G is the number of edges in the cut-set, i.e. $VAL(C,G) = |C| = l$. Let's compute expected value of the same cut in H. For every edge e_j in C, $j \in S$, let's denote with W_{e_j} random variable that denotes weight of edge e_j in graph H. Now, for every edge e_j we have expected weight $\mathbb{E}[W_{e_j}] = p \cdot 1/p = 1$. Now using linearity of expectation we have $\mathbb{E}[VAL(C,H)] = \mathbb{E}[\sum_{j \in S} W_{e_j}] = \sum_{j \in S} \mathbb{E}[W_{e_j}] = l$.

We have proved that in expectation every cut in H will have the same value as in G. Idea of this algorithm is clever, but in order to get sparsification it is not enough just to have same value of every cut in expectation. We have to get a result where with high probability value of every cut is around its expectation and also we have to get significantly less edges in sparsification graph.

In the paper [3] we can find theorem which is stronger then the one we will give now, but for our use we don't need so strong theorem.

Theorem 1. *Let G have minimum cut c. Build G' by including every edge of G with probability p and give it some weight. With high probability, every cut in G' is approximated around its expected value with factor $(1 \pm \epsilon)$, where $\epsilon = \Theta(\sqrt{\log n/pc})$.*

From here we can easily get following lemma.

Lemma 1. *Let G have minimum cut c. Build G' by including every edge of G with probability $p > \Theta(\log n/\epsilon^2 c)$ and give it a weight $1/p$, then with high probability every cut in G' is approximation of the same cut in G with factor $(1 \pm \epsilon)$.*

Since $p \gg 1/c$ and number of edges is around its expactation $m \cdot p$ with very high probability (Chernoff) we can see that in worst case when c is small we get only constant improvement in number of edges. Also in streaming model we can't know value of minimum cut in advance.

Previous lemma is intuitive. If our graph is a path, minimum cut is 1 and we have to keep all edges, since sparsified graph has to be connected also. On the other side, if our graph is clique, minimum cut is n and we probably don't need every edge.

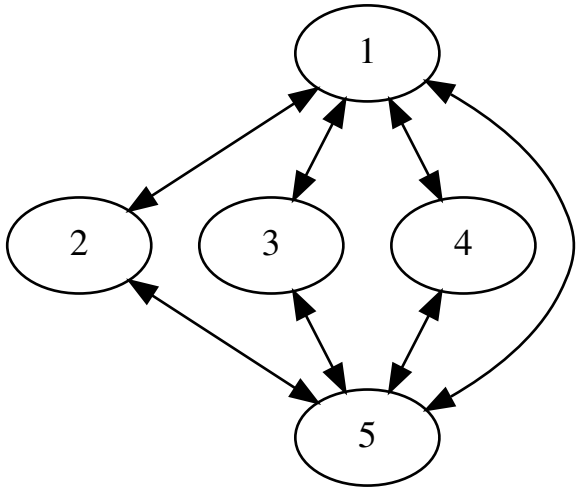
What this algorithm misses is the case when we have both, path and large clique. Minimum cut of the graph is 1, but we can divide it into 2 subgraphs and sparsify clique.

Based on previous example, we have to come up with a new way to sample different parts of a graph. The probability of sampling an edge should depend on the part of the graph around that edge.

Now, we will try to decompose our graph in parts where we could use our previous lemma in every part individualy. Let's define that local value for each edge.

Definition 2. *A graph is* **k-*strong connected*** *if and only if every cut in the graph has value at least k.* **k-*strong connected component*** *is a maximal node-induced subgraph which is k-strong connected. The* ***strong connectivity*** *of an edge e is the maximum k such that there exists a k-strong connected component that contains e. We will denote strong connectivity of an edge e in a graph G with c_e^G.*

We should note that strong connectivity is not the same as connectivity. Let's take a look at graphs in a figure below.



Edge $(1, 5)$ has (standard) connectivity 4, but it is only 2-strong connected because every subgraph that contains edge $(1, 5)$ have a cut with value 1 or 2. The main difference is that in (standard) connectivity we have to disconnect specific pair of vertices, while in strong connectivity we can disconent any pair of vertices.

Looking back at our example with path and clique we can see that strong connectivity for edges on the path is 1, but for edges in the clique is equal to the number of edges in the clique.

Now we will give an algorithm from paper [2] which uses k-strong connectivity of each edge.

Non-streaming algorithm. Below is the code of the algorithm that solves the problem with number of edges and outputs sparsified graph.

Algorithm 1 Benczur-Karger

Require: Graph $G = (V, E)$
Ensure: Sparsified graph H
1: compute the strong connectivity of edge c_e^G for all $e \in G$
2: $H = (V, \emptyset)$
3: **for** e in E **do**
4: $p_e = \min\{\rho/c_e, 1\}$
5: with probability p_e, add e to H with weight $1/p_e$
6: **end for**
7: **return** H

At first sight, this algorithms seems random. But let $\rho = \Theta(\log n/\epsilon^2)$. Now, we can see a relationship between $\rho/c_e = \Theta(\log n/\epsilon^2 c_e)$ and $p = \Theta(\log n/\epsilon^2 c)$ from lemma 1. If we would have a component where every edge have same c_e we could use lemma, because in that case we know that c_e is a minimum cut for that component.

What this algorithm does is that it decomposites a graph into components where for each component we know what is a minimum cut and we can use lemma 1. After that we can use union bound to get error bound for a whole graph.

In paper from Benczur and Karger [2] they proved following two theorems for mentioned algorithm.

Theorem 2. *Given ϵ and a corresponding $\rho = \Theta(\ln n/\epsilon^2)$, every cut in H has value $(1 \pm \epsilon)$ times its value in G with high probability.*

Theorem 3. *With high probability H has $\mathcal{O}(n \log n/\epsilon^2)$.*

To quickly compute strong connectivity for an edge is an open problem, but Benczur and Karger in the paper use good lower bound $c_e^{\tilde{G}} \leq c_e^G$ and proof still holds.

The problem with this algorithm is that it computes strong connectivities for each edge given G before building H and in semi-streaming model we have to build H while receiving edges from G. We will use very similar algorithm as last one, but instead of computing strong connectivity of every edge using G before building H, we will compute strong connectivity of an edge when we receive it, using graph H.

Semi-streaming Algorithm. Below is the code for algorithm from the paper.

Algorithm 2 Semi-stream sparsification

Require: Graph $G = (V, E)$
Ensure: Sparsified graph H
1: $H = (V, \emptyset)$
2: **for** e in E **do**
3: compute the connectivity c_e of e in H
4: $p_e = \min\{\rho/c_e, 1\}$
5: with probability p_e, add e to H with weight $1/p_e$
6: **end for**
7: **return** H

Even though last two algorithms are very similar, analysing last one is more complicated because of dependences that now arise because we compute connectivity in graph H and graph H is build on the fly.

In the paper two main theorems are proved. First one is about proving that H is a sparsification of G after running the algorithm from above and the second theorem is about number of edges in H after running the algorithm.

Theorem 4. *Given $\epsilon > 0$, H is a sparsification, that is $H \in (1 \pm \epsilon)G$, with high probability.*

Theorem 5. *If $H \in (1 \pm \epsilon)G$, H has $\tilde{\mathcal{O}}(n/\epsilon^2)$ edges.*

Proof of Theorem 4

We will give only brief description of the proof. First we decomposite a graph in k-strongly connected components. After doing so, in each component we know what is the minimum cut (k) and we know how many possible cuts are there. For every cut we have a theorem which computes an error with high probability so we can use union bound and sum over all cuts in k-strongly connected component. After doing so for each component we can use union bound once more to prove sparsification.

Only for the sake of completeness of a report we will state the part of the proof where graph is decomposesited.

Error Bound for H_i and H

Lemma 2. *The probability of i being the first integer such that $H_i \notin (1 \pm \epsilon)G_i$ is $\mathcal{O}(1/n^d m)$.*

In order to prove this lemma we need some additional definitions. Let G_i be a graph G after receiving first i edges. Also let $G_{i,j} = \{e : e \in G_i, 2^{j-1} \leq c_e^{(G_i)} < 2^j\}$. Each edge in $G_{i,j}$ has weight 1.

Also we introduce $F_{i,j} = \sum_{k \geq j} 2^{j-k} G_{i,k}$. As we can see from the formula, $F_{i,j}$ is composed of 2^{j-1}-strongly connected components, because we divide weight with corresponding number.

With those definitions, we can now give a proof.

Proof. If $H_j \in (1 \pm \epsilon)G_j$ for all $j < i$, $c_{e_j} \leq (1+\epsilon)c_{e_j}^{(G_i)}$. Let's now decompose H_i into components.

$$\begin{aligned} H_i &= \sum_{j=-\infty}^{\infty} H_{i,j} \\ &= \sum_{j=-\infty}^{\infty} (H_{i,j} + \frac{1}{2}F_{i,j+1}) - \sum_{j=-\infty}^{\infty} \frac{1}{2}F_{i,j+1} \end{aligned}$$

Now we will prove that $H_{i,j} + \frac{1}{2}F_{i,j+1}$ is a sparsification of $G_{i,j} + \frac{1}{2}F_{i,j+1} = F_{i,j}$.

As said before, $F_{i,j}$ consists of 2^{j-1}-strong connected componentes. For every $e \in G_{i,j}$, $c_e^{(G_i)} < 2^j$. So it is sampled with probability at least $p = \rho/(1+\epsilon)2^j$. If we consider one 2^{j-1}-strong connected component and set $\rho = 32((4+d)\ln n + \ln m)(1+\epsilon)/\epsilon^2$, by lemma for k-strongly connected components we have that every cut has error bound $\epsilon/2$ with probablility at least $1 - \mathcal{O}(1/n^{2+d}m)$. Since there are less than n^2 such distrinct strong connected components (number of edges), with probability at least $1 - \mathcal{O}(1/n^d m)$ it holds $H_{i,j} + (1/2)F_{i,j+1} \in (1 \pm \epsilon)F_{i,j}$ for every i, j. Hence,

$$\begin{aligned} H_i &\in \sum_{j=-\infty}^{\infty} (1 \pm \epsilon/2)F_{i,j} - \sum_{j=-\infty}^{\infty} \frac{1}{2}F_{i,j+1} \\ &\subseteq (2 \pm \epsilon)G_i - G_i \\ &= (1 \pm \epsilon)G_i \end{aligned}$$

Therefore, $Pr[(\forall j < i, H_j \in (1 \pm \epsilon)G_j) \wedge (H_i \notin (1 \pm \epsilon)G_i)] = \mathcal{O}(1/n^d m)$.

□

Now we can use union bound to finish the proof of the theorem $Pr[H \notin (1 \pm \epsilon)G] \leq \sum_{i=1}^{m} Pr[(\forall j < i, H_j \in (1 \pm \epsilon)G_j) \wedge (H_i \notin (1 \pm \epsilon)G_i)] = \mathcal{O}(1/n^d)$.

Proof of Theorem 5

They did typo doing this proof at the beginning and because of that their proof is somehow wrong. The lemma stated in the proof doesn't hold for weighted graphs and it is used for weighted graphs. We will write lemma as in [2]. The proof follows same steps with different numbers.

Here is a lemma from paper [2] that we are going to use.

Lemma 3. *If the total edge weight of a graph G is $k(n-1)$ or higher, there exists a k-strong connected components.*

Or we can interpret it in following way.

Corollary 1. *If there doesn't exist a k-strong connected component in graph G, total edge weight of a graph G is at most $k(n-1)$.*

Proof. Let G be a smallest counterexample with n vertices. Since in particular G is not k-strong connected, it must have a cut C of value less than k. Let us remove the edges of C and consider the two sides G_1 and G_2 with n_1 and $n_2 = n - n_1$ vertices respectively. Sinve G is a smallest counterexample and G_1 is not k-strong connected, 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 edge of 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 will now prove one helping lemma about total edge weight in H.

Lemma 4. *If $H \in (1 \pm \epsilon)G$ then total edge weight of H is at most $(1 + \epsilon)m$.*

Proof. Let C_v be a cut $(\{v\}, V - \{v\})$. Since $H \in (1 \pm \epsilon)G$ then it holds $VAL(C_v, H) \leq (1+\epsilon)VAL(C_v, G)$. Total edge weight of H is $(\sum_{v \in V} VAL(C_v, H))/2$ since each edge is counted for two such cuts. Similarly, G has $(\sum_{v \in V} VAL(C_v, G))/2 = m$ edges. Therefore, if $H \in (1 \pm \epsilon)G$ then total edge weight of H is at most $(1 + \epsilon)m$. □

We need one more lemma in order to be able to give a proof for our theorem, but let first define $E_k = \{e : e \in H \text{ and } c_e \leq k\}$. E_k is a set of edges that are sampled with $c_k \leq k$. We want to bound the total weight of edges in E_k.

Lemma 5. $\sum_{e \in E_k} w_H(e) \leq (n-1)(k + k/\rho + 1)$

Proof. Let H' be a subgraph of H that consists of edges in E_k. H' does not have $(k + k/\rho + 1)$-strong connected component. Suppose that it has. Then there exists the first edge e that creates a $(k + k/\rho + 1)$-strong connected component in H'. In that case, e must be in the $(k + k/\rho + 1)$-strong connected component. However, since weight e is at most k/ρ, that component is at least $(k+1)$-strong connected without e. This contradicts that $c_k \leq k$. Therefore, H' does not have any $(k+k/\rho+1)$-strong connected component. Now, using corollary 1 we have $\sum_{e \in E_k} w_H(e) \leq (n-1)(k + k/\rho + 1)$. □

Now we can finally finish our proof of theorem.

Proof. Algorithm as output give a graph H which is sparsification of G. Let's consider the worst case for number of edges. We know that if an edge has strong connectivity c_e then it's maximum weight in G is c_e/ρ. Since the total weight of H is the same, in worst case we have to sample as many as possible edges with lower strong connectivity because edge with lower strong connectivity has lower weight so we can take more of them.

So we have to prefer edges with lower strong connectivity. In that case we will take as many as possible edges from set E_1, then as many as possible from set E_2, and so on. Let's compute what is total weight difference between E_k and E_{k-1}.

$$\sum_{e \in E_k - E_{k-1}} w_H(e) = (n-1)k(1 + 1/\rho + 1/k) - (n-1)(k-1)(1 + 1/\rho + 1/(k-1)) = (n-1)(1 + 1/\rho)$$

As said before, we want to take as many as possible edges from sets E_k with lower k. We know that H has a total weight at most $(1+\epsilon)m$. So we have to see what is largest number k_m so we can take all E_k sets. Mathematically

$$k_m = \frac{(1+\epsilon)m}{(n-1)(1+1/\rho)}$$

Now we know how many of sets we should take and we know what is maximum weight for each set $E_{k+1} - E_k$. So in order to get largest possible number of edges we have to divide that maximum weight with smallest possible weight in that group which is k/ρ.

Then, total number of edges in H is

$$\begin{aligned}\sum_{i=1}^{k_m} \frac{(n-1)(1+1/\rho)}{i/\rho} &= (n-1)(\rho+1)\sum_{i=1}^{k_m}\frac{1}{i} \\ &= \mathcal{O}(n(\rho+1)\log(k_m)) \\ &= \mathcal{O}(n\rho(\log m - \log n)) \\ &= \tilde{\mathcal{O}}(n/\epsilon^2)\end{aligned}$$

□

As last theorem in the paper they proved that lower bound space required in order to sparsify every cut of a graph is $\Omega(n(\log n + \log \frac{1}{\epsilon}))$. We won't prove it here, but they built an example of a graph where mentioned holds.

Conclusion

What this paper misses to say is time complexity of the algorithm. They mention that it is one pass and what is space complexity, but they don't say anything about time complexity.

In the paper they worked with corrently calculated c_e, but it is an open problem to calculate edge strong connectivity. It doesn't change the proof, but they would have to use lower bound as in paper [2]. In their solution they would have to use similar algorithm as in paper [2] and that would add time complexity $\mathcal{O}(n\log^2(n)/\epsilon^2)$ in each interation of for loop. Overall time complexity would be $\mathcal{O}(mn\log^2(n)/\epsilon^2)$. There are better algorithm for computing sparsification of a graph, but this one should be used when we don't have enough memory for all edges.

To conclude, we presented a one pass semi-streaming algorithm for the adversarially ordered data stream model which uses $\tilde{\mathcal{O}}(n/\epsilon^2)$ edges to provide ϵ error bound for cut values with high probability.

References

[1] K. J. Ahn and S. Guha. *Graph sparsification in the semi-streaming model.* International Colloquium on Automata, Languages and Programming, pages 328 - 338, 2009.

[2] András A. Benczúr and David R. Karger. *Approximating s-t minimum cuts in O(n2) time.* In STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, pages 47 - 55, New York, NY, USA, 1996. ACM.

[3] David R. Karger. *Random sampling in cut, flow, and network design problems.* STOC '94 Proceedings of the twenty-sixth annual ACM symposium on Theory of computing, Pages 648 - 657, 1994.