# Not-As-Fast Cut Sparsification of Weighted Graphs

Reading Group Algorithms Companion Paper

Tijn de Vos

January 27, 2022

**Abstract**

In this paper, we consider the techniques for computing cut sparsifiers of weighted graphs using maximum spanning forest packings, as introduced by Forster and de Vos [FdV21]. In particular, we see how the results change under two simplifications. First, we apply the cut sparsification framework of Fung et al. [FHHP11] directly to maximum spanning forest indices to obtain a cut sparsifier of size $O(n\log(n)\log(m/n)/\epsilon^2)$ in $O(m\log(n))$ time. Second, we retrieve a linear time algorithm when we only consider unweighted graphs.

## 1 Introduction

The goal of cut sparsification is to compute for a given (weighted) graph $G$ a subgraph $H$, with new weights, such that with high probability the value of all cuts is preserved. In this paper, we study the work of Forster and de Vos [FdV21], who show that the techniques of Fung et al. [FHHP11] for cut sparsifiaction of unweighted graphs can be applies to weighted graphs when a new 'connectivity estimator' is used. Fung et al. also provide a general framework for cut sparsification, where a connectivity estimator is a function $\lambda\colon E \to \mathbb{R}$ on the edges that satisfies certain conditions. Forster and de Vos use a maximum spanning forest (MSF) index for the connectivity estimator. However, they do not apply it directly to the framework, but in a more complicated fashion – based on the unweighted sparsification algorithm of Fung et al.

In this paper, we present two simplifications of [FdV21]: we directly use the MSF index in framework of [FHHP11], and we see how the weighted algorithm simplifies when we only consider unweighted graphs. The former still improves on the prior work to [FdV21], but is less strong than [FdV21] itself.

### 1.1 Review

For convenience, we restate the most important definitions and theorems from [FdV21]. First of all, we formally define cuts. We define a set of edges $C \subseteq E$ to be *cut* if there exists a partition of the vertices $V$ in two non-empty subjets $A$ and $B$, such that $C$ consists of all edges with one endpoint in $A$ and the other endpoint in $B$. The weight of the cut is the sum of the weights of the edges of the cut: $w_G(C) = \sum_{e\in C} w_G(e)$. The *minimum cut* is defined as the cut with minimum weight. We say that a (reweighted) subgraph $H \subseteq G$ is a $(1 \pm \epsilon)$-*cut sparsifier* for a weighted graph $G$ if for every cut $C$ in $H$, its weight $w_H(C)$ is within a multiplicative factor of $1 \pm \epsilon$ of its weight $w_G(C)$ in $G$. A key concept in the realm of cut sparsification is the connectivity of an edge.

**Definition 1.1.** *Let $G = (V, E)$ be a graph, possibly weighted. We define the* connectivity *of an edge $e = (u, v) \in E$ to be the minimal weight of any cut separating $u$ and $v$. We say that $e$ is $k$-*heavy *if it has connectivity at least $k$. For a cut $C$, we define the $k$-*projection *of $C$ to be the $k$-heavy edges of the cut $C$.*

Next, we give the definition of the MSF index as it appears in [FdV21].

**Definition 1.2.** *Let $G = (V, E)$ be a weighted graph. We say $\mathcal{F} = \{F_1, \dots, F_M\}$ is an $M$-*partial maximum spanning forest packing *of $G$ if for all $i = 1, \dots, M$, $F_i$ is a maximum spanning forest in $G \setminus \bigcup_{j=1}^{i-1} F_j$. If we have that $\bigcup_{i=1}^{M} F_i = G$, then we call $\mathcal{F}$ a* (complete) maximum spanning forest packing *of $G$. Moreover, for $e \in E$ we denote the* MSF index *of $e$ (w.r.t. $\mathcal{F}$) by $f_e$, i.e., $f_e$ is the unique index such that $e \in F_{f_e}$.*

Note that we do not demand the $F_i \in \mathcal{F}$ to be non-empty, as this suits notation best in our applications. Also note that a (partial) MSF packing is fully determined by the MSF indices. The next theorem shows the computation time of the MSF index.

**Theorem 1.3** (See [FdV21], Theorem 3.1 and Theorem 3.3)**.** *Let $G = (V, E)$ be a graph, where we allow parallel edges but no self-loops, and we suppose $m \leq n^2$. Suppose we have weights $w \colon E \to \{1, \dots, n^c\}$ for some $c \geq 0$. Then, for any $M > 0$, there exists an algorithm that computes an $M$-partial MSF packing in $O(m \cdot \min\{\alpha(n)\log(M) + c, \log(n)\})$ time.*

Here $\alpha(\cdot)$ refers to the functional inverse of Ackermann's function, for a definition see e.g. [Tar75].

Next, we review the general framework for cut sparsification as presented in [FHHP19]. Let $G = (V, E)$ be a graph with integer weights, and let $\epsilon \in (0, 1)$, $c \geq 1$ be parameters. Given a parameter $\gamma$ (possibly depending on $n$) and an integer-valued parameter $\lambda_e$ for each $e \in E$. We obtain $G_\epsilon$ from $G$ by independently *compressing* each edge $e$ with parameter

$$p_e = \min\left(1, \frac{16(c+7)\gamma\ln(n)}{0.38\lambda_e\epsilon^2}\right).$$

Compressing an edge $e$ with weight $w(e)$ consists of sampling $r_e$ from a binomial distribution with parameters $w(e)$ and $p_e$. If $r_e > 0$, we include the edge in $G_\epsilon$ with weight $r_e/p_e$.

In the following we describe a sufficient condition on the parameters $\gamma$ and $\lambda_e$ such that $G_\epsilon$ is a $(1 \pm \epsilon)$-cut sparsifier for $G$ with probability at least $1 - 4/n^c$. Hereto we partition the edges according to their value $\lambda_e$:

$$\Lambda := \lfloor \log \max_{e \in E}\{\lambda_e\} \rfloor;$$
$$R_i := \{e \in E : 2^i \leq \lambda_e \leq 2^{i+1} - 1\}.$$

Let $\mathcal{G} = \{G_i = (V, E_i) : 1 \leq i \leq \Lambda\}$ be a set of integer-weighted subgraphs such that $R_i \subseteq G_i$. Moreover suppose that $w_{G_i}(e) \geq w_G(e)$ for each $e \in R_i$. For a given set of parameters $\Pi = \{\pi_1, \dots, \pi_\Lambda\} \subseteq \mathbb{R}^\Lambda$, we define

- $\Pi$-*connectivity*: each edge $e \in R_i$ is $\pi_i$-heavy in $G_i$;
- $\gamma$-*overlap*: for any cut $C$,
$$\sum_{i=0}^{\Lambda} \frac{e_i^{(C)} 2^{i-1}}{\pi_i} \leq \gamma \cdot e^{(C)},$$
where $e^{(C)} = \sum_{e \in C} w_G(e)$ and $e_i^{(C)} = \sum_{e \in C \cap E_i} w_{G_i}(e)$.

The following theorem shows that compressing with parameters adhering to these conditions gives a cut sparsifier with high probability.

**Theorem 1.4** (See [FHHP19, Theorem 1.14])**.** *Fix the parameters $\gamma$ and $\lambda_e$ for each edge $e$. If there exists $\mathcal{G}$ satisfying $\Pi$-connectivity and $\gamma$-overlap for some $\Pi$, then $G_\epsilon$ is a $(1 \pm \epsilon)$-cut sparsifier for $G$, with probability at least $1 - 4/n^c$, where $G_\epsilon$ is obtained by edge compression using parameters $\gamma$ and $\lambda_e$'s.*

## 2 Applying the Framework Directly

In this section, we show that using the MFS index directly in the framework of Fung et al. [FHHP19] also gives us the following result.

**Theorem 2.1.** *There exists an algorithm that, given a polynomially weighted graph $G = (V, E)$, and freely chosen parameter $\epsilon > 0$, computes a graph $G_\epsilon$, which is a $(1 \pm \epsilon)$-cut sparsifier for $G$ with high probability. The algorithm runs in time $O(m \log(n))$ and the number of edges of $G_\epsilon$ is $O\big(n \log(n)/\epsilon^2 \log(m/n)\big)$.*

This is not as sparse as the result of Forster and de Vos [FdV21], but considerably simpler, both in the algorithm itself and in the analysis.

The algorithm consists of the following two steps

1. Compute MSF indices upto $M = n$ in time $O(m \log(n))$.
2. Apply the sparsification framework (Theorem 1.4) with $\lambda_e = f_e \cdot w_e$.

First, let us consider the size of the sparsifier. We have

$$\sum_e w_e/\lambda_e = \sum_e 1/f_e.$$

This is the biggest when the edges are distributed such that the $1/f_e$ are as large as possible. Each forest holds at most $n$ edges, dividing all edges over the first $m/n$ forests gives an upper bound of

$$n \sum_{i=1}^{m/n} 1/i \leq n \log(m/n),$$

which results in expected size $O(cn \log(n)/\epsilon^2 \log(m/n))$. By the same techniques as presented in [FdV21], we can show that the same bound holds with high probability.

To show that this is indeed a sparsifier, we are proving a slightly simpler version of Theorem 4.3 in [FdV21].

Let $\mathcal{F} = T_1, T_2, \ldots$ be an MSF packing of $G$. We define $F_j = T_{2^j} \cup \cdots \cup T_{2^{j+1}-1}$. We analyze what happens if for $e \in F_j$ we set $\lambda'_e = 2^{j-1} w(e)$. As we have $2^j \leq f_e < 2^{j+1}$, we obtain $f_e w(e)/4 \leq 2^{j-1} w(e) \leq f_e w(e)$, thus $\lambda_e/4 \leq \lambda'_e \leq \lambda_e$. Hence this gives the same outcome for our actual choice of $\lambda_e$, up to constant factors.

The proof is very similar to the proof of Theorem 4.3 in [FdV21], with slight adaptations.

We have to provide a set of subgraphs $\mathcal{G}$ and a set of parameters $\Pi$ such that $\Pi$-connectivity and $\gamma$-overlap are satisfied.

To explore the connectivity of edges in $R_i := \{e \in E : 2^i \leq \lambda_e \leq 2^{i+1} - 1\}$ we partition these sets as follows:

$$R_{j,k} := \{e \in F_j : 2^k \leq w(e) \leq 2^{k+1} - 1\}.$$

Now we have that $R_i$ is a union of different $R_{j,k}$ as follows: $R_i = \bigcup_{j=1}^{\min(\lfloor i/2 \rfloor, \Gamma)} R_{j,i-j}$. We will view these edges in the subgraph:

$$E_{j,k} := \bigcup_{k'=k}^{\infty} 2^{\Gamma - j + 2 + \Lambda - k'} R_{j,k'},$$

where $\Gamma = \lfloor \log(n) \rfloor$, as we have at most $n$ spanning forests.

**Lemma 2.2.** *Each edge $e \in R_{j,k}$ is $\pi := 2^{\Gamma+\Lambda}$-heavy in $(V, E_{j,k})$.*

*Proof.* Fix $e \in R_{j,k}$. This edge is $2^{j-1} w(e) \geq 2^{j+k}$-heavy in $\{e \in F_{j-1} : w(e') \geq w(e)\}$. Hence $e$ is $2^{j+k-1}$-heavy in $\{e' \in F_{j-1} : w(e') \geq 2^k\}$. We can rescale this: $e$ is $\big(2^{j+\Lambda-1}\big)$-heavy in $\bigcup_{k'=k}^{\infty} 2^{\Lambda - k'} \cdot \{e' \in F_{j-1} : 2^{k'} \leq w(e') \leq 2^{k'+1} - 1\}$. By rescaling again we get the result. □

Now we take all weight classes together to find the set of subgraphs $\mathcal{G}$ for which $\Pi$-connectivity is satisfied.

**Corollary 2.3.** *Each edge in $e \in R_i$ is $2^{\Gamma+\Lambda}$-heavy in $G_i = (V, E_i)$, with $E_i := \bigcup_{j=1}^{\min(\lfloor i/2 \rfloor, \Gamma)} E_{j,i-j}$.*

*Proof.* Note that $e \in R_i$ satisfies $2^i \leq 2^{2j} w(e) \leq 2^{i+1} - 1$ if $e \in F_j$. Hence $e \in R_{j,k}$ with $2j + k = i$. We are only considering edges in $F_j$ with $1 \leq j \leq \Gamma$, thus we have $R_i = \bigcup_{j=1}^{\min(\lfloor i/2 \rfloor, \Gamma)} R_{j,i-j}$, hence the claim follows directly from Lemma 2.2. $\square$

It remains to show that $\gamma$-overlap is satisfied.

**Lemma 2.4.** *For any cut $C$,*

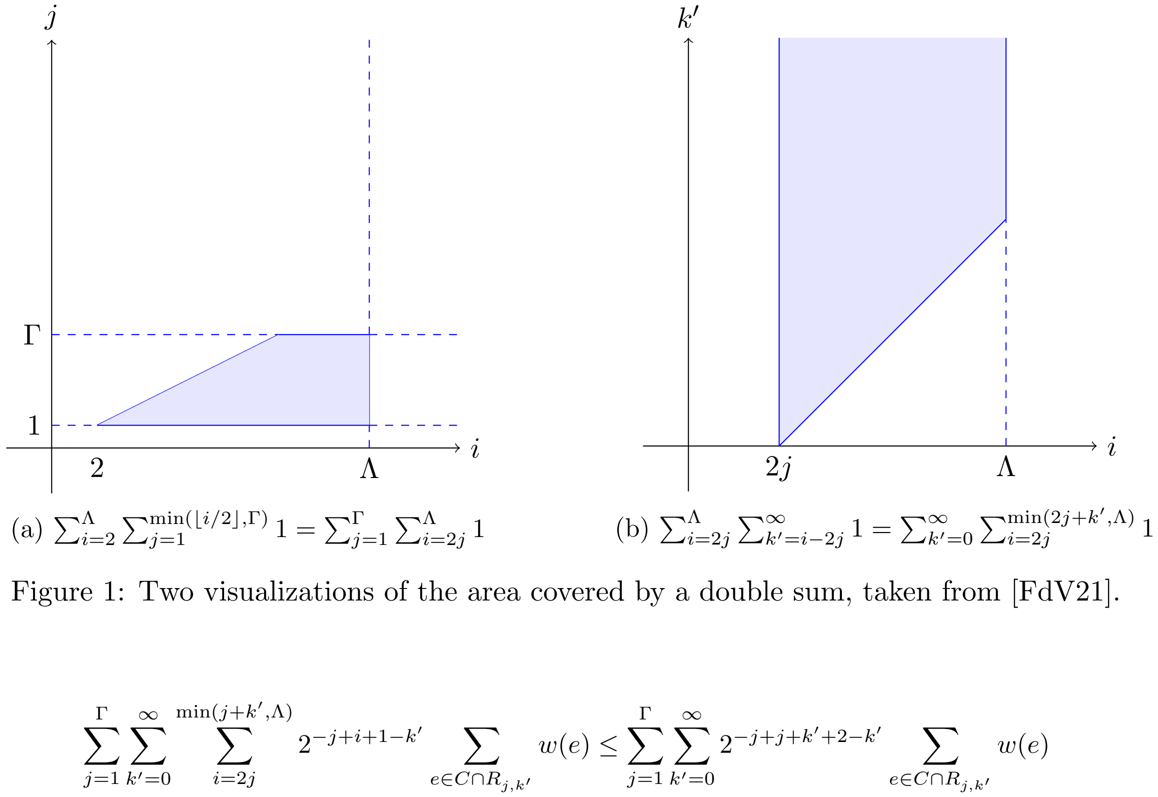
$$\sum_{i=0}^{\Lambda} \frac{e_i^{(C)} 2^{i-1}}{2^{\Gamma+\lambda}} \leq 4 \cdot e^{(C)},$$

*where $e^{(C)} = \sum_{e \in C} w_{G_S}(e)$ and $e_i^{(C)} = \sum_{e \in C \cap E_i} w_{G_i}(e)$.*

*Proof.* This comes down to a technical computation:

$$\begin{aligned}
\sum_{i=0}^{\Lambda} \frac{e_i^{(C)} 2^{i-1}}{2^{\Gamma+\lambda}} &= \sum_{i=0}^{\Lambda} \sum_{j=1}^{\min(\lfloor i/2 \rfloor, \Gamma)} \sum_{k'=i-j}^{\infty} \frac{2^{\Gamma-j+2+\Lambda-k'}}{2^{\Gamma+\Lambda}} \sum_{e \in C \cap R_{j,k'}} w(e) \\
&= \sum_{j=1}^{\Gamma} \sum_{i=2j}^{\Lambda} \sum_{k'=i-j}^{\infty} 2^{-j+i+1-k'} \sum_{e \in C \cap R_{j,k'}} w(e) \\
&= \sum_{j=1}^{\Gamma} \sum_{k'=0}^{\infty} \sum_{i=2j}^{\min(j+k', \Lambda)} 2^{-j+i+1-k'} \sum_{e \in C \cap R_{j,k'}} w(e),
\end{aligned}$$

see Figure 1 for a visual argument of interchanging the sums.



(a) $\sum_{i=2}^{\Lambda} \sum_{j=1}^{\min(\lfloor i/2 \rfloor, \Gamma)} 1 = \sum_{j=1}^{\Gamma} \sum_{i=2j}^{\Lambda} 1$ (b) $\sum_{i=2j}^{\Lambda} \sum_{k'=i-2j}^{\infty} 1 = \sum_{k'=0}^{\infty} \sum_{i=2j}^{\min(2j+k', \Lambda)} 1$

Figure 1: Two visualizations of the area covered by a double sum, taken from [FdV21].

$$\begin{aligned}
\sum_{j=1}^{\Gamma} \sum_{k'=0}^{\infty} \sum_{i=2j}^{\min(j+k', \Lambda)} 2^{-j+i+1-k'} \sum_{e \in C \cap R_{j,k'}} w(e) &\leq \sum_{j=1}^{\Gamma} \sum_{k'=0}^{\infty} 2^{-j+j+k'+2-k'} \sum_{e \in C \cap R_{j,k'}} w(e) \\
&= 4e^{(C)}.
\end{aligned}$$

$\square$

Finally, let us consider what happens when we call this algorithm repeatedly. As for example seen in Theorem 3.2 above, this can lead to a better results. Unfortunately, in this case we can not get the same sparsity in the same time. The reason hereto is that we have a factor $\log(m/n)$ in the running time, rather than a factor $\log(m/(n\log(n)/\epsilon^2))$, which is the desired size. This means that the number of edges does not decrease quickly enough to get the sparser output in (asymptotically) the same running time. With some extra time we can get to size $O(n\log(n)/\epsilon^2)$, namely in time $O\left(\left(m+\frac{n\log^2(n)\log\log(n)}{\epsilon^4}\right)\log(n)\right)$. We omit the proof, which is very similar to Theorem 3.2.

As a final remark, note that using the framework directly, we need to compute MSF indices up to depth $n$, where Forster and de Vos only need to go to depth $m/n$. This is reflected in the running time.

## 3 Cut Sparsification for Unweighted Graphs

In this section, we investigate what happens when we apply the algorithm of Forster and de Vos [FdV21] to unweighted graphs. This means that *any* forest packing is a maximal spanning forest packing. Hence we can use the linear time algorithm of Nagamochi and Ibaraki [NI92a, NI92b] for this step of the algorithm. Note that the full algorithm (see Appendix A) is highly similar to the algorithm for unweighted graphs of Fung et al. [FHHP19], but not quite the same. The difference is that Fung et al. use some graph contractions to get a sparser result. We do not get this sparser result initially, but show that we can repeatedly apply the algorithm to get the same outcome. The following theorem is the analogue of Theorem 4.1 in [FdV21], but with a better time complexity since it is restricted to unweighted graphs.

**Theorem 3.1.** *There exists an algorithm that, given an unweighted graph $G=(V,E)$, and freely chose parameter $\epsilon>0$, computes a graph $G_\epsilon$, which is a $(1\pm\epsilon)$-cut sparsifier for $G$ with high probability. The algorithm runs in time $O(m)$ and the number of edges of $G_\epsilon$ is $O\big(n\left(\log(n)/\epsilon^2\right)\log\left(m/(n\log(n)/\epsilon^2)\right)\big)$.*

To be precise, we give an algorithm where the given bounds on both running time and size of the sparsifier hold with high probability. By a simply halting when the running time exceeds the bound, and outputting an empty graph if we exceed the size bound, this gives the result above.

As said, to achieve a better bound on the size of the sparsifier, we repeatedly apply this theorem to the input graph to obtain the following result.

**Theorem 3.2.** *There exists an algorithm that, given an unweighted graph $G=(V,E)$, and freely chosen parameter $\epsilon>0$, computes a graph $G_\epsilon$, which is a $(1\pm\epsilon)$-cut sparsifier for $G$ with high probability. The algorithm runs in time $O(m)$ and the number of edges of $G_\epsilon$ is $O\big(n\log(n)/\epsilon^2\big)$.*

*Proof.* We obtain this result by repeatedly applying the algorithm from Theorem 3.1, for a total of $k:=\log^*\left(\frac{m}{n\log(n)/\epsilon^2}\right)$ times. In iteration $i$, we set $\epsilon_i:=\epsilon/2^{k-i+2}$ and denote the output of this iteration by $G_i$. This means that $G_i$ is a $(1\pm\epsilon/2^{k-i+2})$-cut sparsifier for $G_{i-1}$. In total, we see that $G_\epsilon:=G_k$ is a $(1\pm\epsilon)$-cut sparsifier for $G$ since

$$\prod_{i=1}^{k}(1+\epsilon/2^{k-i+2})\leq\exp\left(\sum_{i-1}^{k}\log(1+\epsilon/2^{k-i+2})\right)\leq\exp\left(\sum_{i-1}^{k}\epsilon/2^{k-i+2}\right)$$
$$\leq\exp\left(\epsilon\sum_{j=2}^{\infty}2^{-j}\right)=\exp(\epsilon/2)\leq1+\epsilon,$$

as $\epsilon < 1$, and

$$\begin{aligned}
\prod_{i=1}^{k}(1-\epsilon/2^{k-i+2}) &\geq \prod_{j=0}^{\infty} 1-\frac{\epsilon/4}{2^j} & &\geq (1-\epsilon/8)\prod_{j=1}^{\infty} 1-\frac{\epsilon/4}{j^2} \\
&= (1-\epsilon/8)\frac{\sin(\pi\sqrt{\epsilon}/2)}{\sqrt{\epsilon}/2} & &\geq (1-\epsilon/8)(1-\pi^2/24\epsilon) \\
&\geq 1-(1/8+\pi^2/24)\epsilon+\frac{\pi^2}{192}\epsilon^2 & &\geq 1-\epsilon.
\end{aligned}$$

Since $k = \log^*\left(\frac{m}{n\log(n)/\epsilon^2}\right) = O(\log^*(n))$, all bounds hold with high probability simultaneously, and thus the end result holds with high probability.

Now for the size bound, we have that

$$m_i := |E(G_i)| \leq C \cdot \left(\frac{n\log(n)}{\epsilon^2}4^{k-i+2}\log\left(\frac{m_{i-1}}{n\log(n)/\epsilon^2}\right)\right),$$

for some constant $C > 0$, where we denote $m_0 := m$. We will show by induction that

$$m_i \leq C \cdot \left(\frac{n\log(n)}{\epsilon^2}4^{k-i+2} \cdot 2\log^{(i)}\left(\frac{m}{n\log(n)/\epsilon^2}\right)\right),$$

which means in particular that $m_k = O\big(n\log(n)/\epsilon^2\big)$. The claim for $m_1$ is immediate. Suppose it holds for $i-1$, then

$$\begin{aligned}
m_i &\leq C \cdot \left(\frac{n\log(n)}{\epsilon^2}4^{k-i+2}\log\left(\frac{m_{i-1}}{n\log(n)/\epsilon^2}\right)\right) \\
&\leq C \cdot \left(\frac{n\log(n)}{\epsilon^2}4^{k-i+2}\log\left(C\cdot 4^{k-i+3}\cdot 2\log^{(i-1)}\left(\frac{m}{n\log(n)/\epsilon^2}\right)\right)\right) \\
&= C \cdot \left(\frac{n\log(n)}{\epsilon^2}4^{k-i+2}\left((k-i)\log(4)+\log(C\cdot 2^7)+\log^{(i)}\left(\frac{m}{n\log(n)/\epsilon^2}\right)\right)\right) \\
&\leq C \cdot \left(\frac{n\log(n)}{\epsilon^2}4^{k-i+2}\cdot 2\log^{(i)}\left(\frac{m}{n\log(n)/\epsilon^2}\right)\right),
\end{aligned}$$

since

$$\begin{aligned}
(k-i)\log(4)+\log(C\cdot 2^7) &= \left(\log^*\left(\frac{m}{n\log(n)/\epsilon^2}\right)-i\right)\log(4)+\log(C\cdot 2^7) \\
&= \log^*\left(\log^{(i)}\left(\frac{m}{n\log(n)/\epsilon^2}\right)\right)\log(4)+\log(C\cdot 2^7) \\
&< \log^{(i)}\left(\frac{m}{n\log(n)/\epsilon^2}\right),
\end{aligned}$$

if $\frac{m}{n\log(n)/\epsilon^2} > D$, for some constant $D$. This can be assumed to hold, since if $\frac{m}{n\log(n)/\epsilon^2} \leq D$, then Theorem 3.1 immediately gives the desired result. The total running time becomes of the sum of the $k$ iterations:

$$\begin{aligned}
\sum_{i=1}^{k} O(m_{i-1}) &= O\left(\sum_{i=1}^{k-1} C\cdot\left(\frac{n\log(n)}{\epsilon^2}4^{k-i+2}\cdot 2\log^{(i)}\left(\frac{m}{n\log(n)/\epsilon^2}\right)\right)\right) \\
&= O\left(\frac{n\log(n)}{\epsilon^2}4^k\log\left(\frac{m}{n\log(n)/\epsilon^2}\right)\right).
\end{aligned}$$

We have $\log^*(x) = O(\log\log(x))$, hence we obtain $4^{\log^*(x)}\log(x) = O(\log^2(x)) = O(x)$. Using this with $x = \frac{m}{n\log(n)/\epsilon^2}$ gives us total running time

$$\begin{aligned}
&\sum_{i=1}^{k} O(m_{i-1}) \\
=&O\left(\left(m + \frac{n\log(n)}{\epsilon^2}4^k \log\left(\frac{m}{n\log(n)/\epsilon^2}\right)\right)\right) \\
=&O(m).
\end{aligned}$$

□

Now, we give the remaining proof.

*Proof of Theorem 3.1.* The algorithm is given in Appendix A. First off, if $m \le 4\rho n \log\left(m/(n\log(n)/\epsilon^2)\right) = O(cn\log(n)/\epsilon^2 \log\left(m/(n\log(n)/\epsilon^2)\right))$, the algorithm does nothing and returns the original graph. So for this analysis we can assume $m > 4\rho n \log\left(m/(n\log(n)/\epsilon^2)\right)$. We analyze the time complexity of the algorithm in two steps. The first step consists of computing the probabilities $p_e$ for all $e \in E$. The second one is compressing edges, given these probabilities.

The first step contains $i$ iterations of the while loop (lines 10–17). In each iteration we sample edges from $Y_i \subseteq X_i$ with probability 1/2 to form $X_{i+1}$. This takes time at most $O(|X_i|)$. Next, we compute a maximum spanning forest packing of the graph $G_{i+1} = (V, X_{i+1})$. For an unweighted graph, a MSF packing is the same as an NI packing. We know that we can compute an NI packing a graph with $n$ vertices and $m_0$ edges in $O(m_0)$ time (see [NI92a, NI92b]). So this iteration takes at most $O(|X_{i+1}|)$ time. As noted earlier, we have with high probability that $|X_i| \le \left(\frac{2}{3}\right)^i m$. We conclude w.h.p. that the first step takes total time at most

$$\begin{aligned}
\sum_{i=0}^{\Gamma} (O(|X_i|) + O(|X_{i+1}|)) &= \sum_{i=0}^{\Gamma}\left(\left(\frac{2}{3}\right)^i O(m) + \left(\frac{2}{3}\right)^{i+1} O(m)\right) \\
&\le O(m)\cdot 2\sum_{i=0}^{\infty}\left(\frac{2}{3}\right)^i \\
&= O(m)\cdot 2\cdot 3 \\
&= O(m).
\end{aligned}$$

In the second step, we sample each edge $e$ from the binomial distribution with parameters $n_e$ and $p_e$. As shown in the paper, this can be done with a process that takes $T = O(m)$ time with high probability. Concluding, the algorithm takes $O(m) + O(m) = O(m)$ time in total for unweighted-weighted graphs. □

## References

[FdV21] Sebastian Forster and Tijn de Vos. Faster cut sparsification of weighted graphs. 2021.

[FHHP11] Wai Shing Fung, Ramesh Hariharan, Nicholas J A Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. In *Proc. of the Symposium on Theory of Computing (STOC)*, pages 71–80, New York, NY, USA, 2011.

[FHHP19] Wai-Shing Fung, Ramesh Hariharan, Nicholas J A Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. *SIAM Journal on Computing*, 48(4):1196–1223, 2019.

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

[NI92b] Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse $k$-connected spanning subgraph of a $k$-connected graph. *Algorithmica*, 7(1-6):583–596, 1992.

[Tar75] Robert E Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, April 1975.

## A Unweighted Algorithm

**Algorithm 1:** SPARSIFY$(V, E, w, \epsilon, c)$

```
Input: An undirected, unweighted graph $G = (V, E)$, with parameters $\epsilon \in (0,1)$, $c \geq 1$.
Output: An undirected weighted graph $G_\epsilon = (V, E_\epsilon)$.
1  Set $\rho \leftarrow \frac{(7+c)1352\ln(n)}{0.38\epsilon^2}$.
2  if $|E| \leq 4\rho n \log\big(m/(n\log(n)/\epsilon^2)\big)$ then
3      return $G_\epsilon = G$.
4  end
5  Compute an $\lfloor 2\rho \rfloor$-partial spanning forest packing $T_1, T_2, \ldots, T_{\lfloor 2\rho \rfloor}$ for $G$.
6  Set $i \leftarrow 0$.
7  Set $X_0 \leftarrow E$.
8  Set $F_0 \leftarrow \bigcup_{j=1}^{\lfloor 2\rho \rfloor} T_j$.
9  Set $Y_0 \leftarrow X_0 \setminus F_0$.
10 while $|Y_i| > 2\rho n$ do
11     Sample each edge in $Y_i$ with probability $1/2$ to construct $X_{i+1}$.
12     $i \leftarrow i + 1$.
13     Set $k_i \leftarrow \rho \cdot 2^{i+1}$.
14     Compute an $k_i$-partial spanning forest packing $T_1, T_2, \ldots, T_{k_i}$ for the graph
         $G_i := (V, X_i)$.
15     Set $F_i \leftarrow \bigcup_{j=1}^{k_i} T_j$
16     Set $Y_i \leftarrow X_i \setminus F_i$.
17 end
18 Set $\Gamma \leftarrow i$. // $\Gamma$ is the number of elapsed iteration in the previous while-loop.
19 Add each edge $e \in Y_\Gamma$ to $G_\epsilon$ with weight $2^{\Gamma-1}$.
20 Add each edge $e \in F_0$ to $G_\epsilon$ with weight 1.
21 for $j = 1, \ldots, \Gamma$ do
22     foreach $e \in F_j$ do
23         Set $p_e \leftarrow \min\big(1, \frac{384}{169}\frac{1}{4^j}\big)$.
24         Generate $r_e$ from Binom$(2^j, p_e)$.
25         if $r_e > 0$ then
26             Add $e$ to $G_\epsilon$ with weight $r_e/p_e$.
27         end
28     end
29 end
30 return $G_\epsilon = (V, E_\epsilon)$.
```