Faster Cut Sparsification of Weighted Graphs

Sebastian Forster
Universität Salzburg, Austria

Tijn de Vos
Universität Salzburg, Austria

Abstract

A cut sparsifier is a reweighted subgraph that maintains the weights of the cuts of the original graph up to a multiplicative factor of $(1 \pm \epsilon)$. This paper considers computing cut sparsifiers of weighted graphs of size $O(n \log(n)/\epsilon^2)$. Our algorithm computes such a sparsifier in time $O(m \cdot \min(\alpha(n) \log(m/n), \log(n)))$, both for graphs with polynomially bounded and unbounded integer weights, where $\alpha(\cdot)$ is the functional inverse of Ackermann's function. This improves upon the state of the art by Benczúr and Karger (SICOMP 2015), which takes $O(m \log^2(n))$ time. For unbounded weights, this directly gives the best known result for cut sparsification. Together with preprocessing by an algorithm of Fung et al. (SICOMP 2019), this also gives the best known result for polynomially-weighted graphs. Consequently, this implies the fastest approximate min-cut algorithm, both for graphs with polynomial and unbounded weights. In particular, we show that it is possible to adapt the state of the art algorithm of Fung et al. for unweighted graphs to weighted graphs, by letting the partial maximum spanning forest (MSF) packing take the place of the Nagamochi-Ibaraki (NI) forest packing. MSF packings have previously been used by Abraham at al. (FOCS 2016) in the dynamic setting, and are defined as follows: an M-partial MSF packing of G is a set $\mathcal{F} = \{F_1, \ldots, F_M\}$, where F_i is a maximum spanning forest in $G \setminus \bigcup_{j=1}^{i-1} F_j$. Our method for computing (a sufficient estimation of) the MSF packing is the bottleneck in the running time of our sparsification algorithm.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners

Keywords and phrases Cut Sparsification, Graph Algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.61

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2112.03120

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 947702) and is supported by the Austrian Science Fund (FWF): P 32863-N.

1 Introduction

In many applications, graphs become increasingly large, hence storing and working with such graphs becomes a challenging problem. One strategy to deal with this issue is graph sparsification, where we model the graph by a sparse set of (reweighted) edges that preserve certain properties. Especially because the aim is to work with large input graphs, this process should be efficient with respect to the graph size. Among the different types of graph sparsifiers, there are spanners (preserving distances, see e.g. [26, 2, 4, 11]), resistance sparsifiers (preserving effective resistances, see e.g. [10]), cut sparsifiers (preserving cuts, see e.g. [6, 7, 13]), and spectral sparsifiers (preserving Laplacian quadratic forms, see e.g. [28, 27, 21, 23]). This paper focuses on cut sparsifiers, as first introduced by Benczúr and Karger in [6]. 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, the total weight $w_H(C)$ of the edges of the cut in H is within a multiplicative factor of $1 \pm \epsilon$ of the total weight $w_G(C)$ of the edges of the cut in G.

The main approach to compute cut sparsifiers uses the process of *edge compression*: each edge $e \in E$ is part of the sparsifier with some probability p_e, and if selected obtains weight $w(e)/p(e)$. It is immediate that such a scheme gives a sparsifier in expectation, but it has to be shown that the result is also a sparsifier with high probability. The main line of research has been to select good *connectivity estimators* λ_e for each edge such that sampling with $p_e \sim 1/\lambda_e$ yields a good sparsifier. The simplest such result is by Karger [18], where we sample uniformly with each λ_e equal to the weight of the min cut. Continuing along these lines are parameters as: edge connectivity [13], strong connectivity [6, 7], electrical conductance [27], and Nagamochi-Ibaraki (NI) indices [24, 25, 13]. The challenge within the approach of edge compression is to find a connectivity estimator that results in a sparse graph, but can be computed fast.

For weighted graphs, there are roughly three regimes for sparsification. The first regime consists of cut sparsifiers of size $O(n \log^2(n)/\epsilon^2)$. Fung, Hariharan, Harvey, and Panigrahi [12, 13] show that sparsifiers of this asymptotic size can be computed in linear time for polynomially-weighted graphs. For this they introduce a general framework of cut sparsification with a connectivity estimator, see Section 2.1. For unbounded weights, Hariharan and Panigrahi [16] give an algorithm to compute a sparsifier of size $O(n \log^2(n)/\epsilon^2)$ in time $O(m \log^2(n)/\epsilon^2)$.

The second regime consists of cut sparsifiers of size $O(n \log(n)/\epsilon^2)$. Benczúr and Karger [6, 7] show that these can be computed in time $O(m \log^2(n))$ for polynomially-weighted graphs, and in time $O(m \log^3(n))$ for graphs with unbounded weights. Note that these results can be optimized by preprocessing with the algorithms for the first regime.

A third regime, consists of sparsifiers of size $O(n/\epsilon^2)$. The known constructions in this regime yield *spectral* sparsifiers, which are more general than cut sparsifiers. Spectral sparsification was first introduced by Spielman and Teng in [28]. It considers subgraphs that preserve Laplacian quadratic forms. Lee and Sun [23] give an algorithm for finding $(1 \pm \epsilon)$-spectral sparsifiers of size $O(n/\epsilon^2)$ in time $O(m \cdot \text{poly}(\log(n), 1/\epsilon))$. Analyzing their results, we believe that the poly-logarithmic factor contributes at least a factor of $\log^{10}(n)$. While this is optimal in size, both for spectral sparsifiers [5] and cut sparsifiers [3], it is not in time.

In this paper, we improve on the results in the second regime, both for graphs with polynomially bounded and unbounded weights[1]. For an overview of the previous best running times and our results, see Figure 1. We present our sparsification algorithm in Section 4, for the special treatment of unbounded weights we refer to the full version of the paper. Our algorithm improves on the algorithm of Benczúr and Karger [6, 7] for bounded weights, which has been unchallenged for the last 25 years. It also improves on the algorithm of Hariharan and Panigrahi [16] for unbounded weights, which has been unchallenged for the last 10 years. We obtain the following theorem, where $\alpha(\cdot)$ refers to the functional inverse of Ackermann's function, for a definition see e.g. [29]. For any realistic value x, we have $\alpha(x) \leq 4$.

▶ **Theorem 1.** *There exists an algorithm that, given a weighted graph G and a freely chosen parameter $\epsilon \in (0, 1)$, computes a graph G_ϵ, which is a $(1 \pm \epsilon)$-cut sparsifier for G with high probability. The running time of the algorithm is $O(m \cdot \min(\alpha(n) \log(m/n), \log(n)))$ and the number of edges of G_ϵ is $O\big(n \log(n)/\epsilon^2\big)$.*

Using preprocessing with a result from Fung et al. [13] (see Theorem 23), we obtain the following corollary for polynomially-weighted graphs.

[1] See Section 2.2 for our assumptions on the computational model in case of unbounded weights.

Algorithm	Size	Running time
Unweighted		
Fung et al. [13]	$O\left(n\log(n)/\epsilon^2\right)$	$O\left(m\right)$
Polynomial weights		
Benczúr and Karger [7]	$O\left(n\log(n)/\epsilon^2\right)$	$O\left(m\log^2(n)\right)$
Fung et al. [13]	$O\left(n\log^2(n)/\epsilon^2\right)$	$O\left(m\right)$
[13] + [7]	$O\left(n\log(n)/\epsilon^2\right)$	$O\left(m+n\log^4(n)/\epsilon^2\right)$
This paper	$O\left(n\log(n)/\epsilon^2\right)$	$O(m\log(n))$
This paper	$O\left(n\log(n)/\epsilon^2\right)$	$O(m\alpha(n)\log(m/n))$
[13] + this paper	$O\left(n\log(n)/\epsilon^2\right)$	$O\left(m+n\left(\log^2(n)/\epsilon^2\right)\alpha(n)\log(\log(n)/\epsilon)\right)$
Unbounded weights		
Hariharan and Panigrahi [16]	$O\left(n\log^2(n)/\epsilon^2\right)$	$O\left(m\log^2(n)/\epsilon^2\right)$
Benczúr and Karger [7]	$O\left(n\log(n)/\epsilon^2\right)$	$O\left(m\log^3(n)\right)$
[16] + [7]	$O\left(n\log(n)/\epsilon^2\right)$	$O\left(m\log^2(n)/\epsilon^2+n\log^5(n)/\epsilon^2\right)$
Lee and Sun [23]	$O\left(n/\epsilon^2\right)$	$O\left(m\cdot\mathrm{poly}(\log(n),1/\epsilon)\right)$
This paper	$O\left(n\log(n)/\epsilon^2\right)$	$O(m\log(n))$
This paper	$O\left(n\log(n)/\epsilon^2\right)$	$O(m\alpha(n)\log(m/n))$

Figure 1 An overview of the state of the art algorithms for computing cut sparsifiers for undirected graphs with integer weights. Algorithm $A+B$ indicates that algorithm B is preprocessed with algorithm A.

▶ **Corollary 2.** *There exists an algorithm that, given a polynomially-weighted graph G and a freely chosen parameter $\epsilon\in(0,1)$, computes a graph G_ϵ, which is a $(1\pm\epsilon)$-cut sparsifier for G with high probability. The running time of the algorithm is $O(m+n\left(\log^2(n)/\epsilon^2\right)\alpha(n)\log(\log(n)/\epsilon))$ and the number of edges of G_ϵ is $O(n\log(n)/\epsilon^2)$.*

Following Benczúr and Karger [7], the computation of cut sparsifiers of graphs with fractional or even real weights can be reduced to integer weights. For the reduction see Appendix C. Thus our algorithm also gives a speedup for such graphs. Since the integer case is the essential one, we follow prior works and only formulate our results for this particular case.

As a direct application of the cut sparsifier, we can use Theorem 1 and Corollary 2 to replace m by $n\log(n)/\epsilon^2$ in the time complexity of algorithms solving cut problems, at the cost of a $(1\pm\epsilon)$-approximation. We detail the effects for the minimum cut problem. Recently, Gawrychowski, Mozes, and Weiman [14] showed that one can compute the minimum cut of a weighted graph in $O(m\log^2(n))$ time. Using the existing sparsification techniques [7, 13] for preprocessing, the state of the art for $(1+\epsilon)$-approximate min-cut is $O(m+n\log^4(n)/\epsilon^2)$. When we use our new sparsification results, we obtain faster $(1+\epsilon)$-approximate min-cut algorithms when $m=\Omega(n\log(n)/\epsilon^2)$.

▶ **Corollary 3.** *There exists an algorithm that, given a polynomially-weighted graph G and a freely chosen parameter $\epsilon\in(0,1)$, with high probability computes an $(1+\epsilon)$-approximation of the minimum cut in time $O(m+n\log^3(n)/\epsilon^2)$.*

There exists an algorithm that, given a weighted graph G and a freely chosen parameter $\epsilon\in(0,1)$, with high probability computes an $(1+\epsilon)$-approximation of the minimum cut in time $O(m\cdot\min(\alpha(n)\log(m/n),\log(n))+n\log^3(n)/\epsilon^2)$.

For unweighted graphs, even faster minimum cut algorithms exist: Ghaffari, Nowicki, and Thorup [15] show that we can find the minimum cut in $O(\min\{m+n\log^3(n), m\log(n)\})$ time. Combining this with the linear time cut sparsifier of Fung et al. [13], we get $(1+\epsilon)$-approximate minimum cut in unweighted graphs in $O(m + n\log(n)\min\{1/\epsilon + \log^2(n), \log(n)/\epsilon\})$ time.

The remainder of this article is organized as follows. The rest of the introduction consists of a technical overview of our algorithms. Section 2 contains a review of the general sparsification framework from Fung et al. [13] tailored to our needs, and can be skipped by readers that are already familiar with this work. We present our algorithm to compute the MSF indices in Section 3. This is used as a black box in our algorithm, which is presented and analyzed in Section 4.

Technical Overview

The high-level set-up of our sparsification algorithm is similar to the algorithm for unweighted graphs of Fung et al. [13]. Our main contribution consists of showing how to generalize this technique to weighted graphs, by using maximum spanning forest (MSF) indices instead of Nagamochi-Ibaraki (NI) indices. On a less significant note, we prove that by a tightening of the analysis one can show that the size and time bounds hold with high probability, and not only in expectation.

NI indices are defined by means of an NI forest packing: view graphs with integer weights as unweighted multigraphs, and repeatedly compute a spanning forest. The NI index is the (last) forest in which an edge appears (for details see Definition 22). The MSF index is also defined by a forest packing, but in this case the MSF packing: we say $\mathcal{F} = \{F_1, \ldots, F_M\}$ is an *M-partial maximum spanning forest packing* of G if for all $i = 1, \ldots, M$, F_i is a maximum spanning forest in $G \setminus \bigcup_{j=1}^{i-1} F_j$. Now, we say that an edge e has MSF index i (w.r.t. to some (partial) MSF packing $\mathcal{F}$) if e appears in the i-th forest F_i of the (partial) MSF packing $\mathcal{F}$. The MSF index has been used previously in the context of dynamic graph sparsifiers (see Abraham et al. [1]). However, there it was only used because it rendered a faster running time, but using NI indices in the corresponding static construction would have been possible as well. In this paper, we use distinctive properties of the MSF index, and the NI index would not suffice. We show that using the MSF index, we can generalize the sparsification algorithm for unweighted graphs to an algorithm for weighted graphs, thereby demonstrating that the MSF index is a natural analogue for the NI index in the weighted setting. We provide an algorithm to compute an M-partial MSF packing in time $O(m \cdot \min(\alpha(n)\log(M), \log(n)))$ for polynomially-weighted graphs. We show that for unbounded weights we can compute a sufficient estimation, also in time $O(m \cdot \min(\alpha(n)\log(M), \log(n)))$.

An important distinction between the unweighted algorithm of Fung et al. and our weighted algorithm, is that the use of contractions to keep running times low throughout the algorithm is no longer possible: edges of different weights have to be treated differently, hence cannot be contracted. By using multiple iterations with an exponentially decreasing precision parameter we can overcome this problem.

In the case of a polynomially-weighted input graph, the algorithm consists of two main phases. In the first phase, we compute sets $F_0, F_1, \ldots, F_\Gamma \subseteq E$, where edges satisfy some lower bound on the weight of any cut separating their endpoints. In the second phase, we sample edges from each set F_i with a corresponding probability.

We set a parameter $\rho = \Theta\left(\frac{\ln(n)}{\epsilon^2}\right)$ and start by computing a 2ρ-partial maximum spanning forest packing for G. We define F_0 to be the union of these 2ρ forests. We add the edges of F_0 to G_ϵ, which will become our sparsifier. We sample each of the remaining edges $E \setminus F_0$ with probability $1/2$ to construct X_1. To counterbalance for the sampling, we will boost

the weight of each sampled edge with a factor 2. Now we continue along these lines, but in each iteration we let F_i consist of an exponentially growing number of spanning forests: F_i is defined as the union of the forests in a $(2^{i+1} \cdot \rho)$-partial MSF packing packing of X_i. Then, X_{i+1} is sampled from the remaining edges $X_i \setminus F_i$, where again each edge is included with probability $1/2$. We continue this process until there are sufficiently few edges left in X_{i+1}. We add these remaining edges to G_ϵ.

The second phase of the algorithm is to sample edges from the sets F_i and add these sampled edges to G_ϵ. Hereto, note that an edge e of F_i (for $i \geq 1$) was not part of F_{i-1}, meaning it was not part of any spanning forest in a $(2^i \cdot \rho)$-partial MSF packing of X_{i-1}. This implies that for an edge $e \in F_i$ the weight of any cut C in X_{i-1} containing e is at least $2^i \cdot \rho \cdot w(e)$. Now we use the general framework for cut sparsification of Fung et al. [13], which boils down to the fact that this guarantee on the weights of cuts implies that we can sample edges from F_i with probability proportional to $1/(2^i w(e))$. We show that this results in a sufficiently sparse graph.

Intuitively, it might seem redundant to sample edges from $X_i \setminus F_i$ to form X_{i+1}. This is indeed not necessary to guarantee that the resulting graph is a sparsifier. However, it ensures that the number of iterations is limited, which leads to better bounds on the size of the sparsifier and the running time. Since we sample edges with probability $1/2$ in each phase, we need to repeat the sampling $O(\log(m/(m_0))$ times to get the size of X_i down to $O(m_0)$. As this number of steps depends on the initial number of edges m, we get better bounds for size and running time if m is already small. We will exploit this by preprocessing the graph with an algorithm from Fung et al. [13] that gives a cut sparsifier of size $O(n \log^2(n)/\epsilon^2)$ in linear time. Moreover, we can show that repeatedly calling our algorithm has no worse asymptotic time bound than calling it once, since the input graph becomes sparser very quickly. By doing so, we obtain a sparsifier of size $O(n \log(n)/\epsilon^2)$.

Since we only use that the MSF index gives a guaranteed lower bound on the connectivity of an edge, one might wonder why the NI index does not work here. After all, the NI indices of a graph can be computed in linear time, which would result in a significant speed-up. However, when computing the NI index, the weight of an edge influences the number of forests necessary, while computing the MSF index only requires the comparison of weights. Moreover, the number of trees in a MSF packing is always bounded by n. We can use this to bound the number of edges in the created sparsifier. The same technique with NI indices would make the size of the sparsifier depend on the maximum weight in the original graph.

To show that the algorithm outputs a cut sparsifier, it needs to be proven that both the sampling in the first and the second phase preserve cuts. We follow the lines of the analysis of Fung et al. [13], which makes use of cut projections and Chernoff bounds. We show that by partitioning the edge sets according to their weight this method extends to weighted graphs.

One part of the algorithm has remained unaddressed: the computation of the maximum spanning forests. The approach we use here is related to Kruskal's algorithm for computing minimum spanning trees [22]. We start by sketching the M-partial MSF packing algorithm for polynomial weights. We sort the edges according to their weights using radix sort in $O(m)$ time. We create M empty forests on n vertices. Starting with the heaviest edge, we add each edge e to the first forest in which it does not create a cycle. We can find this forest using a binary search in $\log(M)$ steps. By using a disjoint-forest representation for the union-find data structure necessary to carry out these steps, we achieve a total time of $O(m\alpha(n) \log(M))$.

When working with unbounded weights, the bottleneck is the initial sorting of the edges. Radix sort does not guarantee to be efficient for unbounded weights. Instead we could use a comparison-based algorithm, such as merge sort, which takes time $O(m \log(n))$. By employing

a different data structure than before, we can guarantee total running time $O(m \log(n))$. However, we do not need the exact MSF indices for our sampling procedure, an estimate suffices. We can adapt a "windowing" technique from Benczúr and Karger [7] to split the graph into subgraphs, where we can rescale the weights to polynomial weights and apply our previously mentioned algorithm. We then achieve a total running time of $O(m\alpha(n) \log(M))$, as before. For more details on this, we refer to the full version. So in total we have running time $O(m \cdot \min(\alpha(n) \log(m/n), \log(n)))$.

2 Notation and Review

Throughout this paper, we consider $G = (V, E)$ to be an undirected, integer weighted graph on $|V| = n$ vertices with $|E| = m$ edges. We define a set of edges $C \subseteq E$ to be a *cut* if there exists a partition of the vertices V in two non-empty subsets 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 4.** *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.*

The following theorem from Fung et al. [13] bounds the number of distinct k-projections of a graph, it is a generalization of a preceding theorem by Karger, see [17, 20]. This result can be useful when showing that cuts are preserved by a sampling scheme. This is due to the fact that while there may be exponentially many different cuts, this theorem shows that there are only polynomially many cut projections. Hence if one can reduce a claim for cuts to their k-projections, a high probability bound can be obtained through the application of a Chernoff bound.

▶ **Theorem 5.** *For any $k \geq \lambda$ and any $\eta \geq 1$, the number of distinct k-projections in cuts of weight at most ηk in a graph G is at most $n^{2\eta}$, where λ is the weight of a minimum cut in G.*

Throughout this paper, we say a statement holds *with high probability* (w.h.p.) if it holds with probability at least $1 - n^c$, for some constant c. This constant can be modified by adjusting the constants hidden in asymptotic notation.

2.1 A General Framework for Cut Sparsification

We review the general framework for cut sparsification as presented in [13]. This section does not contain new results, and can be skipped by readers that are only interested in our contribution.

The framework shows that edges can be sampled using different notions of connectivity estimators. Although this scheme provides one proof for the validity of multiple parameters, it might be worth noting that an analysis tailored to the used connectivity estimator might provide a better result. For example, when the framework is applied with "edge strengths", it produces a sparsifier of size $O(n \log^2(n)/\epsilon^2)$, a $\log(n)$ factor denser than the edge strength-based sparsifier of Benczúr and Karger [7].

Let $G = (V, E)$ be a graph with integer weights, and let $\epsilon \in (0, 1)$, $c \geq 1$ be parameters. Given a parameter γ (possibly depending on n) and an integer-valued parameter λ_e for each $e \in E$. We obtain G_ϵ 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_ϵ with weight r_e/p_e.

In the following we describe a sufficient condition on the parameters γ and λ_e such that G_ϵ 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 λ_e:

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

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 $\Pi = \{\pi_1, \ldots, \pi_\Lambda\} \subseteq \mathbb{R}^\Lambda$, we define

- Π-*connectivity*: each edge $e \in R_i$ is π_i-heavy in G_i;
- γ-*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 6** (See [13, Theorem 1.14]). *Fix the parameters γ and λ_e for each edge e. If there exists $\mathcal{G}$ satisfying Π-connectivity and γ-overlap for some Π, then G_ϵ is a $(1 \pm \epsilon)$-cut sparsifier for G, with probability at least $1 - 4/n^c$, where G_ϵ is obtained by edge compression using parameters γ and λ_e's.*

2.2 The Computational Model

If we have an input graph $G = (V, E)$ with weights $w\colon E \to \{1, \ldots, W\}$, we assume our computational model has word size $\Theta(\log(W) + \log(n))$. Note that for polynomial weights, this comes down to a word size of $\Theta(\log(n))$. Moreover, we assume that basic operations on such words have uniform cost, i.e., they can be performed in constant time. In particular, these basic operations are addition, multiplication, inversion, logarithm, and sampling a random bit string of word size precision. Such assumptions are in line with previous work [7, 13], where they are made implicitly.

3 A Maximum Spanning Forest Packing

An important primitive in our algorithm is the use of the maximum spanning forest (MSF) index. The concept is similar to the Nagamochi-Ibaraki index, the important difference is that an edge e with weight $w(e)$ appears in $w(e)$ different NI forests. This means that the

number of NI forests depends on the numerical values of the edge weights, and thus can grow far beyond $O(n)$. On the other hand, the number of maximum spanning forests in a MSF packing is bounded by the maximum degree in the graph, hence also by n. While this already has noteworthy implications for polynomially-weighted graphs, it is even more significant for superpolynomially-weighted graphs. We believe that this property might make them suitable for applications other than presented here.

▶ **Definition 7.** *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 bests in our applications. Also note that a (partial) MSF packing is fully determined by the MSF indices.

The following theorem states that computing the MSF indices up to M takes $O(m\alpha(n)\log(M))$ time for polynomially-weighted graphs.

▶ **Theorem 8.** *Let $G = (V, E)$ be a polynomially weighted graph, where we allow parallel edges but no self-loops, and we suppose $m \leq n^2$. Then, for any $M > 0$, there exists an algorithm that computes an M-partial MSF packing in $O(m \cdot \min(\alpha(n)\log(M), \log(n)))$ time.*

The outline of the algorithm is as follows, for a complete proof see the full version.

1. Sort the edges by weight in descending order using radix sort in base n.[2]
2. Create empty forests $F_1, \dots, F_M$.
3. Iterate over the edges in descending order and for each edge $e = (u, v)$ do the following:
   a. Find the smallest index i such that u and v are not connected in F_i.
   b. Store i as the MSF index f_e of e. If u and v are connected in every F_i, store $f_e > M$.
   c. Add e to F_i.

We need at most M trees, since we only compute an M-partial MSF packing. By using radix sort, the initial sorting takes time $O(m)$ time (for a time bound of radix sort, see e.g. [9]). In the full version, we show that the remainder of the algorithm can be executed in $O(m\alpha(n)\log(M))$ time or $O(m\log(n))$, depending on the data-structure used. There we also consider an algorithm for *sparse* graphs with unbounded weights.

4 Cut Sparsification for Weighted Graphs

In this section, we present our algorithm for computing a $(1 \pm \epsilon)$-cut sparsifier G_ϵ for a weighted graph G. This makes use of the framework as presented in Section 2.1 and the maximum spanning forest packing as treated in Section 3. This section works towards proving the following theorem for polynomially-weighted graphs. In the full version, we generalize the techniques of this section to graphs with unbounded weights.

▶ **Theorem 9.** *There exists an algorithm that, given a weighted graph $G = (V, E)$, and freely chosen parameter $\epsilon > 0$, computes a graph G_ϵ, which is a $(1 \pm \epsilon)$-cut sparsifier for G with high probability. The algorithm runs in time $O(m \cdot \min(\alpha(n)\log(m/n), \log(n)))$ and the number of edges of G_ϵ is $O\big(n\left(\log(n)/\epsilon^2\right)\log\left(m/(n\log(n)/\epsilon^2)\right)\big)$.*

[2] Note that conversion to base n takes time $O(\log_n(w(e))) \leq O(\log_n(n^c)) = O(c)$ for each edge if the weights are bounded by n^c, so total time $O(mc)$.

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 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.

To achieve a better bound on the size of the sparsifier, we repeatedly apply this theorem to the input graph, with an exponentially decreasing precision parameter. The proof of this can be found in the full version.

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

4.1 The Algorithm

To sparsify the graph, two methods of sampling are used. One of which is the framework presented in Section 2.1. However, instead of applying the framework to the graph directly, there is another sampling process that precedes it.

To simplify equations, let us set $\rho := \frac{(7+c)1352 \ln(n)}{0.38\epsilon^2}$. If $|E| \leq 4\rho n \log\big(m/(n \log(n)/\epsilon^2)\big)$, we do nothing. That is, we return $G_\epsilon = G$. If not, we start by an initialization step and continue with an iterative process, which ends when the remaining graph becomes sufficiently small.

In the initialization step, we define $X_0 := E$. We compute an $\lfloor 2\rho \rfloor$-partial maximum spanning forest packing $T_1, \ldots, T_{\lfloor 2\rho \rfloor}$ and we define $F_0 := \bigcup_{j=1}^{\lfloor 2\rho \rfloor} T_j$. The remaining edges $Y_0 := X_0 \setminus F_0$ move on to the next phase.

In iteration i, we create X_{i+1} from Y_i by sampling each edge with probability $1/2$. Next, we compute $k_i := \rho \cdot 2^{i+1}$ maximum spanning forests $T_1, \ldots, T_{k_i}$. We define $F_i := \bigcup_{j=1}^{k_i} T_j$, and $Y_i := X_i \setminus F_i$.

We continue until Y_i has at most $2\rho n$ edges, and set Γ to be the number of iterations. We retain all edges in F_0. In other words: add each edge $e \in F_0$ to G_ϵ with weight $w(e)$. The edges of Y_Γ are also retained, but they need to be scaled to counterbalance the $\Gamma - 1$ sampling steps: add each edge $e \in Y_\Gamma$ to G_ϵ with weight $2^{\Gamma-1} w(e)$.

Any other edge $e \in F_i$ is at least $k_i w(e)$-heavy in X_{i-1}, as $e \notin F_{i-1}$. We exploit this heavyness to sample from these edges using the framework. For each $e \in F_i$ we:

- Define $n_e := 2^i w(e)$ and $p_e := \min\left(1, \frac{384}{169} \frac{1}{4^i w(e)}\right)$;
- Generate r_e from the binomial distribution with parameters n_e and p_e;
- If r_e is positive, add e to G_ϵ with weight r_e/p_e.

The factor 2^i in calling upon the binomial distribution can be seen as boosting the weight of the edge by a factor 2^i, which is needed to counterbalance the i sampling steps in creating F_i.

Pseudocode of this algorithm can be found in Algorithm 1. Up to the computation method of the MSF packing, the presented algorithm is the same for polynomially and superpolynomially-weighted graphs. For the unbounded case, we use the MSF index estimator as presented in the full version. There we also detail how this influences the correctness of the algorithm, and the bounds on size and running time.

Algorithm 1 SPARSIFY(V, E, w, ϵ, c).

Input: An undirected graph $G = (V, E)$, with integer weights $w\colon E \to \mathbb{N}^+$, and parameters $\epsilon \in (0, 1)$, $c \geq 1$.
Output: An undirected weighted graph $G_\epsilon = (V, E_\epsilon)$.

Set $\rho \leftarrow \frac{(7+c)1352\ln(n)}{0.38\epsilon^2}$.
if $|E| \leq 4\rho n \log\left(m/(n\log(n)/\epsilon^2)\right)$ **then**
  return $G_\epsilon = G$.
end
Compute an $\lfloor 2\rho \rfloor$-partial maximum spanning forest packing $T_1, T_2, \ldots, T_{\lfloor 2\rho \rfloor}$ for G.
Set $i \leftarrow 0$.
Set $X_0 \leftarrow E$.
Set $F_0 \leftarrow \bigcup_{j=1}^{\lfloor 2\rho \rfloor} T_j$.
Set $Y_0 \leftarrow X_0 \setminus F_0$.
while $|Y_i| > 2\rho n$ **do**
  Sample each edge in Y_i with probability $1/2$ to construct X_{i+1}.
  $i \leftarrow i + 1$.
  Set $k_i \leftarrow \rho \cdot 2^{i+1}$.
  Compute an k_i-partial maximum spanning forest packing $T_1, T_2, \ldots, T_{k_i}$ for the graph $G_i := (V, X_i)$.
  Set $F_i \leftarrow \bigcup_{j=1}^{k_i} T_j$
  Set $Y_i \leftarrow X_i \setminus F_i$.
end
Set $\Gamma \leftarrow i$. // Γ is the number of elapsed iteration in the previous while-loop.
Add each edge $e \in Y_\Gamma$ to G_ϵ with weight $2^{\Gamma-1}w(e)$.
Add each edge $e \in F_0$ to G_ϵ with weight $w(e)$.
for $j = 1, \ldots, \Gamma$ **do**
  foreach $e \in F_j$ **do**
    Set $p_e \leftarrow \min\left(1, \frac{384}{169}\frac{1}{4^j w(e)}\right)$.
    Generate r_e from Binom$(2^j w(e), p_e)$.
    if $r_e > 0$ **then**
      Add e to G_ϵ with weight r_e/p_e.
    end
  end
end
return $G_\epsilon = (V, E_\epsilon)$.

4.2 Correctness

We will prove that G_ϵ constructed in SPARSIFY$(V, E, w, \epsilon$,c) is a $(1 \pm \epsilon)$-cut sparsifier for G with probability at least $1 - 8/n^c$. Following the proof structure of [13], we first define

$$S := \left(\bigcup_{i=0}^{\Gamma} 2^i F_i\right) \cup 2^\Gamma Y_\Gamma,$$

where Γ is the maximum number such that $F_i \neq \emptyset$. We define $G_S := (V, S)$. And we prove the following two lemmas, that together yield the desired result.

▶ **Lemma 10.** *G_S is a $(1 \pm \epsilon/3)$-cut sparsifier for G with probability at least $1 - 4/n^c$.*

▶ **Lemma 11.** *G_ϵ is a $(1 \pm \epsilon/3)$-cut sparsifier for G_S with probability at least $1 - 4/n^c$.*

Let us start by proving Lemma 10. The omitted proofs of the lemmas and corollaries used can all be found in the full version. In creating the sets F_i, we repeatedly makes use of the MSF indices. The MSF index of an edge immediately ensures a certain connectivity of that edge. The following lemma makes this precise.

▶ **Lemma 12.** *Let $i \geq 0$ and $e \in Y_i$ be an edge, and set $k_i := \rho \cdot 2^{i+1}$. Then e is $w(e)k_i$-heavy in $G'_{i,e} = (V, X'_{i,e})$, where $X'_{i,e} := \{e' \in X_i : w(e') \geq w(e)\}$. Consequently, e is also $w(e)k_i$-heavy in $G_i = (V, X_i)$.*

Next, we show in a general setting that certain ways of sampling preserve cuts. The following lemma is a generalization of Lemma 5.5 in [13].

▶ **Lemma 13.** *Let $R \subseteq Q$ be subsets of weighted edges on some set of vertices V, satisfying $0 < w(e) \leq 1$ for all $e \in Q$. Moreover, assume that each edge in R is π-heavy in (V, Q). Suppose that each edge $e \in R$ is sampled with probability $p \in (0, 1]$, and if selected, given a weight of $w(e)/p$ to form a set of edges $\widehat{R}$. We denote, for every cut C:*

$$r^{(C)} := \sum_{e \in R \cap C} w(e), \qquad q^{(C)} := \sum_{e \in Q \cap C} w(e), \qquad \widehat{r}^{(C)} := \sum_{e \in = \widehat{R} \cap C} w(e)/p.$$

Let $\zeta \in \mathbb{N}_{\geq 5}$, and $\delta \in (0, 1]$ such that $\delta^2 p\pi \geq \frac{\zeta \ln(n)}{0.38}$, then

$$\left| r^{(C)} - \widehat{r}^{(C)} \right| \leq \delta q^{(C)}$$

for all cuts C, with probability at least $1 - 4/n^{\zeta - 4}$.

We want to apply this lemma to our sampling procedure. We do this by considering different weight classes separately. We define $X_{i,k} := \{e \in X_i : 2^k \leq w(e) \leq 2^{k+1} - 1\}$, and $x^{(C)}_{i,k} = \sum_{e \in X_{i,k} \cap C} w(e)$. We define $Y_{i,k}$ and $y^{(C)}_{i,k}$ analogously. Some rescaling is necessary to ensure that all weights lie in $(0, 1]$, as Lemma 13 requires. For $A \subseteq E$ and $\beta > 0$, we write βA to indicate we multiply the weight of the edges by a factor of β.

▶ **Lemma 14.** *With probability at least $1 - 4/n^{4+c}$, for every cut C in G_i,*

$$\left| 2^{-k} x^{(C)}_{i+1,k} - 2^{-k-1} y^{(C)}_{i,k} \right| \leq \frac{\epsilon/13}{2^{i/2+1}} \sum_{k'=k}^{\infty} 2^{-k'-1} x^{(C)}_{i,k'}.$$

Now we look at the general case, for which we sum all weight classes. Hereto, we define $x^{(C)}_i = \sum_{e \in X_i \cap C} w(e)$, $x^{(C)}_{i+1} = \sum_{e \in X_{i+1} \cap C} w(e)$, and $y^{(C)}_i = \sum_{e \in Y_i \cap C} w(e)$.

▶ **Corollary 15.** *With probability at least $1 - 4/n^{1+c}$, for every cut C in G_i,*

$$\left| 2x^{(C)}_{i+1} - y^{(C)}_i \right| \leq \frac{\epsilon/13}{2^{i/2}} \cdot x^{(C)}_i.$$

We will repeatedly apply this lemma. To show that the accumulated error does not grow beyond $\epsilon/3$, we use the following fact. For a proof we refer to [13].

▶ **Lemma 16.** *Let $x \in (0, 1]$ be a parameter. Then for any $k \geq 0$,*

$$\prod_{i=0}^{k} \left(1 + \frac{x/13}{2^{i/2}}\right) \leq 1 + x/3,$$

$$\prod_{i=0}^{k} \left(1 - \frac{x/13}{2^{i/2}}\right) \geq 1 - x/3.$$

As a final step towards proving Lemma 10, we prove a lemma that focusses on the sparsification occurring in the last $\Gamma - j + 1$ iterative steps of our algorithm.

▶ **Lemma 17.** *Let*

$$S_j = \left(\bigcup_{i=j}^{\Gamma} 2^{i-j} F_i \right) \cup 2^{\Gamma - j} Y_\Gamma$$

for any $j \geq 0$. Then, S_j is a $\left(1 \pm (\epsilon/3)2^{-j/2}\right)$-cut sparsifier for $G_j = (V, X_j)$, with probability at least $1 - 4/n^c$.

Note that setting $j = 0$ gives us Lemma 10.

To prove Lemma 11, we will invoke the framework from [13], as given in Section 2.1. More specifically, we will apply Theorem 6. We set the parameter $\gamma := 64/3$, and for each $e \in F_i$ we set $\lambda_e := \rho \cdot 4^i w(e)$. This is in line with our choice for p_e:

$$\min\left(1, \frac{16(c+7)\gamma \ln(n)}{0.38\lambda_e \epsilon^2}\right) = \min\left(1, \frac{16(c+7)\gamma \ln(n)}{0.38\rho \cdot 4^i w(e)_e \epsilon^2}\right) = \min\left(1, \frac{384}{169}\frac{1}{4^i w(e)}\right) = p_e.$$

We have to provide a set of subgraphs $\mathcal{G}$ and a set of parameters Π such that Π-connectivity and γ-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:

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

We will view these edges in the subgraph:

$$E_{j,k} := \bigcup_{j'=j-1}^{\Gamma} \bigcup_{k'=k}^{\infty} \rho \cdot 4^{\Gamma - j' + 1} 2^{\Lambda - k' + j'} R_{j',k'}.$$

▶ **Lemma 18.** *Each edge $e \in R_{j,k}$ is $\pi := \rho \cdot 4^\Gamma 2^\Lambda$-heavy in $(V, E_{j,k})$.*

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

▶ **Corollary 19.** *Each edge in $e \in R_i$ is $\rho \cdot 4^\Gamma 2^\Lambda$-heavy in $G_i = (V, E_i)$, with $E_i := \bigcup_{j=1}^{\min(\lfloor i/2 \rfloor, \Gamma)} E_{j,i-2j}$.*

It remains to show that γ-overlap is satisfied.

▶ **Lemma 20.** *For any cut C,*

$$\sum_{i=0}^{\Lambda} \frac{e_i^{(C)} 2^{i-1}}{\rho \cdot 4^\Gamma 2^\lambda} \leq 64/3 \cdot e^{(C)},$$

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

Together Corollary 19 and Lemma 20 show that the conditions of Theorem 6 are met with the given parameters. This proves Lemma 11, and then Theorem 9 follows.

4.3 Size of the Sparsifier

The sparsifier G_ϵ consists of F_0, Y_Γ, and F', where $F' = \cup_{i=1}^{\Gamma} F'_i$, with F'_i the sampled edges of F_i. First of all, note that $|F_0| = O(cn\ln(n)/\epsilon^2)$ and $|Y_\Gamma| = O(cn\ln(n)/\epsilon^2)$. Now take $e \in F_i$. This edge results to an edge in G_ϵ if the sample from the binomial distribution with parameters $n_e = 2^i w(e)$ and $p_e = \min\left(1, \frac{384}{169}\frac{1}{4^i w(e)}\right)$ is positive. The probability that this happens is

$$\begin{aligned}
\mathbb{P}[\text{Binom}(n_e, p_e) > 0] &= \sum_{k=1}^{n_e} \mathbb{P}[\text{Binom}(n_e, p_e) = k] &&\leq \sum_{k=1}^{n_e} k\,\mathbb{P}[\text{Binom}(n_e, p_e) = k] \\
&= \sum_{k=0}^{n_e} k\,\mathbb{P}[\text{Binom}(n_e, p_e) = k] &&= \mathbb{E}[\text{Binom}(n_e, p_e)] \\
&= n_e p_e &&\leq \frac{384}{169} 2^{-i}.
\end{aligned}$$

Note that this probability is equal for all $e \in F_i$. Since F_i is the union of $k_i = \rho \cdot 2^{i+1}$ spanning forests, we know that $|F_i| \leq \rho 2^{i+1} n$. Hence the expected size of F'_i, the sampled edges in F_i, equals

$$\begin{aligned}
\mathbb{E}[|F'_i|] &= \sum_{e \in F_i} \mathbb{P}[\text{Binom}(n_e, p_e) > 0] \leq \sum_{e \in F_i} \frac{384}{169} 2^{-i} = |F_i| \frac{384}{169} 2^{-i} \leq \rho 2^{i+1} n \frac{384}{169} 2^{-i} \\
&= \rho \frac{768}{169} n.
\end{aligned}$$

We have that the total number of sampled edges equals

$$\mathbb{E}[|F'|] = \sum_{i=1}^{\Gamma} \mathbb{E}[|F'_i|] \leq \Gamma \rho \frac{768}{169} n,$$

so it remains to bound Γ, i.e., the number of F_i's. Hereto, note that the while loop of lines 10–17 ends if $|Y_i| \leq 2\rho n$. We bound the number of edges in Y_i by bounding the number of edges of X_i, of which Y_i is a subset. Each edge in $Y_{i-1} \subseteq X_{i-1}$ is sampled with probability $1/2$ to form X_i. So $\mathbb{E}[|X_i|] \leq |X_{i-1}|/2$. Now by a Chernoff bound (see Theorem 26) we obtain:

$$\mathbb{P}\left[|X_i| > \frac{2}{3}|X_{i-1}|\right] \leq \exp\left(-\frac{0.38}{36}|X_{i-1}|\right) > \exp\left(-\frac{cn\ln(n)}{36}\right) = n^{-cn/36},$$

since $|X_{i-1}| \geq |Y_{i-1}| \geq 2\rho n = 2 \cdot \frac{(7+c)1352\ln(n)}{0.38\epsilon^2} n \geq \frac{cn\ln(n)}{0.38}$. We have at most n^2 sets X_i, so we can conclude that with high probability $|X_i| \leq \frac{2}{3}|X_{i-1}|$ in each step, and by induction $|Y_i| < |X_i| \leq \left(\frac{2}{3}\right)^i m$. We see that

$$m\left(\frac{2}{3}\right)^{\Gamma} \leq 2\rho n = \frac{21632}{0.38\epsilon^2} cn\ln(n),$$

which is equivalent to

$$\Gamma \geq \log\left(\frac{m}{\frac{21632}{0.38\epsilon^2} cn\ln(n)}\right) / \log(3/2).$$

So, we can conclude $\Gamma = O\left(\log\left(\frac{m}{cn\log(n)/\epsilon^2}\right)\right)$. This gives that the total number of sampled edges is, in expectation,

$$\mathbb{E}[|F'|] \leq \Gamma \rho \frac{768}{169} n = O(cn\log(n)\log\left(m/(cn\log(n)/\epsilon^2)\right)/\epsilon^2).$$

This compression process can also be seen as the sum of m independent random variables that take values in $\{1, 0\}$.[3] We have just calculated that the expected value μ is at most $Bcn\ln(n)\log\left(m/(cn\log(n)/\epsilon^2)\right)/\epsilon^2$, for some $B > 0$. Using this, we apply a Chernoff bound (Theorem 26) to get an upper limit for the number of sampled edges:

$$\begin{aligned}
&\mathbb{P}\left[|F'| > 2Bcn\ln n\log\left(m/(cn\log(n)/\epsilon^2)\right)/\epsilon^2\right] \\
&\qquad \leq \exp\left(-0.38Bcn\ln(n)\log\left(m/(cn\log(n)/\epsilon^2)\right)/\epsilon^2\right) \\
&\qquad = n^{-0.38cnB\log\left(m/(cn\log(n)/\epsilon^2)\right)/\epsilon^2}.
\end{aligned}$$

We conclude that, with high probability, the number of sampled edges is

$$O(2Bcn\ln(n)\log\left(m/(cn\log(n)/\epsilon^2)\right)/\epsilon^2) = O(cn\log(n)\log\left(m/(cn\log(n)/\epsilon^2)\right)/\epsilon^2).$$

And finally, we conclude that with high probability the number of edges of G_ϵ is bounded by $|E(G_\epsilon)| = |F_0| + |Y_\Gamma| + |F'| = O(cn\log(n)\log\left(m/(cn\log(n)/\epsilon^2)\right)/\epsilon^2)$.

4.4 Time Complexity

First off, if $m \leq 4\rho n\log\left(m/(n\log(n)/\epsilon^2)\right) = O(cn\log(n)/\epsilon^2\log\left(m/(n\log(n)/\epsilon^2)\right))$, 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 phases. The first phase consists of computing the probabilities p_e for all $e \in E$. The second one is compressing edges, given these probabilities.

The first phase 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 $G_{i+1} = (V, X_{i+1})$. We know that we can compute a M-partial maximum spanning forest packing of a polynomially-weighted graph with n vertices and m_0 edges in $O(m_0 \cdot \min(\alpha(n)\log(M), \log(n)))$ time (see Theorem 8). So this iteration takes at most $O(|X_{i+1}| \cdot (\min(\alpha(n)\log(k_{i+1}), \log(n))))$ time. As noted earlier, we have with high probability that $|X_i| \leq \left(\frac{2}{3}\right)^i m$. If $m\alpha(n)\log(m/n) \leq m\log(n)$, we conclude w.h.p. that the first phase takes total time at most

$$\begin{aligned}
\sum_{i=0}^{\Gamma} O(|X_i|) + O(|X_{i+1}|\alpha(n)\log(k_{i+1})) &= \sum_{i=0}^{\Gamma}\left(\frac{2}{3}\right)^i O(m) + \left(\frac{2}{3}\right)^{i+1} O(m\alpha(n)\log(\rho 2^{i+2})) \\
&\leq 3O(m) + 3O(m\alpha(n)\log(\rho 2^{\Gamma})) \\
&= O(m\alpha(n)\log(m/n)).
\end{aligned}$$

And if $m\log(n) < m\alpha(n)\log(m/n)$, we have that w.h.p. the first phase takes total time at most

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

In the second phase, we sample each edge e from the binomial distribution with parameters n_e and p_e. We will show this can be done with a process that takes $T = O(m)$ time with high probability.

[3] To be precise, we set the probability of an edge $e \notin \bigcup_i F_i$ to exist to 0.

▶ **Lemma 21.** *With high probability, the sampling phase of Algorithm 1 takes $O(m)$ time.*

For the proof, see the full version. Concluding, the algorithm takes

$$O(m \cdot \min(\alpha(n)\log(m/n), \log(n)) + O(m) = O(m \cdot \min(\alpha(n)\log(m/n), \log(n))$$

time in total for polynomially-weighted graphs.

5 Conclusion

In this paper, we presented a faster $(1 \pm \epsilon)$-cut sparsification algorithm for weighted graphs. We have shown how to compute sparsifiers of size $O(n\log(n)/\epsilon^2)$ in $O(m \cdot \min(\alpha(n)\log(m/n), \log(n)))$ time, for integer weighted graphs. Both algorithms apply a sampling technique where the MSF index is used as a connectivity estimator.

We have shown that we can compute an M-partial MSF packing in $O(m\alpha(m)\log(M))$ time for polynomially-weighted graphs. For graphs with unbounded integer weights, we have shown that we can compute a complete MSF packing in $O(m\log(n))$ time, and a sufficient estimation of an M-partial MSF packing can be computed in time $O(m\alpha(m)\log(M))$. An open question is whether a more efficient computation is possible. This would improve on our sparsification algorithm, but might also be advantageous in other applications. The NI index has shown to be useful in various applications. We believe to have shown that the MSF index is a natural analogue.

To develop an algorithm to compute an MSF packing, one might be inclined to build upon one of the algorithms that compute a minimum spanning tree faster than Kruskal's algorithm, such as the celebrated linear-time algorithm of Karger, Klein, and Tarjan [19]. However, this algorithm and many other fast minimum spanning tree algorithms make use of edge contractions. It is far from obvious how to generalize this to a packing: in that case, we need to work simultaneously on multiple trees, hence we cannot simply contract the input graph in favor of any single one. To make this work, a more meticulous use of data structures seems necessary.

Computation of the MSF indices in linear time would be an ultimate goal. However, for our application a slightly looser bound suffices. If we can reduce the running time to compute the MSF indices to $O(m + n\log(n))$, then we obtain a time bound of $O(m)$ for cut sparsification. Moreover, we do not need the exact MSF index, an estimate suffices. This can either be a constant-factor approximation of the MSF index for each edge, or an estimate in the weights used in the forests, as done for graphs with unbounded weights in the full version.

References

1 Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. On fully dynamic graph sparsifiers. In *Proc. of the Symposium on Foundations of Computer Science (FOCS)*, pages 335–344, 2016.

2 Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993.

3 Alexandr Andoni, Jiecao Chen, Robert Krauthgamer, Bo Qin, David P Woodruff, and Qin Zhang. On sketching quadratic forms. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 311–319, 2016.

4 Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Structures & Algorithms*, 30(4):532–563, 2007. doi:10.1002/rsa.20130.

5 Joshua Batson, Daniel A Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721, 2012.

6 András A Benczúr and David R Karger. Approximating st minimum cuts in $\tilde{O}(n^2)$ time. In *Proc. of the Symposium on Theory of Computing (STOC)*, pages 47–55, 1996.

7 András A Benczúr and David R Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM Journal on Computing*, 44(2):290–319, 2015.

8 Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, 23(4):493–507, 1952.

9 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

10 Michael Dinitz, Robert Krauthgamer, and Tal Wagner. Towards resistance sparsifiers. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 40 of *LIPIcs*, pages 738–755, 2015.

11 Michael Elkin and Ofer Neiman. Efficient algorithms for constructing very sparse spanners and emulators. *ACM Transactions on Algorithms*, 15, July 2016. doi:10.1145/3274651.

12 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. doi:10.1145/1993636.1993647.

13 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.

14 Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Minimum Cut in $O(m \log^2 n)$ Time. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *Proc. of the International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 57:1–57:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.57.

15 Mohsen Ghaffari, Krzysztof Nowicki, and Mikkel Thorup. Faster algorithms for edge connectivity via random 2-out contractions. In *Proc. of the Symposium on Discrete Algorithms (SODA)*, pages 1260–1279, 2020.

16 Ramesh Hariharan and Debmalya Panigrahi. A general framework for graph sparsification, 2010. arXiv:1004.4080.

17 David R Karger. Global min-cuts in $\mathcal{RNC}$, and other ramifications of a simple min-cut algorithm. In *Proc. of the Symposium on Discrete Algorithms (SODA)*, volume 93, pages 21–30, 1993.

18 David R Karger. Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research*, 24(2):383–413, 1999.

19 David R Karger, Philip N Klein, and Robert E Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM*, 42(2):321–328, 1995.

20 David R Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 43(4):601–640, 1996.

21 Ioannis Koutis and Shen Chen Xu. Simple parallel and distributed algorithms for spectral graph sparsification. *ACM Trans. Parallel Comput.*, 3(2):14:1–14:14, 2016. doi:10.1145/2948062.

22 Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. of the American Mathematical Society*, 7(1):48–50, 1956.

23 Yin Tat Lee and He Sun. An sdp-based algorithm for linear-sized spectral sparsification. In *Proc. of the Symposium on Theory of Computing (STOC)*, pages 678–687, 2017.

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

25 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.

26 David Peleg and Alejandro A Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.

27 Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

28 Daniel A Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

29 Robert E Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, April 1975. doi:10.1145/321879.321884.

A Review: A First Application of the Sparsification Framework

In this section, we review the application of the framework of Section 2.1 with *Nagamochi-Ibaraki (NI) indices* as parameters, as presented by Fung et al. [13]. As the name suggests, NI indices were first introduced by Nagamochi and Ibaraki [24, 25]. The algorithm they provide gives a graph partitioning into forests, and subsequently a corresponding index for each edge, called the NI index.

▶ **Definition 22.** *Let $G = (V, E)$ be a graph, possibly weighted. We say an edge-disjoint sequence $F_1, F_2, \ldots$ of forests is a* Nagamochi-Ibaraki forest packing *for G if F_i is a spanning forest for $G \setminus \bigcup_{j=1}^{i-1} F_i$, where the weights of $\bigcup_{j=1}^{i-1} F_i$ are subtracted of G. If G is a weighted graph, each edge e must be contained in $w(e)$ contiguous forests. We define the* NI index*, denoted by l_e, to be the index of the (last if weighted) forest in which e appears.*

Nagamochi and Ibaraki show that the NI indices can be computed in linear time for unweighted graphs and in $O(m + n\log(n))$ time for weighted graphs, see [25, 24]. As is shown in [13], we can use the NI index as the connectivity estimator in the sparsification framework to obtain the following result.

▶ **Theorem 23.** *Let $G = (V, E)$ be a weighted graph, and let $\epsilon > 0$ be a constant. Let G_ϵ be obtained by independently compressing each edge with parameter $p_e = \min(1, \rho/l_e)$, where $\rho = \frac{224}{0.38}\ln(n)/\epsilon^2$. Then G_ϵ is a $(1 \pm \epsilon)$-cut sparsifier for G with high probability.*

The sampling itself takes at most $O(m)$ time, as explained in Lemma 21. As the NI indices can be computed in $O(m + n\log(n))$ time, this implies that the total running time is $O(m+n\log(n))$. As a graph with $m \leq n\log(n)$ is already sparse, we can assume $m > n\log(n)$. Thus, for our purposes, the total running time is simply $O(m)$.

Next we provide a bound for the number of edges in the sparsifier G_ϵ. Fung et al. [13] prove this same bound in expectation, we provide a proof for this bound "with high probability".

▶ **Lemma 24.** *With high probability, the size of the graph G_ϵ in Theorem 23 is $O(n\log^2(n)/\epsilon^2)$.*

Proof. Let $v \in V$ be a vertex with degree $d_v \geq O(\log^2(n)/\epsilon^2)$ in G. We denote the degree of v in G_ϵ by d'_v and we write $d' := \max_{v\in V} d'_v$. For each neighbor u of v in G, we compress the edge $e = (u, v)$ with parameter $p_e = \min\left(1, \frac{224\ln(n)}{0.38\epsilon^2 l_e}\right)$, where l_e is the NI index of e. For each edge, the probability that it remains after compression is $1 - (1 - p_e)^{w_e}$. From Bernoulli's inequality we see $1 - (1 - p_e)^{w_e} \leq w_e p_e$. Let Y_e be the random variable that is 1 if e remains, and 0 else. We note that $\mathbb{E}\left[\sum_{e:v\in e} Y_e\right] \leq \frac{224}{0.38}\ln^2(n)/\epsilon^2$. Now we apply a Chernoff bound (Theorem 26) to obtain

$$\mathbb{P}\left[d'_v \geq \delta\frac{224}{0.38}\ln^2(n)/\epsilon^2\right] \leq \exp\left(-0.38\delta\frac{224}{0.38}\ln^2(n)/\epsilon^2\right) = n^{-224\delta\ln(n)/\epsilon^2}.$$

Using a union bound we get the desired result

$$\mathbb{P}\left[d' \leq \delta \frac{224}{0.38} \ln^2(n)/\epsilon^2\right] \geq 1 - n^{1-224\delta \ln(n)/\epsilon^2}.$$

Consequently, we obtain that with high probability the number of edges of the sparsifier is at most $O(n \log^2(n)/\epsilon^2)$. ◀

The state of the art for polynomially-weighted graphs is achieved by postprocessing this result with the algorithm by Benczúr and Karger [7]. Thus our improvement on Benczúr and Karger leads to an overall improved result.

B Tail bounds

To analyze the sampling methods used in Section 4, we make use of the well-known Chernoff bound to get a grasp on the tail of various distributions [8].

▶ **Theorem 25.** *Let $Y_1, \ldots, Y_n$ be n independent random variables such that each Y_i takes values in $[0, 1]$. Let $\mu = \sum_{i=1}^{n} \mathbb{E}[Y_i]$ and $\xi = 2\ln(2) > 0.38$. Then for all $\epsilon > 0$*

$$\mathbb{P}\left[\left|\sum_{i=1}^{n} Y_i - \mu\right| > \epsilon\mu\right] \leq 2\exp\left(-\xi \min(\epsilon, \epsilon^2)\mu\right).$$

At times, the expected value μ itself is not known. Fortunately an upper bound on the expected value also suffices.

▶ **Theorem 26.** *Let $Y_1, \ldots, Y_n$ be n independent random variables such that Y_i takes values in $[0, 1]$. Let $\mu = \sum_{i=1}^{n} \mathbb{E}[Y_i]$ and $\xi = 2\ln(2) > 0.38$. Suppose $\mu' \geq \mu$. Then for all $\delta \geq 2$*

$$\mathbb{P}\left[\sum_{i=1}^{n} Y_i > \delta\mu'\right] \leq 2\exp\left(-\xi(\delta - 1)\mu'\right).$$

Proof. Let $\epsilon := (\delta - 1)\frac{\mu'}{\mu}$. We have $\epsilon \geq 1$, so $\min(\epsilon, \epsilon^2) = \epsilon$. The statement now follows directly from Theorem 25. ◀

C Reduction from Real to Integer Weights

In this section, we show how to reduce the computation of a cut sparsifier of a graph with non-negative real weights to integer weights, formalizing the procedure sketched by Benczúr and Karger [7]. Let $G = (V, E, w)$ be a weighted graph, where $w\colon E \to \mathbb{R}$. Denote $W_{\max} := \max_{e \in E} w(e)$ and $W_{\min} := \min\left\{1, \min_{e \in E} w(e)\right\}$. Then the reduction consists of the following steps:

1. Compute $W_{\min}$ and $r := -\lfloor \log(\frac{\epsilon}{2} W_{\min}) \rfloor$.
2. Create $w'\colon E \to \mathbb{R}$ by rounding the weights $w(e)$ to the closest multiple of 2^{-r}, and define $G' := (V, E, w')$.
3. Create $\hat{w}\colon E \to \mathbb{R}$ by $\hat{w}(e) := 2^r w'(e)$.
4. Compute a $(1 \pm \epsilon/3)$-cut sparsifier $\hat{H} = (V, E_H, \hat{w}_H)$ of $\hat{G} = (V, E, \hat{w})$.
5. Output $H = (V, E_H, w_H)$ where $w_H(e) := 2^{-r}\hat{w}_H(e)$.

First, we show that the graph H is indeed a $(1+\epsilon)$-cut sparsifier of G. Hereto, we note that for any cut C we have

$$w_H(C) = 2^{-r} w_{\hat{H}}(C) \leq 2^{-r}(1+\epsilon/3) w_{\hat{G}}(C) = (1+\epsilon/3) w_{G'}(C) \leq (1+\epsilon) w_G(C),$$

where the last inequality holds as each weight $w'(e)$ has at most an additive error of $2^{-r} \leq \frac{\epsilon}{2} W_{\min} \leq \frac{\epsilon}{2}$ with respect to $w(e)$, hence at most an multiplicative error of $\frac{\epsilon}{2}$. Analogously we obtain $w_H(C) \geq (1-\epsilon) w_G(C)$.

By construction, $\hat{G}$ has integer weights, which are bounded by $O(\frac{W_{\max}}{\epsilon W_{\min}})$. Steps 1, 2, 3, and 5 can be implemented in $O(m)$ time. So indeed we have reduced the problem to finding a cut sparsifier of a graph with integer weights. Moreover, note that if G has polynomially bounded real weights, in the sense that $W_{\max} = O(\text{poly}(n))$ and $W_{\min} = \Omega(1/\text{poly}(n))$, then the graph $\hat{G}$ has polynomially bounded integer weights. We can state this independent of ϵ, since for $\epsilon \leq 1/m$ we can always output the entire input graph as a cut sparsifier of optimal size $O(n/\epsilon^2)$ [3].