Learning Spanning Forests Optimally in Weighted Undirected Graphs with CUT queries

Deeparnab Chakrabarty DEEPARNAB@DARTMOUTH.EDU
Department of Computer Science, Dartmouth College

Hang Liao HANG.LIAO.GR@DARTMOUTH.EDU
Department of Computer Science, Dartmouth College

Editors: Claire Vernade and Daniel Hsu

Abstract

In this paper we describe a randomized algorithm which returns a maximal spanning forest of an unknown *weighted* undirected graph making $O(n)$ CUT queries in expectation. For weighted graphs, this is optimal due to a result in [Auza and Lee, 2021] which shows an $\Omega(n)$ lower bound for zero-error randomized algorithms. These questions have been extensively studied in the past few years, especially due to the problem's connections to symmetric submodular function minimization. We also describe a simple polynomial time deterministic algorithm that makes $O(\frac{n \log n}{\log \log n})$ queries on undirected unweighted graphs and returns a maximal spanning forest, thereby (slightly) improving upon the state-of-the-art.

Keywords: Cut-queries, Query Algorithms, Randomized Algorithms, Weighted Spanning Forest.

1. Introduction

Learning an unknown graph via queries has been extensively studied for more than two decades. The general setting is this: there is an undirected graph $G = (V, E)$ whose vertices are known but edges are unknown. Certain kinds of queries are allowed on this graph. The goal is to reconstruct the graph with as few queries as possible. Such active learning questions also have applications in fields such as computational biology (cf. Grebinski and Kucherov (1998)), and connections to data summarizations or sketches where the answers to the queries can be thought of as holding the "relevant information" about the graph. More generally, this question falls under the umbrella of *combinatorial search* (cf. Aigner (1988); Du and Hwang (2000)) which is a vast area of study that wishes to *"determine an unknown object by means of indirect questions about this object"*[1].

In this paper, we consider *cut-query* access to an unknown, undirected, *weighted/multi* graph. Every $e \in E(G)$ has an associated $w(e) > 0$. $w(e) = 0$ for $e \notin E(G)$. Given a subset $S \subseteq V$ of vertices, let ∂S be the set containing edges with exactly one endpoint in S. A CUT query takes input $S \subseteq V$ and returns the *value* $\sum_{e \in \partial S} w(e)$. CUT query based reconstruction algorithms for graphs, weighted and unweighted, have been extensively studied, and a lot of these papers indeed appeared in the ML/AI community, eg, (Angluin and Chen (2008), Choi and Kim (2010), Bshouty (2009), Choi (2013)). Instead of focusing on the graph reconstruction question (which has been almost fully resolved; see Section 1.2), we ask in how few queries can one decide if G is connected, or more generally, find a *maximal* spanning forest[2] in G. The question of connectivity

1. quote from Grebinski and Kucherov (2000)
2. We caution the reader that we are not finding a *maximum weight* spanning forest with respect to these $w(e)$'s. Finding even a good approximation to the maximum weight spanning forest can be shown to need $\widetilde{\Omega}(n^2)$ queries whose proof, although not difficult, is out of scope of this paper.

has seen a lot of interest in the recent years (cf. Rubinstein et al. (2018); Graur et al. (2020); Lee et al. (2021); Auza and Lee (2021); Assadi et al. (2021); Apers et al. (2022); Chakrabarty and Liao (2023)) mainly due to the connections to streaming and sketching, but also due to connections to submodular function minimization (SFM). The cut-function of an undirected graph is a well known (symmetric) submodular function, and such functions can be (non-trivially) minimized using $O(n^3)$ queries (Queyranne (1998)) deterministically, and in $\tilde{O}(n^2)$ queries using randomization (Chekuri and Quanrud (2021)). On the other hand, there is no $\omega(n)$ lower bound known for this question. This led to recent interest in understanding if for the special case of undirected graph connectivity, can one design $O(n)$-query algorithms, or is there a super-linear lower bound.

It is not too hard (cf. Harvey (2008), Theorem 5.10) to design a deterministic $O(n \log n)$ query algorithm to find a spanning forest of an unknown, weighted, undirected graph via mimicking a Prim-style algorithm using a binary-search style idea. To the best of our knowledge (cf. Apers et al. (2022), Table 1), this is still the best known result for deterministic algorithms, even for unweighted undirected graphs. Apers et al. (2022) gives a randomized, zero-error algorithm for this problem on *unweighted* graphs, which makes $O(n)$ queries in expectation. However, as argued in Chakrabarty and Liao (2023), this algorithm used the unweightedness quite crucially, and this latter paper gave an $O(n \log\log n \cdot (\log\log\log n)^2)$-query randomized Monte Carlo algorithm that solved the connectivity question on weighted graphs with constant probability (which is weaker than a zero-error algorithm). It was left open to match Apers et al. (2022) result for weighted graphs. The main result of this paper is an affirmative resolution.

Theorem 1 *Given* CUT *query access to an unknown weighted undirected graph $G = (V, E, w)$ with non-negative weights, there is a polynomial time Las Vegas algorithm returning a maximal spanning forest of G that makes $O(n)$ queries in expectation.*

The query complexity of our algorithm is optimal up to a constant factor; Auza and Lee (2021) prove that any zero-error randomized algorithm to detect whether a graph is connected or not, even when the query model is a much stronger[3] query model than the CUT query model, needs to make at least $\Omega(n)$ many such queries. To the best of our knowledge, it is the only regime of this problem where the upper and lower bounds are tight up to constants, and completes the story for learning maximal spanning forests in weighted undirected graphs using zero-error randomized algorithms.

Our second result is an improved *deterministic* algorithm for undirected, unweighted graphs.

Theorem 2 *Given* CUT *query access to an unknown unweighted undirected graph $G = (V, E)$, there is a deterministic algorithm that makes $O(\frac{n \log n}{\log\log n})$ queries and returns a maximal spanning forest in G.*

This (slightly) improves upon the $O(n \log n)$-query algorithm mentioned in Harvey (2008). It is known that Harvey's algorithm can also work with the much weaker "OR" query model where one only gets to know if the cut value is zero or positive, and in this model, via a connection to communication complexity of graph connectivity (Hajnal et al. (1988)), an $\Omega(n \log n)$ lower bound is known for deterministic algorithms. Our algorithm shows that with the stronger CUT queries one cannot obtain the same lower bound, and thus opens up possibilities of much better deterministic algorithms.

3. They consider the LINEAR model where one specifies a $\binom{n}{2}$-dimensional query vector q and obtains the answer $\sum_e q(e)w(e)$.

Perspective. The problem of finding a maximal spanning forest and deciding the connectivity of a graph is a classic algorithmic question which has been studied in many models of computation including dynamic (Kapron et al. (2013); Duan and Zhang (2017)), streaming (Ahn et al. (2012); Nelson and Yu (2019)), and parallel (Andoni et al. (2018); Behnezhad et al. (2019)) computing. Our first result, Theorem 1, gives a tight understanding of the problem in terms of query complexity, closing the gap between $O(n \log \log n)$ and $O(n)$ left by previous works. Quantitatively, this may not seem like a big improvement, but it has qualitative value in (a) being an end-of-the-line study for this problem, (b) ruling out cut functions on weighted undirected graphs as candidate lower bounds for symmetric SFM, and (c) as we explain in the next subsection, leading to a new algorithmic technique which may be helpful in other problems. Furthermore, our second result, Theorem 2, shows that for deterministic algorithms, the simple $O(n \log n)$-query is not optimal, and therefore can lead to more interest in coming up with deterministic strategies for solving this problem.

1.1. Technical Contribution

In this section, we give a technical overview of both our results. The maximal spanning forest algorithms in our and all previous papers use the following "Borůvka style" framework: begin with a collection of n singleton connected components and in phases merge connected components till one gets a maximal spanning forest. What differs is what each phase does.

Randomized Algorithm (Theorem 1) approach. A phase in our algorithm is a randomized algorithm that takes input a *weighted* graph with t connected components consisting of learnt edges, and then performs queries to discover new edges so that the number of connected components go down to ct for some constant $c < 1$. The whole creativity lies in how to do this using only $O(t)$ queries in expectation. Once we have this, a simple geometric sum gives the $O(n)$ query algorithm in expectation: the algorithm proceeds in $O(\log n)$-rounds making $O(n + cn + c^2 n + \ldots) = O(n)$ many queries in all. We note here that Apers et al. (2022) obtains the same result on unweighted graphs, but our algorithm is *different* and not just a generalization of their algorithm.

To illustrate and underscore our main technical contribution, in Section 3.1, we first focus on a simpler version of the problem which is, in some sense, the first phase. We assume our unknown, *weighted*, undirected graph $G = (R \cup B, E)$ is *bipartite*[4] with vertices being red, R, or blue, B, and $|R| = |B| = n$, and every vertex $r \in R$ has at least one neighbor in B and we begin with no knowledge of the edges. The goal is to design a randomized algorithm (Algorithm 1) which makes $O(n)$ queries in expectation, and learns at least one edge incident to every red vertex, thereby leading to a new graph with $\leq n$ connected components (down from the $2n$ singletons we began with). This forms the heart of the final maximal spanning forest algorithm, which is described in Section 3.2.

Before we describe the idea behind our algorithm, it is worthwhile describing the idea in Apers et al. (2022) in unweighted graphs (who also solve the above problem) and why it fails to generalize with weights. Their algorithm first queries the number of blue neighbors for every red vertex using $O(n)$ CUT queries; this *crucially* uses that the graph is unweighted. After this, they partition the vertices of R into $\lceil \log_2 n \rceil$ classes where class i contains vertices with *degree* $\approx 2^i$, and for each of these classes, the algorithm samples a subset of vertices from B with probability $\approx 1/2^i$. It's not hard to show that $\Theta(1)$ of the vertices in R has exactly one neighbor in the corresponding sampled subset in B; in other words, with high probability there is a matching of size cn for some $c < 1$. At

4. We obtain this bipartite graph via a random bipartition, which is described subsequently in Section 3.2.

this point, Apers et al. (2022) uses an algorithm by Grebinski and Kucherov (2000) which gives an $O(n)$ query algorithm to recover all the edges of this matching.

The Degree Issue with Weights and Our Bypass. As noted in Chakrabarty and Liao (2023), the main issue in implementing the above method in weighted graphs is in the first step of figuring out the number of blue neighbors (let's call this the blue-degree) of a red node. While this is a near triviality in unweighted graphs, with weighted graphs there are some provable hardness (see Chakrabarty and Liao (2023)). Indeed, the lion's share of the Monte-Carlo $O(n \log\log n \cdot (\log\log\log n)^2)$ query algorithm in Chakrabarty and Liao (2023) is spent in *estimating* the blue-degrees of every red node. It is left as an open question whether this can be done in $O(n)$ time, and if so, their algorithm could perhaps be modified to give an $O(n)$ time algorithm. We do not resolve this "degree-estimation" question; indeed, very recently, Chakraborty et al. (2022) (cf. Theorem 4.1) prove a super-constant hardness on the problem of estimating the degree and thus this route possibly cannot give a $O(n)$-algorithm. Rather, *our main insight is that the above idea of "sampling inversely proportional to degree" can be morally simulated even without knowing the degrees.* This idea could potentially be useful in other applications.

Our Algorithm in a Nutshell. We proceed in $\lceil \log_2 n \rceil$ iterations. We maintain a subset $\mathcal{R} \subseteq R$ of red vertices for which we haven't found a blue neighbor, and initially $\mathcal{R} = R$. In the first iteration, we sample a subset $\mathcal{B} \subseteq B$ where every vertex is present with probability $1/n$. Then, we use known graph reconstruction algorithms (Theorem 4.A or Theorem 4.B) to learn the edges in the subgraph $E(\mathcal{R}, \mathcal{B})$. Since $|\mathcal{B}|$ is small, this is a sparse subgraph, and the number of queries needed is small. Next, we *remove* every vertex in $\mathcal{R}$ for which we have found an edge and proceed to the next iteration. Now we sample $\mathcal{B} \subseteq B$ with probability $2/n$, and repeat the same procedure, always removing vertices from $\mathcal{R}$. In the ith iteration, the sampling probability is $2^i/n$, and therefore in the $\log n$th iteration, we reconstruct the graph $E(\mathcal{R}, B)$ where recall $\mathcal{R}$ is the subset of vertices of the original R for whom we haven't discovered an edge. And so, by the end of these iterations, we would've learnt at least one neighbor for every vertex in R completing what we set out to do.

Why is the query complexity of the above algorithm small? Indeed, the worry is that when $\mathcal{B}$ is as big as $\Omega(n)$, the graph between $\mathcal{R}$ and $\mathcal{B}$ may no longer be sparse. We prove (see Lemma 6) that this cannot be the case by noting that by the time $\mathcal{B}$ is "large", all the high-degree vertices in the original $\mathcal{R}$ would already have been removed. In particular, if the degree of a red vertex r is $d(r)$, which remember is something that we don't know, then this would have been removed by the $\approx \log_2(n/d(r))$th iteration. For instance, if $|\mathcal{B}| = \Theta(n)$, then $\Theta(\log n)$ rounds of the process must have passed, and it's highly likely that the only vertices remaining in $\mathcal{R}$ would have degree $\Theta(1)$. And so, in this iteration the graph is sparse as well. This explains the key new idea behind our randomized algorithm and is formally proven in Lemma 7.

Deterministic Algorithm (Theorem 2) Idea. Our deterministic algorithm is actually pretty simple and stems from the observation that with CUT queries one can learn neighborhoods of "high" degree vertices paying $\ll \log n$ queries per vertex. The algorithm keeps doing this and growing connected components BFS style till the number of edges across the components become much smaller than $O(n \log n)$. Since the graph is unweighted, this estimate can be maintained. Once the graph becomes this sparse, once again exploiting the power of CUT queries, the whole graph can be reconstructed using known results. The latter step requires some non-trivial work since naively we only obtain the information about which pairs of components have an edge between them, which

is enough for answering the question whether a graph is connected or not; finding the true edges requires the full power of the CUT-queries. Balancing these two ideas gives an $O\left(\frac{n \log n}{\log \log n}\right)$ query algorithm. We don't believe this is the correct answer, but it is perhaps a first step in obtaining $\ll n \log n$ query algorithms a la Harvey (2008), which, recall, would also work with even a weaker "OR" query model and for which the query complexity is optimal. Details of this algorithm can be found in Section 4.

1.2. Related Works

The question of reconstructing the whole graph using CUT queries has almost been resolved after a long series of works (cf. Grebinski and Kucherov (2000); Alon et al. (2004); Alon and Asodi (2005); Reyzin and Srivastava (2007); Choi and Kim (2010); Mazzawi (2010); Bshouty and Mazzawi (2011b, 2012); Choi (2013)). For unweighted undirected graphs with m edges, Choi and Kim (2010) proved the existence of non-adaptive deterministic algorithms making $O(\frac{m \log \frac{m}{n}}{\log m})$ queries, and Mazzawi (2010) described an efficient adaptive deterministic algorithm with similar query complexity. This query complexity is information theoretically optimal. For (non-negative) weighted graphs, Bshouty and Mazzawi (2011a) prove the existence of non-adaptive $O(\frac{m \log n}{\log m})$-query algorithms. Then in Bshouty and Mazzawi (2011b) the authors describe a deterministic $O(\frac{m \log n}{\log m} + m \log \log m)$-query adaptive algorithm. If we allow randomization, then Choi (2013) gives an efficient, randomized $O(\frac{m \log n}{\log m})$ query algorithm which is information theoretically tight as well. For our efficient randomized algorithm, we use this result. It is an open question to design efficient non-adaptive algorithms to reconstruct using optimal query complexity.

The question of studying just whether an undirected graph is connected using CUT queries was perhaps first explicitly noted in Harvey (2008) due to connections to submodular function minimization. The same question for minimum cuts was initiated by the paper Rubinstein et al. (2018) who described an $O(n\text{polylog } n)$ algorithm to find the minimum cut. In fact, the paper of Apers et al. (2022) mentioned above, gives a randomized Monte Carlo $O(n)$-query algorithm to solve this problem. In unweighted graphs, it is not known whether this query complexity is tight, and the only lower bound is an $\Omega(n \log \log n / \log n)$-lower bound that follows from the communication complexity of connectivity result of Raz and Spieker (1995). For weighted graphs, the best known algorithm to solve the minimum-cut is a randomized Monte Carlo $O(n\text{polylog } n)$-query algorithm by Mukhopadhyay and Nanongkai (2020). There is no $O(n)$ algorithm known for finding the minimum cut in the weighted case.

Graph reconstruction has also been studied under other query models. In particular, there is the weak "OR" query model (also known as independent set (IS) queries), where one asks for a subset S and obtains only the information whether S is an independent set or not. A slightly different model called the bipartite independent set (BIS) query passes two disjoint subsets A, B and obtains whether there exists an edge with one endpoint in A and one endpoint in B. This model is more related to "group testing" question (Dorfman, 1943) where one only gets a weak signal from a query. Angluin and Chen (2008) described an $O(m \log n)$-query algorithm to learn the whole graph using IS queries, and this is information theoretically tight. The question of reconstructing the spanning tree from such weaker models has also been studied. It is not hard to see an $\Omega(n^2)$ lower bound[5]

5. Consider a graph on $2n$ vertices where we have two cliques on n vertices connected with a single edge. Any IS query containing more than one vertex from any of the parts gives no information. So, the problem reduces to finding a 1 in an n^2-dimensional vector where we can only query singletons.

for IS queries. Furthermore, the algorithm in Harvey (2008) works with BIS queries, and indeed, as mentioned above, in this weaker model this factor is tight. In Assadi et al. (2021), the authors study the rounds-vs-query trade-off for the problem of learning a spanning tree for BIS queries, and give a near sharp resolution. Finally, other properties such as estimating the number of edges (Beame et al. (2020); Chen et al. (2020); Addanki et al. (2022), etc) have been studied under these models, and more generally this area of understanding the query complexity of learning/estimating graph parameters under a variety of models is a rich and relevant area of study.

2. Preliminaries and Subroutines

We begin by defining two other notions of queries which have been used in the literature and is often more convenient to use. We start with the notion of CROSS queries. Given a graph $G = (V, E, w)$ and two disjoint subsets $A, B \subseteq V$, $\mathsf{CROSS}(A, B)$ returns $\sum_{e \in E(A,B)} w(e)$, where $E(A, B)$ is the set of vertex pairs $(a, b) \in A \times B$ such that $(a, b) \in E$. It's easy to see the following.

Proposition 3 *A* CROSS *query can be simulated by 3* CUT *queries.*

Proof $\mathsf{CROSS}(A, B) = \frac{1}{2} \cdot (\mathsf{CUT}(A) + \mathsf{CUT}(B) - \mathsf{CUT}(A \cup B))$. ■

We then state the graph reconstruction results with optimal query complexity and the minor modifications we need to make for our purposes. The first result is a deterministic, non-adaptive query algorithm by Bshouty and Mazzawi (2011a) with optimal query complexity.

Theorem 4.A *(Follows from Bshouty and Mazzawi, 2011a, Corollary 4). There exists an adaptive, deterministic algorithm* GR1 *which takes input a bipartite graph $G = (U, V)$ on n vertices and m edges such that U and V are known but m is unknown to the algorithm, and reconstructs the edges of G along with their weights, making $\frac{C_{\mathsf{GR1}} m \log n}{\log m}$* CROSS *queries, for some constant $C_{\mathsf{GR1}} > 0$.*

Theorem 4.B *(Follows from Choi, 2013, Theorem 1). There exists an adaptive, randomized algorithm* GR2 *which takes input a bipartite graph $G = (U, V)$ on n vertices and m edges such that U and V are known but m is unknown to the algorithm, and either reconstructs the edges of G along with their weights, or aborts. The probability that the algorithm aborts or makes more than $\frac{C_{\mathsf{GR2}} m \log n}{\log m}$* CROSS *queries, for some absolute constant $C_{\mathsf{GR2}} > 0$, is at most $O(\frac{\log m}{m})$.*

Proof See Appendix A. ■

The next subroutine we need is the simple DFS-style algorithm to find a spanning forest. In particular, if the number of connected components is q, then the remaining edges in the spanning forest can be found in $O(q \log n)$ many queries. This was also used by Apers et al. (2022) for unweighted graphs but an inspection of their proof shows that it readily works with weights as well. The idea is that a single element in the support can be found using binary search, and every connected component can find an edge coming out of it (if they exist) in $O(\log n)$ queries.

Lemma 5 *(Paraphrasing Apers et al., 2022, Lemma 5.1) Let $G = (V, E)$ be an n-vertex weighted graph with non-negative weights. Let G' be a contraction of G with q many supervertices, which are given explicitly as the partition $P = A_1, \cdots, A_q$ of V. There is a deterministic algorithm* DFSSpanningForest *that takes in G, G' and outputs a set of edges $F \subseteq E$ that form a spanning forest of G' and makes $O(q \log n)$* CUT *queries to G.*

3. Zero Error Randomized $O(n)$-Query Algorithm

As mentioned in Section 1.1, to underscore our main technical contribution, in Section 3.1 we focus on the problem of given a bipartite weighted graph $G = (R \sqcup B, E)$ with $2n$ singletons with the promise that every vertex in R has a neighbor in B, how to make $O(n)$ queries in expectation to learn a subgraph where we learn at least one neighbor incident to every vertex in R. Thus, the number of connected components go down from $2n$ to $\leq n$. This algorithm is described as Algorithm 1.

The generalization of this that is the subroutine to the spanning forest algorithm is Algorithm 2. This algorithm takes a graph with t connected components and makes $O(t)$ queries in expectation to learn edges which leads to $< ct$ connected components for some $c < 1$. This then can be plugged into the Apers et al. (2022) framework to give the spanning forest algorithm. This is described in Section 3.2.

3.1. The Main Idea: Learning edges in a bipartite graph

We assume our unknown, weighted, undirected graph $G = (R \sqcup B, E)$ is *bipartite* with vertices being red, R, or blue, B, and $|R| = |B| = n$, and every vertex $r \in R$ has at least one neighbor in B. We also know the bipartition. The goal is to learn at least one edge incident on every red vertex. In this section, we describe a randomized algorithm to do this making $O(n)$ queries in expectation. As explained in Section 1.1, this forms the main heart of the spanning forest algorithm since these learnt edges reduces the number of connected components by a constant factor.

The algorithm proceeds in $\lceil \log_2 n \rceil$ iterations. It maintains a subset $\mathcal{R}_i \subseteq R$ with $\mathcal{R}_0 = R$ which is supposed to signify the subset of red vertices which hasn't discovered an edge incident on them. In the ith iteration, the algorithm samples a subset blue vertices $\mathcal{B}_i \subseteq B$ at a rate $\frac{2^i}{n}$. We then reconstruct all the edges in $E(\mathcal{R}_i, \mathcal{B}_i)$. Then, we remove any vertices r in $\mathcal{R}_i$ participating in any of these reconstructed edges since we have already found an edge incident on them, to get $\mathcal{R}_{i+1}$. Note that in the final round $\ell := \lceil \log_2 n \rceil$, the subset $\mathcal{B}_\ell = B$, and thus we would reconstruct all the edges between $E(\mathcal{R}_\ell, B)$. In particular, since all vertices in $\mathcal{R}_\ell$ have a neighbor in B, we would succeed with probability 1.

Algorithm 1 SkeletonReduceConnectedComponents

Input: CUT access to a bipartite graph $G = (R \sqcup B, E)$ with $|R| = |B| = n$ and unknown edges with positive weights. Assumption: each vertex in R each has a neighbor in B.

Output: Set of edges with at least one edge incident on every $r \in R$.

$\mathcal{R}_0 \leftarrow R$.
$i \leftarrow 0$.
for $i \leq \lceil \log_2 n \rceil$ **do**
  Sample a set B_i with every $b \in B$ sampled with probability $\frac{2^i}{n}$.
  Recover edges $E_i := E(\mathcal{R}_i, B_i)$ using GR1 in Theorem 4.A.
  $\mathcal{R}'_i = \{v \in \mathcal{R}_i |\ v \text{ is an endpoint of some } e \in E_i\}$.
  $\mathcal{R}_{i+1} \leftarrow \mathcal{R}_i - \mathcal{R}'_i$.
  $i \leftarrow i + 1$.
end
return $\bigsqcup_{i \in [\lceil \log n \rceil]} E_i$.

Theorem 6 *The expected number of recovered edges in Algorithm 1 is* $\mathbf{E}[|\bigsqcup_{i \in [\lceil \log n \rceil]} E_i|] \leq 5n$.

Define $d(r) : R \to [n]$ to be a function maps from $r \in R$ to the number of r's neighbors in B. Note that we don't know how to obtain or even estimate $d(r)$ in $O(1)$ CUT queries, and this definition is *only for analysis*. The proof of the above theorem follows almost immediately from the next lemma: $\mathbf{E}[|\bigsqcup_{i\in[\lceil \log n\rceil]} E_i|] \le \sum_{r\in R}\sum_{b:(r,b)\in E}\frac{5}{d(r)} = 5n$.

Lemma 7 *Fix an edge $e = (r, b) \in E$ where $r \in R, b \in B$.* $\mathbf{Pr}[e \in \bigsqcup_{i\in[\lceil \log n\rceil]} E_i] \le \frac{5}{d(r)}$.

Proof Let $\mathcal{E}_i$ denote the event that $e \in E_i$, that is, the reconstructed graph in the ith round. Our goal is to show $\mathbf{Pr}[\bigsqcup_i \mathcal{E}_i] \le \frac{5}{d(r)}$.

For the event $\mathcal{E}_i$ to occur, the vertex r must have "survived" the first $(i-1)$ rounds (that is, no edge incident to r was discovered before) and the vertex b must be in the set $\mathcal{B}_i$. To this end, for any $0 \le j \le i-1$, let $\mathcal{F}_j$ be the event that no neighbor of r is in $\mathcal{B}_j$. First note that

$$\mathbf{Pr}[\mathcal{F}_j] = \left(1 - \frac{2^j}{n}\right)^{d(r)}$$

since none of its $d(r)$ neighbors are present in $\mathcal{B}_j$. Since all the $\mathcal{F}_j$'s, $0 \le j \le i-1$, and the event $b \in \mathcal{B}_i$ are mutually independent, we get that

$$\mathbf{Pr}[\mathcal{E}_i] = \mathbf{Pr}[b \in \mathcal{B}_i]\prod_{j=0}^{i-1}\mathbf{Pr}[\mathcal{F}_j] = \frac{2^i}{n}\cdot\prod_{j=0}^{i-1}\left(1-\frac{2^j}{n}\right)^{d(r)} < \frac{2^i}{n}\cdot e^{-\frac{(2^i-1)d(r)}{n}} \tag{1}$$

where we used $\forall x \ne 0, 1 + x < e^x$ and geometric series formula for the inequality.

Note that when $2^i \approx n/d(r)$, the RHS is $\approx 1/d(r)$, and for every other i either the first term in the product or the second term in the product in the RHS are orders of magnitude smaller. Therefore, the sum of $\mathbf{Pr}[\mathcal{E}_i]$'s as i ranges from 0 to $\log n$ can be bounded by $O(1/d(r))$. More precisely, let $i^* = \lfloor \log_2 \frac{n}{d(r)} \rfloor$. We consider the following two cases:

- $i \le i^*$: since $\frac{(2^i-1)d(r)}{n} \ge 0$, $e^{-\frac{(2^i-1)d(r)}{n}} \le 1$. In the summation $\sum_{i=0}^{i^*}\frac{2^i}{n}e^{-\frac{(2^i-1)d(r)}{n}}$, each term is at most $\frac{2^i}{n}$, thus

$$\sum_{i=0}^{i^*}\frac{2^i}{n}e^{-\frac{(2^i-1)d(r)}{n}} \le \sum_{i=0}^{i^*}\frac{2^i}{n} \le 2\cdot\frac{2^{i^*}}{n} \le \frac{2n}{d(r)}\frac{1}{n} = \frac{2}{d(r)}. \tag{2}$$

- $i \ge i^* + 1$: let $k = i - (i^* + 1)$. Consider the ratio between $i+1$th term and ith term,

$$\begin{aligned}\left(\frac{2^{i+1}}{n}e^{-\frac{(2^{i+1}-1)d(r)}{n}}\right) / \left(\frac{2^i}{n}e^{-\frac{(2^i-1)d(r)}{n}}\right) &= 2e^{-\frac{(2^{i+1}-1)d(r)}{n}+\frac{(2^i-1)d(r)}{n}} = 2e^{\frac{2^i d(r)-2^{i+1}d(r)}{n}} \\ &= 2e^{-\frac{2^i d(r)}{n}} \le 2e^{-\frac{2^{k+\log_2\frac{n}{d(r)}}d(r)}{n}} \qquad (3) \\ &= 2e^{-2^k}\end{aligned}$$

  where (3) follows because $i = k + (i^*+1) \ge k + \log_2\frac{n}{d(r)}$. Notice that $2e^{-2} < 0.3$ and $2e^{-2}\cdot 2e^{-4} = 0.001$. The numerical value imply that most of the mass in $\sum_{i=i^*+1}^{\lceil \log n\rceil}\frac{2^i}{n}e^{-\frac{(2^i-1)d(r)}{n}}$

is concentrated in the first two terms. One can easily check that $\sum_{i=i^*+1}^{\lceil \log n \rceil} \frac{2^i}{n} e^{-\frac{(2^i-1)d(r)}{n}} \leq \frac{3}{2} \cdot \frac{2^{i^*+1}}{n} e^{-\frac{(2^{i^*+1}-1)d(r)}{n}}$, in other words, the sum is at most $\frac{3}{2}$ of the first summand.

The first term is $\frac{2^{i^*+1}}{n} e^{-\frac{(2^{i^*+1}-1)d(r)}{n}} \leq \frac{2}{d(r)} e^{-\frac{(2^{i^*+1}-1)d(r)}{n}} \leq \frac{2}{d(r)}$. Therefore

$$\sum_{i=i^*+1}^{\lceil \log n \rceil} \frac{2^i}{n} e^{-\frac{(2^i-1)d(r)}{n}} \leq \frac{3}{d(r)}. \tag{4}$$

Equations (2) and (4) together imply $\sum_{i=0}^{\lceil \log n \rceil} \mathbf{Pr}[\mathcal{E}_i] \leq 5/d(r)$ by union bound. ■

Remark: *It is worthwhile to note that the above algorithm* doesn't *use* non-negativity *of weights. This is because of the promise that every red node has at least one blue neighbor. In the full algorithm (see Section 3.2, or Apers et al. (2022); Chakrabarty and Liao (2023)), the bipartition is formed randomly, and we abort the bipartition if less than a constant fraction of red nodes have blue neighbors. However, checking whether a red node r has a blue neighbor can be done with one* CROSS *query between r and B* only if *the weights are non-negative. With negative weights, it is not to hard to show that $\widetilde{\Omega}(n^2)$ queries would be needed to solve the connectivity question.*

3.2. Full Algorithm for Learning Spanning Forest in $O(n)$ queries

Outline. Like previous algorithms (Apers et al., 2022; Chakrabarty and Liao, 2023), our spanning forest algorithm proceeds in phases, and in each phase, the algorithm begins with t of connected components $C_1, \ldots, C_t$ and a set of learnt edges such that each C_i is connected in these learnt edges. In a phase, the algorithm discovers edges crossing these components (thereby connecting them) in such a way that $O(t)$ CUT queries are made in expectation, and with the newly discovered edges the number of connected components goes down from t to ct where $c < 1$. Simply repeating this till the number components becomes 1 leads to the $O(n)$ algorithm. The details of this is given in Algorithm 4. To understand an individual phase, let us introduce the definition of representatives of connected components a la Apers et al. (2022).

Representatives and Active/Inactive Vertices. When the connected components are no longer singletons, for each connected component C_i we need to select a vertex as the *representative* for C_i. To do so, one defines "active" and "inactive" vertices. For a fixed C_i, if there is an edge $e = (u, v)$ with $u \in C_i$ and $v \in C_j$ for any $j \neq i$, in other words, u has at least a neighbor in a different connected component, we say that the vertex u is "active". Otherwise u is "inactive". For a connected component C_i, any "active" vertex in C_i can be its representative. After a phase is run, a representative u can become "inactive". In this case, we can go through vertices in the (expanded) connected components that u belongs to, that are not "inactive", and find an active one as the new representative. It can also happen that a connected component has multiple representatives after a phase, in which case we assign an arbitrary one as the representative. Since an "inactive" vertex will remain "inactive" throughout, one can maintain a representative for each connected components with an $O(n)$ overhead on the queries. Detailed argument can be bound in the proof of Theorem 10. Henceforth, we assume that each component C_i has a *representative* vertex $c_i \in C_i$ with the guarantee that c_i has an edge to $V \setminus C_i$. If not, we learned a connected components in the final maximal spanning forest and we don't need to pass it in as part of the input.

Phase of Reducing Connected Components. The following Monte Carlo algorithm (Algorithm 2), named ReduceConnectedComponents is the general version of SkeletonReduceConnectedComponents (Algorithm 1). It describes one phase of the algorithm, where we start with t connected components and end with at most $\frac{7}{8}n$ connected components with constant probability.

Lines 2 to 6 of Algorithm 2 first creates the bipartite graph that was assumed in Algorithm 1 by coloring each connected component either red or blue uniformly at random. It is designed so as to ensure that a constant fraction of the "red representatives" have an edge to some blue-component vertex. Recall in Algorithm 1, we assumed *every* red vertex had a blue neighbor, but even if a constant fraction had them, even then there would be a constant factor drop. The rest of the algorithm is very similar to Algorithm 1 with a couple of differences: (i) we maintain a "budget" on the queries and if we ever cross it we ABORT, and thus get a Monte-Carlo algorithm, and (ii) instead of the non-constructive deterministic GR1, we replace it with the randomized Monte Carlo GR2 from Theorem 4.B, which always ensures we don't ever make $\omega(t)$ queries.

Algorithm 2 ReduceConnectedComponents

Input: G_2 with t connected components $C_1, \cdots, C_t$, each with ≥ 1 representative.

Output: Either return a graph with $\leq \frac{7}{8}t$ connected components with constant probability or ABORT.

1 $R \leftarrow \emptyset$.
2 For $i \in [t]$, pick color c_i from $\{red, blue\}$ uniformly at random. Color all vertices in C_i to c_i.
3 $B \leftarrow$ vertices colored blue.
4 $R \leftarrow$ red representatives with ≥ 1 blue neighbor. `// Takes` $O(t)$ CROSS `queries.`
5 **if** $|R| < \frac{t}{8}$ **then**
6 ABORT.
7 $i \leftarrow 0$.
8 $\mathcal{R}_0 \leftarrow R$.
9 budget $\leftarrow 120 C_{\mathsf{GR2}} t$. `// Recall` C_{GR2} `from Theorem 4.B.`
10 **for** $i \leq \log t$ **do**
11 Sample a set B_i with every $b \in B$ sampled w.p. $\frac{2^i}{t}$.
12 **while** budget > 0 **do**
13 Run GR2 as described in Theorem 4.B on $E_i := E(\mathcal{R}_i, B_i)$, always decrement budget by 1 whenever a CROSS query is made.
14 **if** budget $= 0$ **then**
15 ABORT. `// ABORT either because sampled edges >` $15t$ `or running GR2 has failed`
16 $\mathcal{R}'_i = \{v \in \mathcal{R}_i |\ v$ is an endpoint of some $e \in E_i\}$.
17 $\mathcal{R}_{i+1} \leftarrow \mathcal{R}_i - \mathcal{R}'_i$.
18 $i \leftarrow i + 1$.
  end
end
19 **return** $G_2 + \bigsqcup_{i \in [\lceil \log t \rceil]} E_i$.

Lemma 8 *Let $G = (V, E)$ be a weighted graph with non-negative weights. Suppose G_2 is a subgraph of G with t connected components for some $t \geq \frac{n}{\log n}$. Algorithm 2 makes $O(t)$ queries, and with probability $\geq 1/10$ returns a graph with $\leq \frac{7}{8}t$ connected components, or returns* ABORT.

We union bound the ABORT probability by upper bound each ABORT events. The first event in Line 6 occurs when there are too many red representatives with no blue neighbor. The second event in Line 15 occurs either because the sampling step samples too many edges or the subroutine GR2 from Theorem 4.B has failed. By choosing constants carefully, these events are bounded by a constant respectively.

Proof See Appendix A. ■

One can now easily obtain a zero-error algorithm by repeating if an ABORT is encountered.

Algorithm 3 ZeroErrorReduceCC

Input: G_2 with t connected components $C_1, \cdots, C_t$, each with ≥ 1 representative.
Output: A graph with $\leq \frac{7}{8}t$ connected components.
while *True* **do**
  $Result \leftarrow$ ReduceConnectedComponents(G_2) (Algorithm 2).
  if *Result is not* ABORT **then**
    return *Result*
end

Corollary 9 *Let $G = (V, E)$ be a weighted graph with non-negative edge weights. Given a subgraph G_2 of G with t connected components for some $t \geq \frac{n}{\log n}$, Algorithm 3 returns a graph with $\leq \frac{7}{8}t$ connected components making $O(t)$* CROSS *queries in expectation.*

Finishing up the Borůvka style argument. One can now use the above ZeroErrorReduceCC as a subroutine to get a zero-error randomized algorithm to find a spanning forest of an undirected weighted graph which makes $O(n)$ queries in expectation. We include the algorithm and analysis towards the final (maximal) spanning forest algorithm [6].

Theorem 10 *Let $G = (V, E, w)$ be a weighted graph in which we want to find a spanning forest where $|V| = n$. V is known, E and edge weights $w \in (\mathbb{R}^+)^{|E|}$ are hidden. Algorithm 4 is a zero error algorithm that reconstructs a spanning forest of G making $O(n)$* CUT *queries in expectation.*

Proof See Appendix A. ■

4. Deterministic Spanning Forest Algorithm

We describe a polynomial time deterministic algorithm to learn a spanning forest of an undirected *unweighted* graph which makes $O(\frac{n \log n}{\log \log n})$ queries, where n is the number of vertices in G. The algorithm proceeds in two stages. In the first stage, the algorithm finds "dense connected components" in $O(\frac{n \log n}{\log \log n})$ queries with the guarantee that the number of unknown edges *across* the

6. The algorithm and analysis below are nearly identical to that in Chakrabarty and Liao (2023)

Algorithm 4 SpanningForestAlgorithm

Input: $G = (V, E)$ with $w(e) \geq 0$ for all $e \in E$.
Output: A spanning forest of G.
$n \leftarrow |V|$.
$i \leftarrow 0$.
$G_i \leftarrow (V, \emptyset)$.
for $v \in V$ **do**
  if $|E(\{v\}, V \backslash v)| > 0$ **then**
    Mark v as active.
  else
    Mark v as inactive.
end
while $i < 6 \log \log n$ **do**
  $G_{i+1} \leftarrow$ ZeroErrorReduceCC(G_i) (Algorithm 3)
  If any of old representatives in G_i is still active, mark it as the next representative in G_{i+1}. Otherwise keep marking vertices in G_{i+1} inactive until one finds an active vertex.
  $i \leftarrow i + 1$
end
$\mathcal{F} \leftarrow$ Find the spanning forest of G_i with DFSSpanningForest in Lemma 5.
return $\mathcal{F}$.

components is "small". In the second stage, it learns *all* these edges by (i) shrinking all connected components to be a "super node", (ii) using graph reconstruction algorithms to find *pseudo-edges* connecting these super-nodes, and then (iii) using ideas from coin-weighing algorithms to recover the true edges.

4.1. Find Dense Connected Components.

We first describe a deterministic algorithm which makes $O\left(\frac{n \log n}{\log \log n}\right)$ queries and returns a collection $\mathcal{C} = (C_1, \ldots, C_k)$ of connected components along with $E' \subseteq E$ which are spanning trees of each C_i. The guarantee is that the number of cross-edges $E[\mathcal{C}]$ defined as $\{(u, v) \in E : u \in C_i, v \in C_j, i \neq j\}$ is small.

We maintain $A \subseteq V$ to be a set of "active vertices" which is initialized to V. These are the set of vertices from which we haven't begun exploring.

Definition 11 *For any vertex v, let* $\mathsf{resedge}(v)$ *be the set of edges between v and the set of vertices A. Let* $\mathsf{resdeg}(v) := |\mathsf{resedge}(v)|$ *to be the number of these edges.*

We will use the following subroutines.

Lemma 12 *(Bshouty, 2009) Let $x \in \{0, 1\}^N$ be an unknown Boolean vector accessed via querying a subset $S \subseteq [N]$ and obtaining $\sum_{i \in S} x_i$ (sum-query access). If x has d ones, then there is a polynomial time deterministic algorithm* BshoutyCW *to reconstruct x which makes $O(d \log(N/d)/\log d)$ sum-queries.*

Next is the notion of an ADDITIVE query. Given a graph $G = (V, E, w)$ and subset $S \subseteq V$, the additive query ADDITIVE(S) returns the sum of weights $\sum_{e=(a,b)\in E, a\in S, b\in S} w(e)$.

Lemma 13 *(Paraphrasing Bshouty and Mazzawi, 2011b, Theorem 3) There exists a polynomial time deterministic algorithm that reconstructs a weighted hidden graph $G = (V, E, w)$ where $w : E \mapsto \mathbb{R}^+$ using $O(\frac{m \log n}{\log m} + m \log \log m)$ ADDITIVE queries.*

Proposition 14 *Any algorithm making t ADDITIVE queries can be simulated by an algorithm making $n + t$ CUT queries.*

Proof This follows by noting that $\text{ADDITIVE}(S) = \frac{1}{2} \cdot \left(\sum_{v\in V} \text{CUT}(v) - \text{CUT}(S)\right)$. That is, once $\text{CUT}(\{v\})$ is known for all the n different $v \in V$, any ADDITIVE query can be answered using a single CUT query. ■

Using proposition 14, we get

Corollary 15 *Given $m \geq n$, there exists a polynomial time deterministic algorithm* WtdGraphReconDetBM *that reconstructs a weighted hidden graph $G = (V, E, w)$ where $w : E \mapsto \mathbb{R}^+$ using $O(n + \frac{m \log n}{\log m} + m \log \log m)$ CUT queries.*

Lemma 16 *For any vertex v and any subset $A \subseteq V$, the set of edges $\text{resedge}(v)$ can be learnt using $O\left(\frac{\text{resdeg}(v) \cdot \log n}{\log \text{resdeg}(v)}\right)$ queries via the deterministic algorithm* BshoutyCW *as described in Lemma 12.*

Proof Once v and A are fixed, then one can think of learning $\text{resedge}(v)$ as figuring out the following unknown vector $x \in \{0, 1\}^{|A|}$ which has a coordinate for every vertex in A. The uth coordinate is 1 if $(v, u) \in E$ and 0 otherwise. Note that querying $\sum_{u\in S} x_u$ for $S \subseteq A$ is precisely a CROSS query. The lemma follows from Lemma 12. ■

Our algorithm proceeds by doing a breadth first search from active vertices only trying to connect to other active vertices only if $\text{resdeg}(v) \geq L = \Theta(\log n/(\log \log n)^2)$. If no such vertex exists, then we treat the component discovered so far as one component. The full details are described in Algorithm 5.

Lemma 17 *Algorithm 5 makes $O(\frac{n \log n}{\log \log n})$ CUT queries and returns $\mathcal{C}$ with $|E[\mathcal{C}]| = O\left(\frac{n \log n}{(\log \log n)^2}\right)$.*

Proof Queries are only made in Line 11 and in Line 13 of Algorithm 5. Amortized, the number of queries made in Line 11 is at most n since we do it whenever the queue is emptied and a vertex never re-enters Q because it enters the queue as an inactive vertex. The queries of Line 13 are made only when $\text{resdeg}(v) \geq L$, and so the number of queries is $O\left(\text{resdeg}(v) \cdot \frac{\log n}{\log L}\right)$. We can charge $O\left(\frac{\log n}{\log L}\right) = O\left(\frac{\log n}{\log \log n}\right)$ to every vertex w that is being added to C (alternately being deemed inactive). Since a vertex is added to C at most once, amortized the total number of such queries is at most $O(\frac{n \log n}{\log \log n})$.

Orient all the edges (u, v) of $E[\mathcal{C}]$ for $u \in C_i$ and $v \in C_j$ from u to v if $i < j$. That is, if u is marked *inactive* earlier than v. Note that the out-degree of every such vertex is $< L$ because when it was made inactive, $\text{resdeg}(v) < L$, and the final out-degree can only go down (maybe some active vertex was added to C as BFS progressed). Since $|E[\mathcal{C}]|$ is precisely the sum of these out-degrees, the lemma follows.

■

Algorithm 5 ConnectedComponentDiscovery

Input: A graph $G = (V, E)$ with CUT query access.

Output: Return $\mathcal{C} = (C_1, \ldots, C_k)$, $E' \subseteq E$. Total number of queries: $O\left(\frac{n \log n}{\log \log n}\right)$.

/* Guarantees: (i) Each C_i is connected using edges of E, (ii) number of cross-edges $|E[\mathcal{C}]|$ is $O\left(\frac{n \log n}{(\log \log n)^2}\right)$ */

$E' \leftarrow \emptyset$; $A \leftarrow V$.

$L \leftarrow \frac{\log n}{(\log \log n)^2}$.

$\mathcal{C} \leftarrow \emptyset$. // Set of connected components

while $A \neq \emptyset$ **do**

  Let $x \in A$ be an arbitrary active vertex.

  Start new connected component $C \leftarrow \{x\}$ and $A \leftarrow A \setminus \{x\}$.

  Q.add(x) // Start BFS from x; Q is a queue

  while $Q \neq \emptyset$ **do**

    $v \leftarrow Q$.remove().

    $C \leftarrow C \cup \{v\}$. // Add v to the connected component.

    Query resdeg(v).

    if resdeg(v) $\geq L$ **then**

      Use algorithm BshoutyCW to learn resedge(v).

      $E' = E' \cup$ resedge(v).

      For all w such that $(v, w) \in E'$: Q.add(w) and $A \leftarrow A \setminus \{w\}$.

      /* Add w to Q and mark it inactive */

  end

  Add C to $\mathcal{C}$. // C is connected; all vertices $v \in C$ have resdeg(v) $< L$.

end

return $\mathcal{C}$

4.2. Recover True Edges between Components

Consider the unknown undirected *weighted*/multi graph $H = (\mathcal{C}, F)$ whose vertices are the components returned by Algorithm 5 and we form F by adding the pair (C_i, C_j) for every $v_i \in C_i$ and $v_j \in C_j$ such that $(v_i, v_j) \in E(G)$. We call the edges in F as "pseudo-edges". We begin with a simple observation.

Lemma 18 *A* CUT *query in H can be simulated by a* CUT *query in G. H is connected if and only if G is connected.*

Proof Given a subset $\mathcal{T} \subseteq \mathcal{C}$, note that $\partial_H(T)$ is precisely $\partial_G(\bigcup_{C \in \mathcal{T}} C)$: the weight/number of parallel copies of (C_i, C_j) for $C_i \in \mathcal{T}$ and $C_j \notin \mathcal{T}$ is precisely the number of edges of the form (v_i, v_j) where $v_i \in C_i$ and $v_j \in C_j$. Thus, if H is disconnected, then G is disconnected. If H is connected, then G is connected because every C_i is connected in G. ■

Using the fact that H can be reconstructed using $O(\frac{m_H \log n}{\log m_H} + m_H \log \log m_H)$ deterministic CUT queries via Corollary 15 where m_H is the total number of edges (even counting without multiplicity), and using the fact that $m_H \leq nL$, we get that H can be reconstructed in $O(\frac{nL \log n}{\log(nL)} +$

$nL \log\log(nL)) = O(nL + nL \log\log n) = O(\frac{n \log n}{\log\log n})$ queries. Thus, if we only cared to know if G is connected or not, we would have our $O(n \log n / \log\log n)$ query algorithm by just checking if the reconstructed H is connected or not.

However, the spanning forest of H contains pseudo-edges of the form (C_i, C_j), but we need to return true edges of G. We now describe the process to do the same. We first begin with a simple observation.

Lemma 19 *Suppose (C_i, C_j) is a pseudo-edge in F. Then, a true edge (v_i, v_j) with $v_i \in C_i$ and $v_j \in C_j$ can be found in $O(\log |C_i| + \log |C_j|)$ CUT queries using a binary-search style routine* $\mathsf{BinSearch}$.

Proof Indeed, for this we don't need the full power of CUT queries, but just knowing whether a cut is empty or not suffices. We take an arbitrary half of $A \subseteq C_i$ and perform a $\mathsf{CROSS}(A, C_j)$ query. If it is > 0, then we recurse on (A, C_j) and if it is $= 0$ we recurse on $(C_i \setminus A, C_j)$. In $\log |C_i|$ such queries, we discover $v_i \in C_i$ which has at least one edge to some vertex in C_j. We then repeat the same process by taking an arbitrary half $B \subseteq C_j$ but not performing $\mathsf{CROSS}(\{v_i\}, B)$. In $O(\log |C_j|)$ more queries we would recover the desired edge (v_i, v_j). ■

Now fix a spanning forest $\mathcal{F}$ of H. For each $(C_i, C_j) \in \mathcal{F}$ we want to recover a true edge (v_i, v_j) with $v_i \in C_i$ and $v_j \in C_j$. If the degree of every vertex $C_i \in \mathcal{F}$ was $\leq D$, then we could simply use the algorithm in Lemma 19 to obtain all such true edges with number of queries equaling

$$\sum_{(C_i, C_j) \in \mathcal{F}} O(\log |C_i| + \log |C_j|) \leq D \cdot \sum_{C_i \in \mathcal{F}} \log |C_i| \underbrace{\leq}_{\text{AM-GM}} O(kD \log(n/k)) \leq O(nD)$$

where $|\mathcal{C}| = k$. So, if $D \leq \frac{\log n}{\log\log n}$, we would be done. However, D could indeed be as large as $\Theta(n)$. To take care of this, we need to handle the "high-degree" vertices of $\mathcal{F}$ separately, again using the fact that CUT queries give us more power. Here is the observation. Let C_i be a vertex with degree $D \geq \frac{\log n}{\log\log n}$ in $\mathcal{F}$. Let $C_1, \ldots, C_D$ be its neighbors. Now consider the bipartite graph H' whose vertices are the vertices in C_i in one part and the vertices $\{C_1, C_2, \ldots, C_j\}$ in the other part, with an edge between $v_i \in C_i$ and C_j iff there exists $v_j \in C_j$ such that $(v_i, v_j) \in E(G)$. We now use $\mathsf{WtdGraphReconDetBM}$ to learn all the edges in H' deterministically. Thus, for all the pseudo-edges (C_i, C_j) incident on C_i, we discover *one* endpoint (that lying in C_i) of $\geq D$ many true edges. This is how we take care of high-degree vertices. The full algorithm details are given in Algorithm 6.

Theorem 20 *Algorithm 5 and Algorithm 6 is a deterministic algorithm that returns a maximal spanning forest of G in $O\left(\frac{n \log n}{\log\log n}\right)$ queries.*

Proof We have already analyzed the query complexity of Algorithm 5 in Lemma 17. Lemma 18 implies that the number of edges in H is $O\left(\frac{n \log n}{(\log\log n)^2}\right)$. The number of queries made in Line 1 of Algorithm 6 is therefore, as argued in the beginning of this subsection, $O(\frac{n \log n}{\log\log n})$. The number of queries made in Line 6 of Algorithm 6 is $O(\frac{m_i \log n_i}{\log m_i} + m_i \log\log m_i)$ where m_i is the number of edges in the bipartite graph H_i and n_i is the number of vertices. The second term, summed over all calls, and using the fact that $\sum_i m_i = O(nL)$, would together be at most $O(n \log n / \log\log n)$.

Algorithm 6 JoinConnectedComponents

Input: Unknown undirected graph $G = (V, E)$ with CUT query access; $\mathcal{C} = (C_1, \ldots, C_k)$; $E' \subseteq E$ such that each C_i is connected using edges in E'.

Output: Return spanning forest of G Total number of queries: $O\left(\frac{n \log n}{\log \log n}\right)$.

Learn the pseudo-graph $H = (\mathcal{C}, F)$ using WtdGraphReconDetBM in Corollary 15.
$\mathcal{F}$ be arbitrary spanning forest of H.
$E'' \leftarrow \emptyset$.
for *$C_i \in \mathcal{F}$ with degree $D > \frac{\log n}{\log \log n}$* **do**
    Let $C_1, \ldots, C_D$ be neighbors of C_i in $\mathcal{F}$.
    Use WtdGraphReconDetBM to learn bipartite graph $H_i := (C_i, \{C_1, \ldots, C_D\})$.
    Replace $\{C_i\}$ in $\mathcal{F}$ with $\{v : v \in C_i\}$ and edges $E' \cap E[C_i]$.
    Add minimal collection of edges from H_i to ensure $C_i \cup \{C_1, \ldots, C_D\}$ is connected, to E''.
end
/* Now all C_i's in $\mathcal{F}$ have degree at most $\frac{\log n}{\log \log n}$. */
for *all remaining (C_i, C_j) or (C_i, v) pseudo-edges of $\mathcal{F}$* **do**
    Use BinSearch to learn true edge (v_i, v_j) or (v_i, v) with $v_i \in C_i$ and $v_j \in C_j$, and add to E''.
end
return $E' \cup E''$.

The first term, if $m_i \geq n_i$ would give $O(1)$ per edge, which again amortized would be $O(nL)$. If $m_i \ll n_i$, then using $m_i \geq \log n / \log \log n$, we would still pay $O(\frac{\log n}{\log \log n})$ per true edge added to E'' at that round. Since the total number of true edges in E'' is at most $n - 1$, we would in all pay $O(n \log n / \log \log n)$. ■

5. Conclusion

We give an optimal Las Vegas algorithm style that recovers a maximal spanning forest of an unknown, weighted, undirected graph using $O(n)$ queries in expectation. The algorithm uses Apers et al. (2022)'s framework, and extends the $O(n)$ zero-error algorithm in unweighted graph to weighted case. It also closes the gap between the lower bound result of $\Omega(n)$ in Auza and Lee (2021) and previous upper bound of $O(n \log n)$. A key ingredient of the improvement is that we circumvent the degree estimation, which is possible not an $O(1)$-query operation in weighted graphs given the results of Chakraborty et al. (2022). We do this by removing the high degree representatives step by step using random sampling with a sample rate increasing from close to zero to one.

For unweighted graphs, we give an $O(n \log n / \log \log n)$-query poly-time deterministic algorithm slightly improving the state-of-the-art. Our algorithm doesn't work with weights yet again due to the need for knowing degree (Line 11 in Algorithm 5), and we don't know how to bypass it without randomization. However, the more interesting question is to obtain $O(n)$ query deterministic algorithms even for unweighted graphs, or prove its impossibility.

Acknowledgments

Both authors were partially supported by NSF grant CCF-2041920.

References

Raghavendra Addanki, Andrew McGregor, and Cameron Musco. Non-adaptive edge counting and sampling via bipartite independent set queries. In *Proc., European Symposium on Algorithms*, 2022.

Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 459–467, 2012.

Martin Aigner. *Combinatorial search*. John Wiley & Sons, Inc., 1988.

Noga Alon and Vera Asodi. Learning a hidden subgraph. *SIAM Journal on Discrete Mathematics (SIDMA)*, 18(4):697–712, 2005.

Noga Alon, Richard Beigel, Simon Kasif, Steven Rudich, and Benny Sudakov. Learning a hidden matching. *SIAM Journal on Computing (SICOMP)*, 33(2):487–501, 2004.

Alexandr Andoni, Zhao Song, Clifford Stein, Zhengyu Wang, and Peilin Zhong. Parallel graph connectivity in log diameter rounds. In *Proc., IEEE Conference on the Foundations of Computer Science (FOCS)*, pages 674–685, 2018.

Dana Angluin and Jiang Chen. Learning a hidden graph using $O(\log n)$ queries per edge. *J. Comput. System Sci.*, 74(4):546–556, 2008.

Simon Apers, Yuval Efron, Pawel Gawrychowski, Troy Lee, Sagnik Mukhopadhyay, and Danupon Nanongkai. Cut query algorithms with star contraction. *Proc., IEEE Conference on the Foundations of Computer Science (FOCS)*, 2022.

Sepehr Assadi, Deeparnab Chakrabarty, and Sanjeev Khanna. Graph connectivity and single element recovery via linear and or queries. In *Proc., European Symposium on Algorithms*, 2021.

Arinta Auza and Troy Lee. On the query complexity of connectivity with global queries. *arXiv preprint arXiv:2109.02115*, 2021.

Paul Beame, Sariel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. Edge estimation with independent set oracles. *ACM Trans. on Algorithms (TALG)*, 16(4):1–27, 2020. Preliminary version in Proc. ITCS, 2018.

Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Lacki, and Vahab Mirrokni. Near-optimal massively parallel graph connectivity. In *Proc., IEEE Conference on the Foundations of Computer Science (FOCS)*, pages 1615–1636, 2019.

Nader H Bshouty. Optimal algorithms for the coin weighing problem with a spring scale. In *Proc., Conf. on Learning Theory (COLT)*, 2009.

Nader H Bshouty and Hanna Mazzawi. On parity check (0, 1)-matrix over $\mathbb{Z}_p$. In *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1383–1394, 2011a.

Nader H. Bshouty and Hanna Mazzawi. Algorithms for the coin weighing problems with the presence of noise. *Electron. Colloquium Comput. Complex.*, page 124, 2011b. URL https://eccc.weizmann.ac.il/report/2011/124.

Nader H. Bshouty and Hanna Mazzawi. Toward a deterministic polynomial time algorithm with optimal additive query complexity. *Theoretical Computer Science*, 417:23–35, 2012.

Deeparnab Chakrabarty and Hang Liao. A query algorithm for learning a spanning forest in weighted undirected graphs. In *Proc., International Conference on Algorithmic Learning Theory (ALT)*, pages 259–274, 2023.

Diptarka Chakraborty, Gunjan Kumar, and Kuldeep S. Meel. Support size estimation: The power of conditioning. *arXiv, 2211.11967*, 2022.

Chandra Chekuri and Kent Quanrud. Isolating cuts, (bi-)submodularity, and faster algorithms for connectivity. In *Proc., International Conference on Algorithms, Logic, and Programming (ICALP)*, 2021.

Xi Chen, Amit Levi, and Erik Waingarten. Nearly optimal edge estimation with independent set queries. In *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2916–2935, 2020.

Sung-Soon Choi. Polynomial time optimal query algorithms for finding graphs with arbitrary real weights. In Shai Shalev-Shwartz and Ingo Steinwart, editors, *Proc., Conf. on Learning Theory (COLT)*, volume 30, pages 797–818, 2013.

Sung-Soon Choi and Jeong Han Kim. Optimal query complexity bounds for finding graphs. *Artif. Intell.*, 174(9-10):551–569, 2010.

Robert Dorfman. The detection of defective members of large populations. *The Annals of Mathematical Statistics*, 14(4):436–440, 1943.

Dingzhu Du and Frank K Hwang. *Combinatorial group testing and its applications*, volume 12. World Scientific, 2000.

Ran Duan and Le Zhang. Faster randomized worst-case update time for dynamic subgraph connectivity. In *Proc., Workshop on Algorithms and Data Structures (WADS)*, pages 337–348, 2017.

Andrei Graur, Tristan Pollner, Vidhya Ramaswamy, and S Matthew Weinberg. New query lower bounds for submodular function minimization. *Proc., Innovations in Theoretical Computer Science (ITCS)*, page 64, 2020.

Vladimir Grebinski and Gregory Kucherov. Reconstructing a hamiltonian cycle by querying the graph: Application to dna physical mapping. *Discrete Applied Mathematics*, 88(1-3):147–165, 1998.

Vladimir Grebinski and Gregory Kucherov. Optimal reconstruction of graphs under the additive model. *Algorithmica*, 28(1):104–124, 2000.

András Hajnal, Wolfgang Maass, and György Turán. On the communication complexity of graph properties. In *Proc., ACM Symposium on the Theory of Computing (STOC)*, pages 186–191, 1988.

Nicholas James Alexander Harvey. *Matchings, matroids and submodular functions*. PhD thesis, Massachusetts Institute of Technology, 2008.

Bruce M Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1131–1142. SIAM, 2013.

Troy Lee, Miklos Santha, and Shengyu Zhang. Quantum algorithms for graph problems with cut queries. In *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 939–958, 2021.

Hanna Mazzawi. Optimally reconstructing weighted graphs using queries. In *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 608–615, 2010.

Sagnik Mukhopadhyay and Danupon Nanongkai. Weighted min-cut: sequential, cut-query, and streaming algorithms. In *Proc., ACM Symposium on the Theory of Computing (STOC)*, pages 496–509, 2020.

Jelani Nelson and Huacheng Yu. Optimal lower bounds for distributed and streaming spanning forest computation. In *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1844–1860, 2019.

Maurice Queyranne. Minimizing symmetric submodular functions. *Mathematical Programming*, 82:3–12, 1998.

Ran Raz and Boris Spieker. On the "log rank"-conjecture in communication complexity. *Combinatorica*, 15(4):567–588, 1995.

Lev Reyzin and Nikhil Srivastava. Learning and verifying graphs using queries with a focus on edge counting. In *Proc., International Conference on Algorithmic Learning Theory (ALT)*, pages 285–297. Springer, 2007.

Aviad Rubinstein, Tselil Schramm, and S. Matthew Weinberg. Computing exact minimum cuts without knowing the graph. In *Proc., Innovations in Theoretical Computer Science (ITCS)*, pages 39:1–39:16, 2018.

Appendix A. Missing Proofs

Proposition 21 *(from Chakrabarty and Liao, 2023, Claim 3) Let $t_1, t_2, \cdots, t_j$ be integers at least 2. If $\sum_{i=1}^{j} t_i = t$ where $\frac{n}{\log n} \leq t \leq n$ and $j \leq \log n$, then $\sum_{i=1}^{j} \frac{t_i}{\log t_i} \leq \frac{4t}{\log t} = O(\frac{t}{\log t})$.*

Proof Partition $\{t_i\}$ into sets $P_1 = \{t_i | t_i \in [2, t/\log^2 t)\}, P_2 = \{t_i | t_i \in [t/\log^2 t, t]\}$. For $t_i \in P_1$, note $|P_1| \leq \log n \leq 2\log t$, it follows that

$$\sum_{i \in P_1} \frac{i}{\log i} \leq \sum_{i \in P_1} i \leq 2\log t \cdot \frac{t}{\log^2 t} = \frac{2t}{\log t}.$$

For $t_i \in P_2$, since $t_i > \sqrt{t}, \log t_i \geq \frac{1}{2}\log t$. So $t_i/\log t_i \leq t_i/\frac{1}{2}\log t = \frac{2t_i}{\log t}$. For those t_i,

$$\sum_{i \in P_2} \frac{i}{\log i} \leq \sum_{i \in P_2} \frac{2i}{\log t} \leq \frac{2t}{\log t}.$$

Combine the two inequalities,

$$\sum_{i \in P} \frac{i}{\log i} = \sum_{i \in P_1} \frac{i}{\log i} + \sum_{i \in P_2} \frac{i}{\log i} \leq \frac{4t}{\log t} = O(\frac{t}{\log t}).$$

■

Theorem 22 *(Theorem 4.A in the main body) (Follows from Bshouty and Mazzawi, 2011a, Corollary 4). There exists an adaptive, deterministic algorithm* GR1 *which takes input a bipartite graph $G = (U, V)$ on n vertices and m edges such that U and V are known but m is unknown to the algorithm, and reconstructs the edges of G along with their weights, making $\frac{C_{\mathsf{GR1}} m \log n}{\log m}$* CROSS *queries, for some absolute constant $C_{\mathsf{GR1}} > 0$.*

Theorem 23 *(Theorem 4.B in the main body) (Follows from Choi, 2013, Theorem 1). There exists an adaptive, randomized algorithm* GR2 *which takes input a bipartite graph $G = (U, V)$ on n vertices and m edges such that U and V are known but m is unknown to the algorithm, and either reconstructs the edges of G along with their weights, or aborts. The probability that the algorithm aborts or makes more than $\frac{C_{\mathsf{GR2}} m \log n}{\log m}$* CROSS *queries, for some absolute constant $C_{\mathsf{GR2}} > 0$, is at most $O(\frac{\log m}{m})$.*

To prove Theorem 4.A and Theorem 4.B, we use the following results:

Lemma 24 *(Bshouty and Mazzawi, 2011a, Corollary 4). There exists a non-adaptive determinstic algorithm* GraphReconstructionBM *that uses $O(\frac{m \log n}{\log m})$* ADDITIVE *queries and reconstruct any weighted hidden graph with at most m edges.*

Note that the above algorithm uses additive queries. An additive query on $S \subseteq V$ returns the sum of edge weights in the induced subgraph with vertex set S, and in general is stronger than CROSS queries[7]. Observe that if $G = (R, B)$ is a bipartite graph, an additive query on S is equivalent to a CROSS query CROSS$(S \cap R, S \cap B)$. We will only be reconstructing bipartite graphs, and therefore, we only need the following corollary which is a restatement of the above lemma.

7. While 3 additive queries can simulate a CROSS query, it was shown in Lee et al. (2021) that $\Omega(n)$ CROSS queries may be needed to simulate an additive query.

Corollary 25 *There exists a non-adaptive, deterministic algorithm* GraphReconstructionBM *(*GRBM *in short) that uses* $O(\frac{m \log n}{\log m})$ CROSS *queries and reconstruct any weighted hidden* ***bipartite*** *graph with at most* m *edges.*

Bshouty and Mazzawi (2011a) only proves the existence of a query algorithm but neither gives an explicit construction, nor does it give a polynomial time recovery algorithm. Indeed, it is an outstanding open question to obtain a non-adaptive and/or deterministic algorithm with this query complexity. The polynomial time complexity was resolved in Choi (2013) via an adaptive, randomized Monte Carlo[8] algorithm.

Lemma 26 *(Paraphrasing Choi, 2013, Theorem 1). One can construct a randomized polynomial time adaptive algorithm* GraphReconstructionChoi *(*GRC *in short) which given an unknown weighted graph* G *on* n *vertices with at most* m *edges, makes* $O\left(\frac{m \log n}{\log m}\right)$ CROSS *queries, and with probability* $1 - O(\frac{\log m}{m})$ *returns the edges of* G *and their weights.*

As stated, both results (Theorem 4.A and Theorem 4.B) assume knowledge of this parameter m which is an upper bound on the number of edges. For our purposes, the number of edges would be unknown (in fact, random variables), and so we use a simple modification via a doubling-trick to get an efficient algorithm that recovers the hidden edges of a graph using CROSS queries that doesn't require to know the number of the hidden edges to be known. Now we are ready to state the proof.

Proof We run WtdGraphReconChoi in Lemma 26 with number of edge guesses $2^1, 2^2, \cdots$ growing in powers of 2. Let C_{GRC} be the hidden constant in the query complexity of Lemma 26. After guessing 2^i, WtdGraphReconChoi terminates with $C_{\mathsf{GRC}} \cdot \frac{2^i \log n}{\log 2^i}$ queries. If WtdGraphReconChoi returns recovered edges, we check

1. If the edges indeed exist in G.
2. If all the edges are recovered.

For a guess 2^i, checking the recovered edges are indeed in G takes at most 2^i queries. Checking if all edges are recovered takes 1 CROSS query because G is bipartite and it takes 1 query to get the sum of all edge weights. One then check if the sum of the recovered edge weights equal to the sum of actual edge weights.

When the guess $2^i \geq m$, running WtdGraphReconChoi with guess 2^i fails with probability $O(\frac{\log m}{m})$. Hence the probability that the algorithm stops at guess 2^{i^*} where i^* is the smallest i such that $2^i \geq m$ is $1 - O(\frac{\log m}{m})$. Conditioned on the algorithm stops on the guess 2^{i^*}, we show that the total number of queries is $\sum_{i=1}^{i^*} C_{\mathsf{GRC}} \cdot \frac{2^i \log n}{\log 2^i} \leq 16 C_{\mathsf{GRC}} \cdot \frac{m \log n}{\log m}$.

8. The error probability in Theorem 1 of Choi (2013) is only stated as $1 - o(1)$, however, in the statement of Theorem 3 which uses Theorem 1, it is mentioned as $O(\log m/m)$.

$$\sum_{i=1}^{i^*}\left(C_{\mathsf{GRC}}\cdot\frac{2^i\log n}{\log 2^i}\right)\le 4C_{\mathsf{GRC}}\cdot\left(\frac{2^{i^*+1}\log n}{\log 2^{i^*+1}}\right)\quad\text{(by Proposition 21)}$$
$$\le 16C_{\mathsf{GRC}}\cdot\left(\frac{2^{i^*-1}\log n}{\log 2^{i^*+1}}\right)$$
$$\le 16C_{\mathsf{GRC}}\cdot\left(\frac{m\log n}{\log m}\right)$$

Set C_{GR2} to be $16C_{\mathsf{GRC}}$ and we are done.
The proof of Theorem 4.A is similar to the proof of Theorem 4.B. ■

Lemma 27 *Let $G=(V,E)$ be a weighted graph with non-negative weights. Suppose G_2 is a subgraph of G with t connected components for some $t\ge\frac{n}{\log n}$. Algorithm 2 makes $O(t)$ queries, and with probability $\ge 1/10$ returns a graph with $\le\frac{7}{8}t$ connected components, or returns* ABORT.

Proof Let $C_1,\cdots,C_t$ be the connected components of G_2. Algorithm 2 begins by coloring the components red or blue uniformly at random. If a component is colored red/blue, we color all its vertices with the same color. Let B be the set of vertices colored blue. Call a representative r of a component "good" if it is colored red and $d(r)\ge 1$. Note that a representative is good with probability $\ge 1/4$; its component has to be colored red and one of its neighbor's component has to be colored blue. Since it has at least one neighbor outside its component (since it's a representative), the probability follows. Hence $\mathbf{E}[|R|]\ge t/4$ and $\mathbf{Pr}[|R|\le\frac{t}{8}]\le\frac{1}{2}$. Therefore, the probability Algorithm 2 aborts in Line 6 is $\le\frac{1}{2}$.

Fix an iteration i. Let X_i be the number of edges in $E(\mathcal{R}_i,B_i)$ which is a random variable. The induced subgraph $(\mathcal{R}_i,B_i)$ has at most n vertices. By Theorem 4.A or Theorem 4.B, it takes at most $\max(C_{\mathsf{GR1}},C_{\mathsf{GR2}})\cdot\left(\frac{X_i\log n}{\log X_i}\right)$ queries to recover the edges in $E(\mathcal{R}_i,B_i)$. More precisely, if we run the deterministic algorithm from Theorem 4.A, we recover these edges with probability 1. If we run the efficient randomized algorithm from Theorem 4.B, we recover these edges with probability $1-O(\log m/m)$.

By Lemma 7, $\mathbf{E}[\sum_{0\le i\le\lceil\log t\rceil}X_i]=\sum_e\mathbf{Pr}[e\in\cup_{i\in[\log t]}E_i]\le\sum_{r\in\mathcal{R}_0}\frac{5}{d(r)}\cdot d(r)=5|\mathcal{R}_0|\le 5t$. By Markov's Inequality, $\mathbf{Pr}[\sum_{0\le i\le\lceil\log t\rceil}X_i\ge 15t]\le\frac{1}{3}$. Conditioned on the event $\sum_{0\le i\le\lceil\log t\rceil}X_i\le 15t$, it takes

$$\sum_{i=0}^{\lceil\log t\rceil}\max(C_{\mathsf{GR1}},C_{\mathsf{GR2}})\left(\frac{X_i\log n}{\log X_i}\right)\le\max(C_{\mathsf{GR1}},C_{\mathsf{GR2}})\log n(4\cdot\frac{15t}{\log t})\quad\text{(by Proposition 21)}$$
$$\le 120\max(C_{\mathsf{GR1}},C_{\mathsf{GR2}})t$$

many queries to recover all edges with probability $\ge 1-O(\log t\log m/m)$. Note we have multiplied by $\log t$ to account for the union bound over the $\log t$ different reconstructions.

Since the budget is initialized to be $120\max(C_{\mathsf{GR1}},C_{\mathsf{GR2}})t$, we can recover all the edges when $\sum_i X_i\le 15t$ with failure probability $O(\frac{\log t\log m}{m})\le\frac{1}{100}$, for large enough m. Contrapositively, if we abort in Line 15, either $\sum_i X_i>15t$ (which is $\le 1/3$) or due to the failure probabilities of

the $\log t$ different invocations of GR2 (which is $\leq 1/100$). The overall ABORT probability can be union bounded by

$$\mathbf{Pr}[\text{ABORT in Line 6}] + \mathbf{Pr}[\text{ABORT due to } \sum_i X_i \geq 15t] +$$
$$\mathbf{Pr}[\text{ABORT due to failures of GR2}] \leq \frac{1}{2} + \frac{1}{3} + \frac{1}{100} < \frac{9}{10}.$$

If Algorithm 2 doesn't abort, $\mathcal{R}_i = \emptyset$ at the end since every representative in R are connected to some component in B after edge sampling. Since $|R| \geq \frac{t}{8}$, the number of connected components shrinks by $\geq t/8$, and therefore $G_2 + \bigsqcup_{i\in[\lceil \log t \rceil]} E_i$ has $\leq \frac{7}{8}t$ connected components.

We still need to show the number of queries is $O(t)$. We can check if a representative has a blue neighbor using 1 CROSS query and there can be at most t representatives. In the sampling/recovery stage, the total number of queries is bounded by the initialized value of budget which is $120 \max(C_{\mathsf{GR1}}, C_{\mathsf{GR2}})t$. Hence the algorithm uses at most $O(t)$ queries. ■

Theorem 10 *Let $G = (V, E, w)$ be a weighted graph in which we want to find a spanning forest where $|V| = n$. V is known, E and edge weights $w \in (\mathbb{R}^+)^{|E|}$ are hidden. Algorithm 4 is a zero error algorithm that reconstructs a spanning forest of G making $O(n)$ CUT queries in expectation.*

Proof

The queries used in Algorithm 4 are from: (1) calling Algorithm 3 in Line 12, (2) finding active vertices in line 6 - 10 and 13, and (3) finding the spanning forest with subgraph G_i in Line 15.

Line 12: By Corollary 9, the total expected number of queries that used in calling Algorithm 3 is at most $\sum_{i=0}^{6\log\log n} \left(\frac{7}{8}\right)^i n = O(n)$.

Line 6 - 10 and 13: To learn whether a vertex is active takes 1 query. Every time Algorithm 4 invokes Algorithm 3, in each connected component, we find one "active" vertex (if there is any) by iterating over vertices that are not "inactive" at the end. Once a vertex is "inactive", it becomes "inactive" forever. The only vertices that we may query more than once are the representatives, in which case in the ith call of Algorithm 3 there are at most $(7/8)^i n$ of them. So the total number of queries used to find representatives is at most $n + \sum_{i=0}^{6\log\log n}(\frac{7}{8})^i n = O(n)$.

Line 15: After $6\log\log n$ calls of Algorithm 3, the number of connected components of $G_{6\log\log n}$ is at most $(\frac{7}{8})^{6\log\log n - 1} \leq \frac{1}{\log n}$. We apply Lemma 5 to find the remaining tree edges deterministically which takes $O(n)$ CROSS queries. Summing up the queries from the three parts, we conclude the total expected number of queries is $O(n)$. ■