Graph Connectivity and Single Element Recovery via Linear and OR Queries

Sepehr Assadi
Rutgers University, New Brunswick, NJ, USA

Deeparnab Chakrabarty
Dartmouth College, Hanover, NH, USA

Sanjeev Khanna
University of Pennsylvania, Philadelphia, PA, USA

Abstract

We study the problem of finding a spanning forest in an undirected, n-vertex multi-graph under two basic query models. One are Linear queries which are linear measurements on the incidence vector induced by the edges; the other are the weaker OR queries which only reveal whether a given subset of plausible edges is empty or not. At the heart of our study lies a fundamental problem which we call the *single element recovery* problem: given a non-negative vector $x \in \mathbb{R}^N_{\geq 0}$, the objective is to return a single element $x_j > 0$ from the support. Queries can be made in rounds, and our goals is to understand the trade-offs between the query complexity and the rounds of adaptivity needed to solve these problems, for both deterministic and randomized algorithms. These questions have connections and ramifications to multiple areas such as sketching, streaming, graph reconstruction, and compressed sensing. Our main results are as follows:

- For the single element recovery problem, it is easy to obtain a deterministic, r-round algorithm which makes $(N^{1/r}-1)$-queries per-round. We prove that this is tight: any r-round *deterministic* algorithm must make $\geq (N^{1/r}-1)$ Linear queries in some round. In contrast, a 1-round $O(\text{polylog}(N))$-query *randomized* algorithm is known to exist.
- We design a *deterministic* $O(r)$-round, $\tilde{O}(n^{1+1/r})$-OR query algorithm for graph connectivity. We complement this with an $\tilde{\Omega}(n^{1+1/r})$-lower bound for any r-round deterministic algorithm in the OR-model.
- We design a *randomized*, 2-round algorithm for the graph connectivity problem which makes $\widetilde{O}(n)$-OR queries. In contrast, we prove that any 1-round algorithm (possibly randomized) requires $\widetilde{\Omega}(n^2)$-OR queries. A *randomized*, 1-round algorithm making $\widetilde{O}(n)$-Linear queries is already known.

All our algorithms, in fact, work with more natural graph query models which are *special* cases of the above, and have been extensively studied in the literature. These are Cross queries (cut-queries) and BIS (bipartite independent set) queries.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Query Models, Graph Connectivity, Group Testing, Duality

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.7

Related Version *Full Version*: https://arxiv.org/abs/2007.06098

Funding *Sepehr Assadi*: Supported in part by NSF CAREER award CCF-2047061, and gift from Google Research.
Deeparnab Chakrabarty: Supported in part by the NSF awards CCF-1813053 and CCF-2041920.
Sanjeev Khanna: Supported in part by the NSF awards CCF-1763514, CCF-1617851, CCF-1934876, and CCF-2008305.

1 Introduction

Many modern applications compel algorithm designers to rethink random access to input data, and revisit basic questions in a *query access model* where the input is accessed only via answers to certain kinds of queries. There are many reasons for this ranging from data volume (only snapshots of the data can be accessed) to data ownership (access is restricted via certain APIs).

In this paper, we study algorithms accessing an unknown, undirected multi-graph G on n vertices in the following two basic query models. Think of the graph as an unknown non-negative $\binom{n}{2}$ dimension vector x_G with $\text{supp}(x_G)$ denoting the positive coordinates. With this view, answers to these queries below can be interpreted as *measurements* on this vector.

- **Linear Queries** (Linear): Given any non-negative[1] $\binom{n}{2}$ dimension vector a_G, what is $a_G \cdot x_G$?
- **OR Queries** (OR): Given any subset S of the $\binom{n}{2}$ dimensions, is $\text{supp}(x_G) \cap S$ empty?

Reverting back to the combinatorial nature of graphs, it is perhaps more natural to think of different kinds of queries, and indeed the following two have been extensively studied. These are however special[2] cases, respectively, of the queries above.

- **Cross-additive Queries** (Cross): Given two disjoint subsets A, B of V, $\mathsf{Cross}(A, B)$ returns the number of edges, including multiplicity, that have one endpoint in A and the other in B.
- **Bipartite Independent Set Queries** (BIS): Given two disjoint subsets A, B of V, $\mathsf{BIS}(A, B)$ returns whether or not there is an edge that has one endpoint in A and the other in B.

The above query models (and similar variants such as additive queries [30], cut-queries [49], edge-detection queries [6, 9]) have a rich literature [1, 4, 9, 16, 17, 30, 41, 44, 49]. Most previous works, however, have focused on either *graph reconstruction* [15–17, 41], or on *parameter estimation* (e.g., estimating the number of edges [9] or triangles [11]). In this work, however, our goal is to understand the power and limitations of these queries to reveal ***structural properties*** of the underlying graph. In particular, we study the following basic property.

▶ **Problem 1** (Graph connectivity). *Given query access to an undirected multigraph on the vertex set $V = \{1, \ldots, n\}$, return a* spanning forest.

It is not too hard to implement the *classic* BFS or DFS traversals to obtain an $\tilde{O}(n)$-query deterministic algorithm for the above problem in either query model. However, such algorithms are *adaptive*, that is, the queries depend on the answers obtained so far. A much more modern algorithm of Ahn, Guha, and McGregor [2] gives[3] an $\tilde{O}(n)$-Linear query *non-adaptive* but *randomized* algorithm for the problem. This raises the following questions that motivate us

> *What is the rounds-of-adaptivity versus query-complexity trade-off for deterministic algorithms for Problem 1? Can randomization also help in the OR and BIS models?*

It turns out that understanding the complexity of Problem 1 is closely related to understanding an even more basic problem which we discuss below.

---

[1] Non-negativity is for convenience. A general linear query can be broken into two non-negative queries.

[2] The Cross (and BIS queries) correspond to $\{0, 1\}$ vectors a_G (and subsets) corresponding to cuts. Indeed, our algorithms work with the weaker queries while our lower bounds will be for the stronger queries. It should also be clear that the Linear (and respectively Cross) queries are at least as strong as OR (resp, BIS) queries.

[3] Using results in [51], one can also obtain a $\tilde{O}(n)$-query deterministic algorithm in the Cross-query model.

Single Element Recovery

Consider a non-negative real-valued vector $x \in \mathbb{R}^N_{\geq 0}$ and suppose we have access to x only via Linear or OR queries where the dimension is now N. We define the following problem which we call the *single-element recovery* problem (following the standard "support-recovery" problem in compressed sensing).

▶ **Problem 2** (Single-element recovery). *Given a non-negative real-valued vector $x \in \mathbb{R}^N_{\geq 0}$, accessed via either Linear-queries or OR-queries, output* any arbitrary *element*[4] *from the support* $\text{supp}(x)$.

To see how the above problem relates to Problem 1, consider the vector of possible edges incident to a single vertex. A spanning forest must find an edge incident to this vertex. This corresponds to solving Problem 2 on this vector. The problem is also interesting in its own right, with connections to *combinatorial group testing* [23, 24, 43], *compressed sensing* [19, 22, 33], and *coin-weighing problems.* [12, 30, 40, 50]. While most of these works have focused on recovering the full support, we ask the simpler question of just recovering a single element.

If one allows *randomization*, then one can use ℓ_0-samplers [34] to solve the above problem using $O(\log^2 N \log(\frac{1}{\delta}))$ Linear queries[5], *non-adaptively.* In fact, ℓ_0-samplers return a random element in $\text{supp}(x)$. The parameter δ is the error probability. There have been numerous applications of these (see the table in Figure 1 of [37], for instance), and indeed many applications (including the AGM [2] algorithm alluded to above) need only an arbitrary element in the support. This is precisely what is asked in Problem 2. Furthermore, the upper bound for randomized algorithms is nearly tight [34, 37], and therefore, for randomized algorithms, our understanding is pretty much complete. But what can be said about *deterministically* finding a single support element[6]? This is an important question for it relates to deterministic analogs to the various applications stated above.

It is not too hard to make a couple of observations. One, any *non-adaptive* deterministic algorithm for Problem 2 using Linear-queries can in fact be recursively used to completely recover the whole vector. This implies an $\tilde{\Omega}(N)$ information theoretic lower bound. Two, if one allows more rounds, then one can indeed do better using a binary-search style idea. More precisely, in each round the algorithm partitions the search space into $N^{1/r}$ parts and using $N^{1/r}$ queries finds a non-zero part. In this way in r rounds, one can gets algorithm making $N^{1/r}$-queries per round. This leads to the following fundamental question which we answer in our paper.

> *What is the rounds-of-adaptivity versus query-complexity trade-off for deterministic algorithms for Problem 2?*

1.1 Motivation and Perspective

Why should we care about the questions above?

- We think that algorithmic question of computation on graphs via queries is as natural and important as the reconstruction question. Indeed, our study was inspired by trying to understand the power of *cut*-queries to check whether a graph was connected or not; this

[4] In the case of OR-queries, we can only return the j with $x_j > 0$
[5] A similar result holds with OR queries as well. See Section 3
[6] A "deterministic ℓ_0 sampler", if you allow us the abuse of notation.

is an (extremely) special case of submodular function minimization. More recently, this type of "property-testing via queries" question on graphs has been asked for matchings by Nisan [44], and more generally for matrix properties by [51] and [46]. Single element recovery is also as natural as whole-vector recovery. Indeed, one can imagine a scenario where recovering a big[7] subset of the support (diseased blood samples, say) faster and with fewer queries may be more beneficial than reconstructing the whole vector.

- The Linear query model is closely connected to *linear sketches* that have found plenty of applications in dynamic streaming; see, e.g. [29, 35, 39]. The single element recovery problem also has connections to the *universal relation* $\mathbf{UR}^{\subset}$ problem in communication complexity, which was studied in [37, 42]. Understanding these questions, therefore, have ramifications to other areas. As a concrete example, one consequence of our results is a deterministic, $O(r)$-pass dynamic streaming algorithm for graph connectivity in $\widetilde{O}(n^{1+1/r})$ space. This was not known before.
- We believe the question of the trade-off between rounds versus query complexity is natural and important, especially in today's world of massively parallel computing. Such trade-offs are closely related to similar questions in communication complexity, number of passes in streaming algorithms, etc. It is worthwhile building up an arsenal of tools to attack such questions. Indeed, one main contribution of this paper is to show how LP-duality can be used as one such tool.
- Why do we focus on deterministic algorithms? Mainly because, as mentioned above, our understanding of the complexity of randomized algorithms for the problems above is near complete. However, in some applications one may require exponentially low error, or has to deal with an "adversary" (say, the one giving updates to a streaming algorithm) that is not oblivious to the algorithm's randomness; see, e.g. [10]. This further motivates the study of deterministic algorithms in this context. Furthermore, we need to design lower-bounding techniques which only work against deterministic algorithms, and this is of technical interest.

1.2 Our Results

Our first result is a tight lower bound for the question on single element recovery. The binary-search style algorithm mentioned above is the best one can do.

▶ **Result 1.** *For the single element recovery with Linear-query access, any r-round, deterministic algorithm must make $\geq N^{1/r} - 1$ queries in some round.*

We should remind the reader that the above lower bound is for vectors whose domain is non-negative rationals. In particular, it does not hold for Boolean vectors[8]. Moving to the continuous domain allows one to use tools from geometry, in particular duality theory and Caratheodory's theorem, to prove the tight lower bound.

As mentioned above, Linear queries are stronger than OR queries, and thus the above lower bound holds for OR queries as well. The proof for OR queries, however, is combinatorial, arguably simpler, and more importantly can be generalized to prove the following lower bound for Problem 1 as well.

---

[7] As we show later in Lemma 16, algorithmically we can get results when the "single" in single element recovery can be larger.

[8] Indeed, for Boolean vector with Linear queries one can recover the whole vector if the query vector has exponentially large coefficients. Even when the coefficients are small ($\{0, 1\}$ even), the vector can be recovered with $O(n/\log n)$-queries which is information theoretically optimal.

▶ **Result 2.** *Any r-round deterministic algorithm for finding a spanning forest, must make $\widetilde{\Omega}(n^{1+\frac{1}{r}})$-OR queries.*

As we explain below, the above smooth trade-off between rounds and query complexity is optimal, even when we allow the weaker BIS-queries. Algorithmically, we have the following result. We mention that such a result was not known even using Linear or Cross queries. A similar lower bound as in Result 2 with Cross-queries is left open.

▶ **Result 3.** *For any positive integer r, there exists an $O(r)$-round deterministic algorithm which makes $\tilde{O}\left(n^{1+\frac{1}{r}}\right)$-BIS queries per round, and returns a spanning forest of the graph.*

It is worth remarking that our algorithm with Linear queries (which is implied by the weaker BIS queries) above also implies an $O(r)$-pass $\tilde{O}(n^{1+1/r})$-space *deterministic* algorithm for maintaining a spanning forest in *dynamic* graph streams. As the edge updates arise, one simply updates the answers to the various queries made in each round. This result was not known before.

Finally, we show that for Problem 1, randomization is helpful in decreasing the number of rounds. More precisely, we consider Monte-Carlo algorithms.

▶ **Result 4.** *There exists a 2-round randomized algorithm for graph connectivity which makes $\tilde{O}(n)$-OR queries per round. There exists a 4-round randomized algorithm for graph connectivity which makes $\tilde{O}(n)$-BIS queries per round. Any non-adaptive, randomized algorithm for graph connectivity must make $\widetilde{\Omega}(n^2)$-OR queries.*

Table 1 summarizes our contributions.

Table 1 Summary of the state-of-the-art and our results for graph connectivity and single-element recovery problems. In each cell, we write the number of rounds followed by the query complexity per round. All lower bounds are with respect to the stronger model (Linear and OR). For upper bounds, if there is a discrepancy between the stronger and weaker models, we show this using a | as partition. Bold results are ours. The remaining results are folklore unless a reference is explicitly cited. The **?** indicates the main open question of our paper.

		Linear | Cross		OR | BIS	
		Upper Bound	Lower Bound	Upper Bound	Lower Bound
Single Element Recovery	Det	r , $N^{1/r}-1$	$\boldsymbol{r}$, $\boldsymbol{N^{1/r}-1}$	r , $N^{1/r}-1$	$\boldsymbol{r}$, $\boldsymbol{N^{1/r}-1}$
	Rand	$r=1$, $O(\log^2 N)$	$r=1$, $\Omega(\log^2 N)$ [34]	$r=1$, $O(\log^2 N)$	$r=1$, $\Omega(\log^2 N)$
Graph Connectivity	Det	$\boldsymbol{O(r)}$, $\boldsymbol{n^{1+1/r}}$	**?**	$\boldsymbol{O(r)}$, $\boldsymbol{n^{1+1/r}}$	$\boldsymbol{r}$, $\boldsymbol{\widetilde{\Omega}(n^{1+1/r})}$
	Rand	$\boldsymbol{r=1}$, $\widetilde{O}(n)$ [2] | $\widetilde{O}(n)$ [2,51]	r, $\Omega(n/\log n)$	$\boldsymbol{r=2|4}$, $\boldsymbol{\widetilde{O}(n)}$	$\boldsymbol{r=1}$, $\boldsymbol{\widetilde{\Omega}(n^2)}$

1.3 Technical Overview

In this section we give a technical overview of our results. These highlight the main underlying ideas and will assist in reading the detailed proofs which appear in the subsequent sections.

Overview of Result 1. It is relatively easy to prove an r-round lower bound for single element recovery in the OR-query model via an adversary argument (see [7]). At a high level, OR-queries only mildly interact with each other and can be easily fooled. Linear queries, on the other hand, strongly interact with each other. To illustrate: if we know $x(A)$ and $x(B)$ for $B \subseteq A$, then we immediately know $x(A \setminus B)$. This is untrue for OR-queries – if x has a non-zero entry in both A and B, nothing can be inferred about its entries in $A \setminus B$. Indeed, this power manifests itself in the non-adaptive, randomized algorithm using Cross-queries; it is important that we can use subtraction. This makes proving lower bounds against Linear-queries distinctly harder.

In our proof of Result 1, we use duality theory. To highlight our idea, for simplicity, let's consider a warmup *non-adaptive* problem. The algorithm has to ask $\ll \sqrt{N}$ queries, and on obtaining the response, needs to return a subset $S \subseteq [N]$ of size $\ll \sqrt{N}$ with the guarantee that $\text{supp}(x) \cap S$ is not empty. Note that if this were possible, then there would be a simple 2-round $o(\sqrt{N})$-algorithm – simply query the individual coordinates of S in the second round. This is what we want to disprove. Therefore, given the first round's $\ll \sqrt{N}$ queries, we need to show there exists responses such that *no matter* which set S of $\ll \sqrt{N}$ size is picked, there exists a feasible $x \in \mathbb{R}^N_{\geq 0}$ which sets all entries in S to 0. Note this is a $\exists\forall\exists$-statement. How does one go ahead establishing this?

We first observe that for a fixed response $\boldsymbol{a}$ and a fixed set S, whether or not a feasible $x \in \mathbb{R}^N_{\geq 0}$ exists is asking whether a system of linear inequalities has a feasible solution. Farkas Lemma, or taking the dual, tells us exactly when this is the case. The nice thing about the dual formulation is that the "response" $\boldsymbol{a}$ becomes a "variable" in the dual program, as it should be since we are trying to find it. To say it another way, taking the dual allows us to assert conditions that the response vector $\boldsymbol{a}$ must satisfy, and the goal becomes to hunt for such a vector. How does one do that? Well, the conditions are once again linear inequalities, and we again use duality. In particular, we use Farkas Lemma again to obtain conditions certifying the *non-existence* of such an $\boldsymbol{a}$. The final step is showing that the existence of this certificate is impossible. This step uses another tool from geometry – Carathedeory's theorem. Basically, it shows that if a certificate exists, then a *sparse* certificate must exist. And then a simple counting argument shows the impossibility of sparse certificates. This, of course, is an extremely high-level view and for just the warmup problem. In Section 2 we give details of this warmup, an also details of how one proves the general r-round lower bound building on it.

The interested reader may be wondering about the two instantiations of duality (isn't the dual of the dual the primal?). We point out that duality can be thought of as transforming a $\exists$ statement into a $\forall$ statement: feasibility is a $\exists$ statement, Farkas implies infeasibility is a different $\exists$ statement, and negating we get the original feasibility as a $\forall$ statement. Since we were trying to assert a $\exists\forall\exists$-statement, the two instantiations of duality hit the two different $\exists$.

Overview of Result 2. At a high level, the lower bound for Problem 1, the spanning forest problem, boils down to a "direct sum" version of Problem 2, the single element recovery problem. Imagine the graph is an $n \times n$ bipartite graph. Therefore, finding a spanning forest requires us finding an edge incident to *each* of the n vertices on one side. This is precisely solving n-independent versions of Problem 2 in parallel. However, note that a single query can "hit" different instances at once. The question is, as all direct-sum questions are, does this make the problem n-times harder? We do not know the answer for Linear queries and leave this as the main open question of our work. However, we can show that the simpler,

combinatorial proof of Result 1 against OR-queries does have a direct-sum version, and gives an almost tight lower bound for Problem 1. This is possible because OR-queries, as mentioned in the previous paragraph, have only mild interaction between them. We show that this interaction cannot help by more than a poly(r)-factor. Our proof is an adversary argument, and a similar argument was used recently by Nisan [44] to show that matchings cannot be approximated well by deterministic algorithms with OR-queries. Details of this are given in [7].

Overview of Result 3. In Section 3, we show some simple, folklore, and known results for single element recovery. We build on these algorithms to obtain our algorithms for Problem 1. With every vertex one associates an unknown vector which is an indicator of its neighborhood. If one applies the r-round binary-search algorithm for the single element recovery problem on each such vector, then in r-rounds with $O(n^{1+1/r})$-BIS queries, for every vertex one can obtain a single edge incident on it. This alone however doesn't immediately help: perhaps, we only detect $n/2$ edges and get $n/2$ disconnected clusters. Recursively proceeding only gives an $O(r \log n)$-round algorithm. And we would like no dependence on n.

To make progress, we actually give a more sophisticated algorithm for single element recovery than binary search, which gives more and may be of independent interest. In particular, we describe an algorithm (Lemma 16) for single element recovery which in $O(r)$ rounds, and making $N^{1/r}$-queries per round, can in fact return as many as $N^{1/4r}$ elements in the support. Once we have this, then for graph connectivity we observe that in $O(r)$ rounds, we get polynomially many edges incident on each vertex. Thus as rounds go on, the number of effective vertices decreases, which allows us to query more aggressively. Altogether, we get an $O(r)$-round algorithm making only $\tilde{O}(n^{1+1/r})$-BIS queries. The details of this are described in Section 4.

Overview of Result 4. In the overview of the deterministic algorithm, we had to be a bit conservative in that even after every vertex found k edges (k being 1 or $n^{O(1/r)}$) incident on it, we pessimistically assumed that after this step the resulting graph still has $\Theta(n/k)$ disconnected clusters, and we haven't learned *anything* about the edges across these clusters. In particular, we allow for the situation that the cross-cluster edges can be dense. With randomization, however, we get to sample k *random* edges incident on a vertex. This is where we use the recent result of Holm *et al.* [31] which shows that if the k incident edges are random, then, as long as $k = \Omega(\log n)$, the number of *inter-component* edges between the connected components induced by the sampled edges, is $O(n/k)$. That is the cross-cluster edges are sparse. Therefore, in a single round with $\tilde{O}(n)$-randomized BIS queries, we can obtain a disconnected random subgraph, but one such that, whp, there exist at most $\tilde{O}(n)$ edges across the disconnected components.

Given the above fact, the algorithm is almost immediate. After round 1, we are in a sparse graph (where nodes now correspond to subsets of already connected vertices). If we were allowed general OR-queries, then a single round with $\tilde{O}(n)$-OR queries suffices to learn this sparse graph, which in turn, gives us a spanning forest in the original graph. This follows from algorithms for single element recovery when the vector is promised to be sparse (discussed in Section 3). Unfortunately, these queries may not be BIS-queries; recall that BIS-queries are restricted to ask about edges across two subsets. Nevertheless, we can show how to implement the above idea using 2-extra rounds with only BIS-queries, giving a 4-round algorithm. Details can be found in [7].

To complement the above, we also prove that even with randomization, one cannot get non-adaptive (1-round) $o(n^2/\log^2 n)$-query algorithms with OR-queries . Indeed, the family of examples is formed by two cliques (dense graphs) which could have a single edge, or not, that connects them. A single collection of $o(n^2/\log^2 n)$-OR queries cannot distinguish between these two families. Details can be found in [7].

1.4 Related Works

Our work falls in the broad class of algorithm design in the *query access model*, where one has limited access to the input. Over the years there has been a significant amount of work relevant to this paper including in graph reconstruction [1, 3, 4, 6, 12, 14, 15, 17, 30, 41, 47], parameter estimation [9, 11, 21, 48], minimum cuts [8, 49] sketching and streaming [2, 5, 8, 27, 28, 34, 36, 37, 42, 51], combinatorial group testing, compressed sensing, and coin weighing [12, 19, 22–25, 50]. It is impossible to do complete justice, but in the full version [7] we give a more detailed discussion of some of these works and how they fit in with our paper.

2 Lower Bound for Single Element Recovery

The following is the formal statement of the r-round lower bound for single element recovery.

▶ **Theorem 1.** *Any r-round deterministic algorithm for Single Element Recovery must make $\geq (N^{1/r} - 1)$-Linear queries in some round.*

In fact, we prove (see Corollary 9) that if $k_1, \ldots, k_r$ are r positive integers such that $\prod_{i=1}^{r}(k_i + 1) < N$, then no r-round algorithm making $\leq k_i$ queries in round i can be successful for the Single Element Recovery problem. This implies Theorem 1. To begin with, we give a proof for essentially the $r = 2$ case, which was the warm-up question we discussed in Section 1.3. More precisely, we prove that if $(k + 1)s < N$ then no one-round algorithm making $\leq k$ queries can return a subset S of size $\leq s$ with $x_j > 0$ for some $j \in S$. That is, a subset which traps an element in $\mathrm{supp}(x)$. This essentially implies the $r = 2$ case with $k_1 = k$ and $k_2 = s - 1$. This proof contains the core ideas behind the more general statement, which follows via an inductive application of the same idea. The complete proof of Theorem 1 can be found in Section 2.1. For now, we focus on proving the following statement.

▶ **Theorem 2.** *If $(k + 1)s < N$, then there cannot exist a 1-round deterministic algorithm making k-Linear queries for the trapping problem with parameter s.*

Note that if s divides N, then $k = \frac{N}{s} - 1$ queries suffice and so the above theorem is tight.

Proof. Since x is a non-zero, non-negative vector, by scaling, we assume that $x([N]) = 1$. We let $\mathbf{A}$ denote the $k \times N$ matrix corresponding to the k queries arranged as row vectors. We use $\boldsymbol{a} \in \mathbb{R}^k_{\geq 0}$ to denote the answers we will give to fool any algorithm. To find this, fix any subset S with $|S| \leq s$, and consider the following system of inequalities parametrized by the answer vector $\boldsymbol{a}$. The only inequalities are the non-negativity constraints. Below, and throughout, $[N] := \{1, 2, \ldots, N\}$.

$$\mathcal{P}(\boldsymbol{a}; S) = \{x \in \mathbb{R}^N_{\geq 0} \ : \ x([N]) = 1 \quad \mathbf{A} \cdot x = \boldsymbol{a} \quad x(S) = 0\} \tag{P}$$

Note that if $\mathcal{P}(\boldsymbol{a}; S)$ has a feasible solution, then given the answers $\boldsymbol{a}$ to its queries, the algorithms *cannot* return the subset S. This is because there is a non-negative x consistent with these answers with S disjoint from its support. In other words, S is *safe* for the lower

bound w.r.t. $\boldsymbol{a}$. Therefore, if there exists an answer vector $\boldsymbol{a}$ such that *every* subset $S \subseteq [N]$ with $|S| \leq s$ is safe with respect to $\boldsymbol{a}$, that is $\mathcal{P}(\boldsymbol{a}; S)$ is feasible, then we would have proved our lower bound. We use use duality and geometry to prove the existence of this vector (if $(k+1)s < N$).

The first step is to understand when for a *fixed* set S, the system $\mathcal{P}(\boldsymbol{a}; S)$ is infeasible. This is answered by Farkas Lemma. In particular, consider the following system[9] of inequalities where the variables are the Lagrange multipliers corresponding to the equalities in (P). We note that the variables are *free*, since $\mathcal{P}(\boldsymbol{a}; S)$ has only equalities in the constraints. For convenience, we have eliminated the variable corresponding to the subset S and have moved it to the right hand side. Below, and throughout, for any subset $S \subseteq [N]$, we use $\mathbf{1}_S$ to denote the N-dimensional 0, 1-vector which has 1 in the coordinates $i \in S$.

$$\mathcal{C}_S := \left\{ \ (y^{(0)}, \boldsymbol{y}) \in \mathbb{R} \times \mathbb{R}^k : \ \ y^{(0)} \cdot \mathbf{1}_{[N]} + \boldsymbol{y} \cdot \mathbf{A} \leq \mathbf{1}_S \ \right\}$$

Farkas Lemma asserts that $\mathcal{P}(\boldsymbol{a}; S)$ is *infeasible* if and only if there exists $(y^{(0)}, \boldsymbol{y}) \in \mathcal{C}_S$ such that $y^{(0)} \cdot 1 + \boldsymbol{y} \cdot \boldsymbol{a} > 0$. Contrapositively, we get that $\mathcal{P}(\boldsymbol{a}; S)$ is feasible, that is S is safe with respect to $\boldsymbol{a}$, iff $y^{(0)} + \boldsymbol{y} \cdot \boldsymbol{a} \leq 0$ for *all* $\boldsymbol{y} \in \mathcal{C}_S$. Since we want an answer $\boldsymbol{a}$ such that *all* subsets S with $|S| \leq s$ are safe, we conclude that such an answer exists if and only if the following system of linear inequalities has a feasible solution.

$$\mathcal{Q} := \left\{ \ \boldsymbol{a} \in \mathbb{R}^k_{\geq 0} : \quad \boldsymbol{y} \cdot \boldsymbol{a} \leq -y^{(0)}, \quad \forall S \subseteq [N], |S| \leq s, \ \ \forall (y^{(0)}, \boldsymbol{y}) \in \mathcal{C}_S \ \right\} \tag{D}$$

In summary, to prove the lower bound, it suffices to show that $\mathcal{Q}$ has a feasible solution, and this solution will correspond to the answers to the queries. Suppose, for the sake of contradiction, $\mathcal{Q}$ is *infeasible*. Then, again by Farkas Lemma (but on a different system of inequalities), there exists multipliers $\lambda_t > 0$ corresponding to constraints $\left(S_t \text{ s.t. } |S_t| \leq s, \ \ (y^{(0)}_t, \boldsymbol{y}_t) \in \mathcal{C}_{S_t}\right)$ for some $t = 1 \ldots T$ such that (P1) $\sum_{t=1}^{T} \lambda_t \boldsymbol{y}_t \geq \mathbf{0}_k$ where $\mathbf{0}_k$ is the k-dimensional all zero (row) vector, and (P2) $\sum_{t=1}^{T} \lambda_t y^{(0)}_t > 0$. Note that this time λ_t's are non-negative since $\mathcal{Q}$ has inequalities in the constraints.

We can focus on the λ_t's which are *positive* and discard the rest. The next key observation is to *upper bound* the size T of the support. Note that the conditions (P1) and (P2) can be equivalently stated as asserting that the $(k+1)$-dimensional *cone* spanned by the vectors $(y^{(0)}_t, \boldsymbol{y}_t)$ contains a non-negative point with first coordinate positive. Caratheodory's theorem (for cones) asserts that any such point can be expressed as a conic combination of at most $(k+1)$ vectors. Therefore, we can assume that $T \leq k+1$.

Now we are almost done. Since $(y^{(0)}_t, \boldsymbol{y}_t) \in \mathcal{C}_{S_t}$, we have $y^{(0)}_t \cdot \mathbf{1}_{[N]} + \boldsymbol{y}_t \cdot \mathbf{A} \leq \mathbf{1}_{S_t}$. Taking λ_t combinations and adding, we get (since all $\lambda_t > 0$) that

$$\left(\sum_{t=1}^{T} \lambda_t y^{(0)}_t\right) \cdot \mathbf{1}_{[N]} + \left(\sum_{t=1}^{T} \lambda_t \boldsymbol{y}_t\right) \cdot \mathbf{A} \ \leq \ \sum_{t=1}^{T} \lambda_t \mathbf{1}_{S_t}$$

Since every $|S_t| \leq s$, the support of the right hand side vector is $\leq sT \leq s(k+1)$. The support of the left hand side vector is $= N$. This is because the second summation is a non-negative vector by (P1), and the first has full support. This contradicts $(k+1)s < N$. Hence, $\mathcal{Q}$ has a feasible solution, which in turn means there exists answers $\boldsymbol{a}$ which foils $\mathbf{A}$. This proves Theorem 2. ◀

[9] Here $\boldsymbol{y}$ and $\mathbf{1}_S$ are *row vectors*. In the general proof, there will be multiple $\boldsymbol{y}$'s indexed with super-scripts. All of them are row-vectors.

2.1 The General r-round Lower Bound

We begin by formally defining what an r-round deterministic algorithm is, and what it means for such an algorithm to successfully solve Single Element Recovery.

▶ **Definition 3** (r-round deterministic algorithm.)**.** *An r-round deterministic algorithm $\mathcal{A}$ proceeds by making a collection of linear queries $\mathbf{A}^{(1)} \in \mathbb{R}_{\geq 0}^{k_1 \times N}$ and obtains the answer $\boldsymbol{a}^{(1)} = \mathbf{A}^{(1)} \cdot x$. This is the first round of the algorithm. For $1 < i \leq r$, in the ith round the algorithm makes a collection of linear queries $\mathbf{A}^{(i)} \in \mathbb{R}_{\geq 0}^{k_i \times N}$. This matrix depends on the history $(\mathbf{A}^{(1)}, \boldsymbol{a}^{(1)}), \ldots, (\mathbf{A}^{(i-1)}, \boldsymbol{a}^{(i-1)})$. Upon making this query it obtains the answer $\boldsymbol{a}^{(i)} = \mathbf{A}^{(i)} \cdot x$. We call $\Pi_r := \big((\mathbf{A}^{(1)}, \boldsymbol{a}^{(1)}), \ldots, (\mathbf{A}^{(r)}, \boldsymbol{a}^{(r)})\big)$ the r-round **transcript** of the algorithm. The output of the deterministic algorithm $\mathcal{A}$ only depends on Π_r.*

*A vector $y \in \mathbb{R}_{\geq 0}^N$ is said to be **consistent** with respect to a transcript Π_r if $\mathbf{A}^{(i)} \cdot y = \boldsymbol{a}^{(i)}$ for all $1 \leq i \leq r$. A transcript $\Pi_r = \big((\mathbf{A}^{(1)}, \boldsymbol{a}^{(1)}), \ldots, (\mathbf{A}^{(r)}, \boldsymbol{a}^{(r)})\big)$ is **feasible** for the algorithm if there is some vector y consistent with respect to it, and if the algorithm indeed queries $\mathbf{A}^{(i)}$ given the $(i-1)$-round transcript $\big((\mathbf{A}^{(1)}, \boldsymbol{a}^{(1)}), \ldots, (\mathbf{A}^{(i-1)}, \boldsymbol{a}^{(i-1)})\big)$*

▶ **Definition 4.** *An r-round deterministic algorithm $\mathcal{A}$ is said to successfully solve Single Element Recovery if for all non-zero $x \in \mathbb{R}_{\geq 0}^N$, upon completion of r-rounds the algorithm $\mathcal{A}$ returns a coordinate $j \in [N]$ with $x_j > 0$. In particular, if the algorithm returns a coordinate j given a feasible transcript Π_r, then* every *x that is consistent with Π_r must have $x_j > 0$.*

For technical reasons, we add a 0th-round for any r-round algorithm. In this round, the query "matrix" $\mathbf{A}^{(0)}$ is the single N-dimensional row with all ones. That is, we ask for the sum of x_j for all $j \in [N]$. We assume that the answer $\boldsymbol{a}^{(0)}$ is the scalar 1 to capture the fact that the vector x is non-zero.

Next we define the notion of safe subsets with respect to a transcript generated till round i. A safe subset of coordinates are those for which there is a consistent vector x whose support is disjoint from the subset, that is, $x_j = 0$ for all $j \in S$, or equivalently $x(S) = 0$ since $x \geq 0$.

▶ **Definition 5.** *Given an i-round transcript $\Pi_i = \big((\mathbf{A}^{(0)}, \boldsymbol{a}^{(0)}), \ldots, (\mathbf{A}^{(i)}, \boldsymbol{a}^{(i)})\big)$, a subset $S \subseteq [N]$ is* **safe** *w.r.t. Π_i if the following system of linear inequalities*

$$\mathcal{P}(\boldsymbol{a}^{(\leq i)}; S) := \left\{ \quad x \in \mathbb{R}^N : \begin{cases} \mathbf{A}^{(j)} \cdot x = \boldsymbol{a}^{(j)} & \forall 0 \leq j \leq i \\ x(S) = 0 \\ x \geq 0 \end{cases} \right\} \tag{Primal}$$

has a feasible solution.

▷ Claim 6. If Π_r is a feasible r-round transcript of an algorithm $\mathcal{A}$ such that *all* singletons are safe w.r.t Π_r, then the algorithm $\mathcal{A}$ cannot be successful in solving Single Element Recovery.

Proof. Given Π_r, the algorithm $\mathcal{A}$ must return some coordinate $j \in [N]$. However $\{j\}$ is safe. That is, there is a feasible solution x to $\mathcal{P}(\boldsymbol{a}^{(\leq r)}, \{j\})$. Indeed, if x were the input vector, the algorithm would return a coordinate not in the support. ◁

▶ **Definition 7** (Transcript Creation Procedure)**.** *Given an r-round algorithm $\mathcal{A}$, the transcript creation procedure is the following iterative process. In round i, given the transcript $\Pi_{i-1} := \big((\mathbf{A}^{(0)}, \boldsymbol{a}^{(0)}), \ldots, (\mathbf{A}^{(i-1)}, \boldsymbol{a}^{(i-1)})\big)$ upon which the algorithm $\mathcal{A}$ queries $\mathbf{A}^{(i)}$, and the transcript creation procedure produces an answer $\boldsymbol{a}^{(i)}$ such that $\Pi_i = \Pi_{i-1} \circ (\mathbf{A}^{(i)}, \boldsymbol{a}^{(i)})$ is feasible.*

Our main theorem, which implies Theorem 1, is the following.

▶ **Theorem 8** (Transcript Creation Theorem). *Let $k_1, \ldots, k_r$ and $s_0, s_1, \ldots, s_r$ be positive integers such that $s_0 \leq n-1$ and $(k_i+1)s_i \leq s_{i-1}$ for all $i \geq 1$. Then given any r-round algorithm $\mathcal{A}$ making $\leq k_i$ queries in round i, there is a transcript creation procedure to create an r-round transcript such that for all $0 \leq i \leq r$, any subset $S \subseteq [N]$ with $|S| \leq s_i$ is safe with respect to Π_i.*

▶ **Corollary 9.** *Let $k_1, \ldots, k_r$ be any r positive integers with $\prod_{i=1}^{r}(k_i+1) < n$. No r-round algorithm $\mathcal{A}$ which makes $\leq k_i$ queries in round i can be successful for the Single Element Recovery problem. In particular, this implies Theorem 1.*

Proof. Set $s_r = 1$, $s_{r-1} = (k_r+1)$, and in general, $s_i = (k_r+1)(k_{r-1}+1)\cdots(k_{i+1}+1)$. Note that the conditions of Theorem 8 are satisfied. Therefore given any algorithm $\mathcal{A}$ making $\leq k_i$ queries in round i, we can create a r-round transcript such that all singleton sets are safe with respect to Π_r. Claim 6 implies $\mathcal{A}$ cannot be succesful. ◀

2.1.1 Proof of the Transcript Creation Theorem

We start with writing the dual representation of safe sets. Fix a subset $S \subseteq [N]$ and a transcript Π_i. By Farkas lemma we know that the system $\mathcal{P}(\boldsymbol{a}^{(\leq i)}; S)$ is infeasible only if there exists a infeasibility certificate

$$\left(\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(i)}\right) \in \mathbb{R} \times \mathbb{R}^{k_1} \times \cdots \mathbb{R}^{k_i} \ : \ \sum_{j=0}^{i} \boldsymbol{y}^{(j)} \cdot \mathbf{A}^{(j)} \leq \mathbf{1}_S \quad \text{and} \quad \sum_{j=0}^{i} \boldsymbol{y}^{(j)} \cdot \boldsymbol{a}^{(j)} > 0$$

Here $\mathbf{1}_S$ is the n-dimensional indicator vector of the subset S, that is, it has 1 in the coordinates $j \in S$ and 0 otherwise. Taking negations, we get that the system $\mathcal{P}(\boldsymbol{a}^{(\leq i)}; S)$ is *feasible*, that is $S \subseteq [N]$ is safe w.r.t Π_{i-1}, if and only if the following condition holds

$$S \text{ is safe w.r.t. } \Pi_i \text{ iff} \quad \boldsymbol{y}^{(\leq i)} \cdot \boldsymbol{a}^{(\leq i)} := \sum_{j=0}^{i} \boldsymbol{y}^{(j)} \cdot \boldsymbol{a}^{(j)} \leq 0 \quad \text{for all} \quad \boldsymbol{y}^{(\leq i)} \in \mathcal{C}_S^{(i)} \quad \text{(Dual)}$$

where,

$$\mathcal{C}_S^{(i)} := \left\{ \ \boldsymbol{y}^{(\leq i)} := (\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(i)}) \in \mathbb{R} \times \mathbb{R}^{k_1} \times \cdots \times \mathbb{R}^{k_i} : \ \sum_{j=0}^{i} \boldsymbol{y}^{(j)} \cdot \mathbf{A}^{(j)} \leq \mathbf{1}_S \ \right\}$$

We are now ready to prove Theorem 8 via induction on i. The above representation is the dual definition of safe sets, and this definition is what is easy to induct with.

Base Case: $i = 0$. We need to show that any subset $S \subseteq [N]$ of size $|S| \leq s_0 = N-1$ is safe with respect to the transcript $(\mathbf{A}^{(0)}, \boldsymbol{a}^{(0)})$. To remind the reader, $\mathbf{A}^{(0)}$ is just the all ones vector and $\boldsymbol{a}^{(0)}$ is just the scalar 1. Using (Dual), we need to show for any subset $S \subseteq [N]$ with $|S| \leq N-1$, we must have

$$\boldsymbol{y}^{(0)} \cdot \boldsymbol{a}^{(0)} \leq 0 \ \text{ for all } \ \boldsymbol{y}^{(0)} \in \mathbb{R} \ \text{ such that } \ \boldsymbol{y}^{(0)} \cdot \mathbf{A}^{(0)} \leq \mathbf{1}_S$$

However, $\boldsymbol{y}^{(0)} \cdot \mathbf{A}^{(0)}$ is the n-dimensional vector which is $\boldsymbol{y}^{(0)}$ on all coordinates. Since $|S| \leq n-1$, there is some coordinate $j \notin S$ such that $\mathbf{1}_S[j] = 0$. Thus, $\boldsymbol{y}^{(0)} \leq 0$ implying $\boldsymbol{y}^{(0)} \cdot \boldsymbol{a}^{(0)} \leq 0$. The base case holds.

Inductive Case: $i \geq 1$. Assume the conclusion of the theorem holds for all $0 \leq j \leq i-1$. That is, there is a procedure which has created a transcript $\Pi_{i-1} = \left((\mathbf{A}^{(0)}, \boldsymbol{a}^{(0)}), \cdots, (\mathbf{A}^{(i-1)}, \boldsymbol{a}^{(i-1)})\right)$ such that every subset $S \subseteq [N]$ with $|S| \leq s_{i-1}$ is safe w.r.t Π_{i-1}. Using (Dual), we can rewrite this as the following statement

$$\forall S \subseteq [N], |S| \leq s_{i-1}, \quad \text{for all } \boldsymbol{y}^{(\leq\, i-1)} \in \mathcal{C}_S^{(i-1)} \quad \text{we have} \quad \boldsymbol{y}^{(\leq\, i-1)} \cdot \boldsymbol{a}^{(\leq\, i-1)} \leq 0. \tag{IH}$$

Given Π_{i-1}, the algorithm $\mathcal{A}$ now queries $\mathbf{A}^{(i)}$ in round i. Our goal is to find answers $\boldsymbol{a}^{(i)} \in \mathbb{R}_{\geq 0}^{k_i}$ such that any subset $S \subseteq [N]$ with $|S| \leq s_i$ is safe w.r.t $\Pi_i = \Pi_{i-1} \circ (\mathbf{A}^{(i)}, \boldsymbol{a}^{(i)})$. Again referring to (Dual), we need to find $\boldsymbol{a}^{(i)} \in \mathbb{R}_{\geq 0}^{k_i}$ satisfying the following system of linear inequalities.

$$\mathcal{Q}^{(i)} := \left\{ \boldsymbol{a}^{(i)} \in \mathbb{R}_{\geq 0}^{k_i} : \quad \boldsymbol{y}^{(i)} \cdot \boldsymbol{a}^{(i)} \leq -\left(\boldsymbol{y}^{(\leq\, i-1)} \cdot \boldsymbol{a}^{(\leq\, i-1)}\right), \quad \forall S \subseteq [N], |S| \leq s_i, \ \ \forall \boldsymbol{y}^{(\leq i)} \in \mathcal{C}_S^{(i)} \right\}$$

Although it may appear that the above system has infinitely many constraints, it suffices to write the constraints for extreme points for the polyhedra $\mathcal{C}_S^{(i)}$'s. To complete the proof, we need to show that $\mathcal{Q}^{(i)}$ is non-empty; if so, we can select any $\boldsymbol{a}^{(i)} \in \mathcal{Q}^{(i)}$ for completing the transcript creation procedure, and proving the theorem by induction. The next lemma does precisely that; this completes the proof of the theorem. ◀ Theorem 8

▶ **Lemma 10.** *The system of inequalities $\mathcal{Q}^{(i)}$ has a feasible solution.*

Proof. For the sake of contradiction, suppose not. Applying Farkas lemma (again), we get the following certificate of infeasibility. There exists the tuples ($\lambda_t > 0$, $S_t \subseteq [N]$ with $|S_t| \leq s_i$, $\boldsymbol{y}_t^{(\leq i)} \in \mathcal{C}_{S_t}^{(i)}$) for $1 \leq t \leq k_i + 1$ such that

(P1): $\sum_{t=1}^{k_i+1} \lambda_t \boldsymbol{y}_t^{(i)} \geq \mathbf{0}_{k_i}$, and

(P2): $\sum_{t=1}^{k_i+1} \lambda_t \left(\boldsymbol{y}_t^{(\leq\, i-1)} \cdot \boldsymbol{a}^{(\leq\, i-1)}\right) > 0$.

Since $\boldsymbol{y}_t^{(\leq i)} \in \mathcal{C}_{S_t}^{(i)}$, we get $\sum_{j=0}^{i} \boldsymbol{y}_t^{(j)} \cdot \mathbf{A}^{(j)} \leq \mathbf{1}_{S_t}$ for all $1 \leq t \leq k_i$. Taking the positive λ_t-combinations of these inequalities, we get

$$\sum_{t=1}^{k_i+1} \lambda_t \cdot \left(\sum_{j=0}^{i} \boldsymbol{y}_t^{(j)} \cdot \mathbf{A}^{(j)} \right) \leq \sum_{t=1}^{k_i+1} \lambda_t \mathbf{1}_{S_t} \tag{P3}$$

Now, define $\boldsymbol{w}^{(j)} := \sum_{t=1}^{k_i+1} \lambda_t \boldsymbol{y}_t^{(j)}$ for $0 \leq j \leq i$. (P1) above implies (Q1): $\boldsymbol{w}^{(i)} \geq \mathbf{0}_{k_i}$, and (P2) implies (Q2): $\boldsymbol{w}^{(\leq\, i-1)} \cdot \boldsymbol{a}^{(\leq\, i-1)} > 0$. And finally, (P3) translates to

$$\underbrace{\sum_{j=0}^{i-1} \boldsymbol{w}^{(j)} \cdot \mathbf{A}^{(j)}}_{\text{Call this } \boldsymbol{u}_1} + \underbrace{\boldsymbol{w}^{(i)} \cdot \mathbf{A}^{(i)}}_{\text{Call this } \boldsymbol{u}_2} \leq \underbrace{\sum_{t=1}^{k_i+1} \lambda_t \mathbf{1}_{S_t}}_{\text{Call this } \boldsymbol{v}} \tag{Q3}$$

Now we are ready to see the contradiction. First observe that the vector $\boldsymbol{v}$ has at most $(k_i+1)s_i$ positive entries since it is the sum of k_i+1 vectors each of support $\leq s_i$. Since $\boldsymbol{w}^{(i)}$ and $\mathbf{A}^{(i)}$ are both non-negative, $\boldsymbol{u}_2$ is a non-negative vector. This implies that $\boldsymbol{u}_1$ must have $\leq (k_i+1)s_i$ positive entries. From the conditions of the theorem, we get $(k_i+1)s_i \leq s_{i-1}$. Thus, $\boldsymbol{u}_1$ has $\leq s_{i-1}$ positive entries. This in turn implies there exists a scalar θ such that $\theta \boldsymbol{u}_1 \leq \mathbf{1}_S$ for some subset $S \subseteq [N]$ with $|S| \leq s_{i-1}$. That is,

$$\sum_{j=0}^{i-1} (\theta \boldsymbol{w}^{(j)}) \cdot \mathbf{A}^{(j)} \leq \mathbf{1}_S \quad \Rightarrow \quad \theta \boldsymbol{w}^{(\leq\, i-1)} \in \mathcal{C}_S^{(i-1)}$$

The induction hypothesis (IH) implies $\theta \boldsymbol{w}^{(\leq\, i-1)} \cdot \boldsymbol{a}^{(\leq\, i-1)} \leq 0$. This contradicts (Q2). This completes the proof of the lemma. ◀

3 Algorithms Warmup: Algorithms for Single Element Recovery

In this section, we state some simple and/or well known algorithms for single element recovery which we use these as subroutines for our algorithms for graph connectivity as well.

▶ **Lemma 11.** *Let $x \in \mathbb{R}^N_{\geq 0}$ be a non-zero, non-negative vector, and let r be a positive integer. There exists an r-round* deterministic *algorithm* $\mathsf{BinarySearch}_r(x)$ *which makes $(N^{1/r}-1)$-OR queries per round, and returns a coordinate j with $x_j > 0$. If Linear queries are allowed, then one can recover x_j as well.*

Proof. (Sketch) Divide $[N]$ into $N^{1/r}$ blocks each of size $N^{1-1/r}$, and run OR query on each block but the last, taking $N^{1/r}-1$ queries in all. If one of them evaluates to 1, recurse on that for the next $r-1$ rounds. Otherwise, recurse on the last block. ◀

Next, we state a standard result from the combinatorial group testing and coin-weighing literature [19, 25, 32, 38, 45] which says that if the support of x is known to be small, then there exist efficient *one-round* deterministic algorithms to recover the complete vector.

▶ **Lemma 12.** *[32, 45] Let $x \in \mathbb{R}^N_{\geq 0}$ be a non-zero vector, and let d be any positive integer. There exists a 1-round (non-adaptive) deterministic algorithm* $\mathsf{BndSuppRec}(x, d)$ *which makes $O(d^2 \log N)$-OR queries and (a) either asserts $\mathrm{supp}(x) > d$, or (b) recovers the full support of x. With Linear queries, the number of queries reduces to $O(d \log N)$.*

Proof. (Sketch) We give a very high level sketch only for the sake of completeness. For the case of $d = 1$, take the $O(\lceil \log N \rceil \times N)$ matrix A where column i is the number i represented in binary. Then Ax (the "OR product") points to the unique element in the support. To see the *existence* of a deterministic procedure for larger d, one can proceed by the probabilistic method. If one samples each coordinate with probability $1/d$, then with constant probability the vector restricted to this sample has precisely support 1 for which the above "$d = 1$" algorithm can be used to recover it. Repeating this $O(d \log N)$ times leads to error probability which swamps the union bound over $\leq N^d$ possible sets, implying the existence of a deterministic scheme. Finally, another $O(d)$ arises since we need to recover all the $\leq d$ coordinates. All this can be made explicit by using ideas from error correcting codes; we point the interested reader to [32, 45] for the details. ◀

Next we move to randomized algorithms. Here ideas from F_0-estimation [5, 27] and ℓ_0-sampling [18, 28, 34] give the following algorithms.

▶ **Lemma 13.** *Let $x \in \mathbb{R}^N_{\geq 0}$ be a non-zero vector. There exists a 1-round (non-adaptive) randomized algorithm* $\mathsf{RandSuppSamp}(x)$ *which makes $O(\log^2 N \log(\frac{1}{\delta}))$-OR queries and returns a random $j \in \mathrm{supp}(x)$ with probability $\geq 1-\delta$.*

Proof. (Sketch) Suppose we knew the support $\mathrm{supp}(x) = d$. Then, we sample each $j \in [N]$ with probability $1/d$ to get a subset $R \subseteq [N]$. With constant probability $\mathrm{supp}(x \cap R) = 1$ and, conditioned on that, it contains a random $j \in \mathrm{supp}(x)$. Therefore, running the algorithm $\mathsf{BndSuppRec}(x \cap R, 1)$ asserted in Lemma 12, we can find a random $j \in \mathrm{supp}(x)$ with constant probability. Repeating this $O(\log(1/\delta))$ times gives the desired error probability. Since we don't know $\mathrm{supp}(x)$, we run for various powers of 2 in 1 to N. ◀

▶ **Lemma 14.** *(Theorem 7 in [13], also in [20, 26]) Let $x \in \mathbb{R}^N_{\geq 0}$ be a non-zero vector. There exists a 1-round (non-adaptive) randomized algorithm* $\mathsf{SuppEst}(x)$ *which makes $O(\log N \cdot \log(1/\delta))$-OR-queries and returns an estimate $\tilde{s}$ of the support which satisfies $\frac{\mathrm{supp}(x)}{3} \leq \tilde{s} \leq 3\mathrm{supp}(x)$ with probability $\geq 1-\delta$.*

4 Deterministic Algorithm for Graph Connectivity

In this section, we prove the following theorem which formalizes Result 3.

▶ **Theorem 15.** *Let r be any fixed positive integer. There exists an $35r$-round deterministic algorithm* DetGraphConn(G) *which makes at most $O(n^{1+\frac{1}{r}}\log n)$-BIS-queries per round on an undirected multigraph G, and returns a spanning forest of G.*

We start by establishing some simple subroutines which we need. We will refer to some algorithmic paradigms defined in Section 3.

4.1 Simple Subroutines

We begin by strengthening the simple algorithm BinarySearch asserted in Lemma 11. While in r-rounds with $O(N^{1/r})$-OR queries BinarySearch$_r(x)$ recovers a single element in the support, one can in fact get many more elements from the support. This result may be of independent interest.

▶ **Lemma 16.** *Let $x \in \mathbb{R}_+^N$ be a non-zero, non-negative vector, and let r be a positive integer, and let $c < r$. There exists a $\lceil 2r/c \rceil$-round* deterministic *algorithm* DetFindMany$_{r,c}(x)$ *which makes $O(N^{c/r}\log N)$-OR queries per round, and returns $\min(N^{c/4r}, \text{supp}(x))$ distinct coordinates from $\text{supp}(x)$.*

Proof. In the first round, we partition the range $[N]$ into $N^{c/2r}$ blocks of size $N^{1-c/2r}$ each. Let these blocks be $B_1, \ldots, B_k$ with $k = N^{c/2r}$. For each $i \in [k]$, we run the algorithm BndSuppRec$(x \cap B_i, N^{c/4r})$ asserted in Lemma 12. The total number of queries used here is $O(N^{c/2r} \cdot \left(N^{c/4r}\right)^2 \log N) = O(N^{c/r}\log N)$.

At the end of this round, either we recover $\text{supp}(x \cap B_i)$ for each block, and thus recover $\text{supp}(x)$, and we are done. Or, there is at least one block of size $N^{1-c/2r}$ which is guaranteed to contain $\geq N^{c/4r}$ elements in its support. We call this the heavy block of round 1. Next, we now proceed to recover $N^{c/4r}$ elements from this heavy block of round 1.

In the second round, we partition the indices of this heavy block again into $N^{c/2r}$ blocks of size $N^{1-2c/2r}$ each, and run BndSuppRec again on this block with $d = N^{c/4r}$. Once again, either we recover the entire support of the heavy block (which is guaranteed to contain at least $N^{c/4r}$ elements) and we are done. Or find a block of size $N^{1-2c/2r}$ that contains at least $N^{c/4r}$ elements in its support– this is the heavy block of round 2 – and we now proceed to recover $N^{c/4r}$ elements in the heavy block of round 2.

We continue in this manner, and after $\lceil 2r/c \rceil - 1$ rounds, either we have already recovered at least $N^{c/4r}$ elements in the support of x, or have identified a heavy block of size $N^{1-\left(\left(\frac{2r}{c}-1\right)\cdot\frac{c}{2r}\right)} = N^{c/2r}$ that contains at least $N^{c/4r}$ elements in the support of x. In the final round, we can simply probe each entry completing the proof. ◀

▶ Remark 17. The trade-off between the number of queries and number of elements recovered is not tightly established for the purpose of what we need in the graph connectivity algorithm. For instance, using the same idea as above, in 2 rounds one can actually recover $\min(N^{1/4}, \text{supp}(x))$ coordinates making $O(N^{3/4})$-queries per round.

Next, we give an algorithm to find edges between two disjoint sets of vertices using BIS-queries.

▶ **Lemma 18.** *Let A and B be two* disjoint *sets of vertices with at least one edge between them. There exists a $2r$-round deterministic algorithm* DetFindEdge$_r(A, B)$ *which makes $O(|A|^{1/r} + |B|^{1/r})$-BIS queries per round, and returns an edge (a, b) with $a \in A$ and $b \in B$.*

Proof. Consider the $|B|$ dimensional vector x where x_b indicates the number of edges from a vertex $b \in B$ to vertices in A. We can simulate an OR-query in this vector using a BIS-query in the graph – for any subset $S \subseteq B$, $\mathsf{OR}(S)$ on x has the same answer as $\mathsf{BIS}(A, S)$. Therefore, using Lemma 11, in r-rounds and $|B|^{1/r}$-BIS-queries, we can find a coordinate $b^* \in B$ with $x_{b^*} > 0$. That is, there is an edge between b^* and some vertex in A.

We can find one such vertex $a \in A$ to which b^* has an edge, again as above. We define the $|A|$-dimensional vector y where y_a indicates the number of edges from b^* to a. Once again, the OR-query on y can be simulated using a BIS query on the graph – for any subset $S \subseteq A$, $\mathsf{OR}(S)$ on y is the same as $\mathsf{BIS}(S, \{b^*\})$. ◀

4.2 The Connectivity Algorithm

Now we give the $O(r)$-round deterministic algorithm to find a spanning forest. First, we need the following simple claim.

▷ Claim 19. Let $G(V, E)$ be an arbitrary connected multigraph graph on n vertices, and let D be an arbitrary integer in $\{0, 1, \ldots, (n-1)\}$. Let V_L denote all vertices in V which has at most D neighbors in G, and let $V_H = V \setminus V_L$. Let $E' \subseteq E$ be an arbitrary set of edges that satisfies the following property: for each vertex $u \in V_L$, the set E' contains all edges incident on u, and for every each vertex $v \in V_H$, the set E' contains D arbitrary edges incident on v. Then the graph $G' = (V, E')$ contains at most $\lfloor n/D \rfloor$ connected components.

Proof. Suppose G' has $K \geq \frac{n}{D}$ connected components. Thus, there must exist some component C with $\leq D$ vertices. Firstly, that C must have some vertex $v \in V_H$. If not, then since vertices in V_L have all their edges in G also in G', this component would be disconnected in G which contradicts G's connectedness. Secondly, observe that this leads to a contradiction: v has at least D neighbors in G', and since there are at most $D-1$ other vertices in C, one of v's neighbor in G' must lie outside C. This contradicts that C is a connected component. ◁

We are now ready to describe the algorithm $\mathsf{DetGraphConn}(G)$. For simplicity, assume G is connected and our goal is to find a spanning *tree*. Subsequently, we explain how to modify the algorithm to find a spanning forest of a general graph. The algorithm proceeds in $O(\log r)$ phases starting with phase 0. The input to phase i is a partition $\Pi_i = (S_1, \ldots, S_p)$ of the vertices. Each S_j in Π_i is guaranteed to be a connected in the graph G. Π_0 is the trivial partition of n singletons. Given Π_i, we define the graph $G_i = (\Pi_i, \mathcal{E}_i)$ where $\mathcal{E}_i$ is the collection of *pseudo-edges* between components: we have a pseudo-edge $(S_a, S_b) \in \mathcal{E}_i$ if and only if there exists some edge in G between a vertex $u \in S_a$ and a vertex $v \in S_b$. Thus, G_0 is indeed the original graph. Note that by our assumption that G is connected, all the G_i's are connected. We will be collecting pseudo-edges which will imply the connected components; we initialize this set $\mathcal{F}$ to empty set. We will maintain the following invariant for a phase: $|\Pi_i| \leq n^{1-\frac{4^i-1}{r}}$; this is certainly true for $i = 0$. Next, we describe a phase i.

1. For each $S \in \Pi_i$, we construct a vector x indexed by all sets in $\Pi_i \setminus S$ where x_T indicates whether there is a pseudo-edge (S, T) in G_i. Next, we run the algorithm $\mathsf{DetFindMany}_{r,c}(x)$ asserted in Lemma 16 to either find all pseudo-edges incident on S, or at least $n^{4^i/4r}$ of them. To do so, we set c such that $N^{c/4r} = n^{4^i/4r}$, where N is the dimension of x. That is, $N = |\Pi_i| - 1$.Indeed, we should set $c = \theta \cdot 4^i$ where $N^\theta = n$. Note, $\theta \geq 1$. Also note that the OR-queries on x can be simulated using BIS-queries on the original graph G. This is because we are looking at edges between S and a union of a subset of parts in $\Pi_i \setminus S$.

The number of rounds is $\lceil \frac{2r}{c} \rceil \leq \lceil \frac{2r}{4^i} \rceil$. The number of BIS-queries per round is $O(N^{c/r} \log N) = O(n^{4^i/r} \log n)$ per subset $S \in \Pi_i$. And thus, the total number of queries made is $N \cdot O(n^{4^i/r} \log n) \leq |\Pi_i| \cdot O(n^{4^i/r} \log n) \leq n^{1-\frac{4^i-1}{r}} \cdot O(n^{4^i/r} \log n) = O(n^{1+1/r} \cdot \log n)$.

2. Let $\mathcal{E}'_i \subseteq \mathcal{E}_i$ be the pseudo-edges obtained from the previous step. Let $\mathcal{F}'_i$ be an arbitrary spanning forest of $\mathcal{E}'_i$. We add all these edges to the collection $\mathcal{F}$. Note, $\mathcal{F}'_i$ is a collection of $\leq |\Pi_i|$ pseudo-edges.
3. Applying Claim 19 to the graph G_i, adding the pseudo edges in $\mathcal{E}'_i$ reduces the number of connected components to at most $|\Pi_i|/n^{\frac{4^i}{4r}}$. We now repeat the above two steps 11 more times *sequentially*, and each time the number of connected components multiplicatively drops by $n^{\frac{4^i}{4r}}$. Thus, after the 12 sub-phases we end up with the partition Π_{i+1} of connected components, with $|\Pi_{i+1}| \leq |\Pi_i|/n^{\frac{12 \cdot 4^i}{4r}} \leq n^{1-\frac{4^i-1}{r}} \cdot n^{-\frac{12 \cdot 4^i}{4r}} = n^{1-\frac{4^{i+1}-1}{r}}$, as desired. The second inequality follows from the invariant before phase $(i+1)$ started.

To summarize, Phase i performs $O(\frac{r}{4^i})$-rounds and makes $O(n^{1+1/r} \log n)$-BIS queries per round. We run phase 0 to $L = O(\log r)$, till we get $|\Pi_L| \leq \sqrt{n}$. After than we run a clean up phase.

4. *Clean-up Phase.* Once $|\Pi_L| = O(\sqrt{n})$, for each pair (S, T) in $\Pi_L \times \Pi_L$, we make a single BIS-query to detect if the pseudo-edge $(S, T) \in \mathcal{E}_L$. The total number of queries is $O(n)$. We add an arbitrary spanning tree of $\mathcal{E}_L$ to the set $\mathcal{F}$. At this point, $\mathcal{F}$ lets us know the structure of connectivity via pseudo-edges. The next step is to recover the actual graph edges.
5. *Tree Building Phase.* Note that the total number of pseudo-edges in $\mathcal{F}$ is $< n - 1$. For each $(S, T) \in \mathcal{F}$, we now desire to find an edge (s, t) in the graph where $s \in S$ and $t \in T$. Note that once we do this, we have the spanning tree the graph. This can be done in $2r$ more rounds using the algorithm $\mathsf{DetFindEdge}_r(S, T)$ using $O\left(|S|^{1/r} + |T|^{1/r}\right)$-BIS queries per round. Therefore, the total number of queries per round of this phase is $O(n) \cdot O(n^{1/r}) = O(n^{1+1/r})$.

The number of rounds is $\sum_{i=1}^{O(\log r)} \frac{24r}{4^i} + 1 + 2r \leq 35r$.

This ends the description of the algorithm when G is connected. If G had more than one connected component, then one can recognize the connected components as the algorithm progresses. More precisely, if the algorithm is processing the partition $\Pi_i = (S_1, \ldots, S_p)$ and find that S_i has no edges coming out of it, then by the invariant that S_i is connected, the algorithm can discard this component and proceed on the remaining graph as if it were connected. The analysis becomes better as the effective number of vertices decrease but the number of available queries don't. This completes the proof of Theorem 15.

References

1 Hasan Abasi and Nader H. Bshouty. On learning graphs with edge-detecting queries. *CoRR*, abs/1803.10639, 2018.

2 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proc., SODA*, pages 459–467, 2012.

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

4 Noga Alon, Richard Beigel, Simon Kasif, Steven Rudich, and Benny Sudakov. Learning a hidden matching. In *Proc., FOCS*, page 197, 2002.

5 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. System Sci.*, 58(1):137–147, 1999.

6 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.

7 Sepehr Assadi, Deeparnab Chakrabarty, and Sanjeev Khanna. Graph connectivity and single element recovery via linear and OR queries. *arXiv preprint arXiv:2007.06098*, 2020.

8 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Polynomial pass lower bounds for graph streaming algorithms. In *Proc., STOC*, pages 265–276, 2019.

9 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.

10 Omri Ben-Eliezer, Rajesh Jayaram, David P. Woodruff, and Eylon Yogev. A framework for adversarially robust streaming algorithms. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, *Proc., ACM Symposium on Principles of Database Systems (PODS)*, 2020.

11 Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. Triangle estimation using polylogarithmic queries. *CoRR*, abs/1808.00691, 2018.

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

13 Nader H Bshouty. Lower bound for non-adaptive estimation of the number of defective items. In *Proc., International Symposium on Algorithms and Computation (ISAAC 2019)*, pages 2:1–2:9, 2019.

14 Nader H Bshouty and Hanna Mazzawi. Reconstructing weighted graphs with minimal query complexity. *Theoretical Computer Science*, 412(19):1782–1790, 2011.

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

16 Sung-Soon Choi. Polynomial time optimal query algorithms for finding graphs with arbitrary real weights. In *Proc., Conf. on Learning Theory*, pages 797–818, 2013.

17 Sung-Soon Choi and Jeong Han Kim. Optimal query complexity bounds for finding graphs. In *Proc., STOC*, pages 749–758, 2008.

18 Graham Cormode and Donatella Firmani. A unifying framework for ℓ_0-sampling algorithms. *Distributed and Parallel Databases*, 32(3):315–335, 2014.

19 Graham Cormode and S Muthukrishnan. Combinatorial algorithms for compressed sensing. In *SIROCCO*, pages 280–294. Springer, 2006.

20 Peter Damaschke and Azam Sheikh Muhammad. Competitive group testing and learning hidden vertex covers with minimum adaptivity. *Discrete Mathematics, Algorithms and Applications*, 2(03):291–311, 2010.

21 Holger Dell and John Lapinskas. Fine-grained reductions from approximate counting to decision. In *Proc., STOC*, pages 281–288. ACM, 2018.

22 David L Donoho et al. Compressed sensing. *IEEE Transactions on information theory*, 52(4):1289–1306, 2006.

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

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

25 AG D'yachkov, VS Lebedev, PA Vilenkin, and SM Yekhanin. Cover-free families and superimposed codes: constructions, bounds and applications to cryptography and group testing. In *Proceedings. 2001 IEEE International Symposium on Information Theory (IEEE Cat. No. 01CH37252)*, page 117. IEEE, 2001.

26 Moein Falahatgar, Ashkan Jafarpour, Alon Orlitsky, Venkatadheeraj Pichapati, and Ananda Theertha Suresh. Estimating the number of defectives with group testing. In *Proc., IEEE International Symposium on Information Theory (ISIT)*, pages 1376–1380, 2016.

27 Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. System Sci.*, 31(2):182–209, 1985.

28 Gereon Frahling, Piotr Indyk, and Christian Sohler. Sampling in dynamic data streams and applications. *International Journal of Computational Geometry & Applications*, 18:3–28, 2008.

29 Sumit Ganguly. Lower bounds on frequency estimation of data streams. In *International Computer Science Symposium in Russia (CSR)*, pages 204–215, 2008.

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

31 Jacob Holm, Valerie King, Mikkel Thorup, Or Zamir, and Uri Zwick. Random k-out subgraph leaves only $O(n/k)$ inter-component edges. In *Proc., FOCS*, pages 896–909. IEEE, 2019.

32 FK Hwang and VT Sós. Non-adaptive hypergeometric group testing. *Studia Sci. Math. Hungar*, 22(1-4):257–263, 1987.

33 Piotr Indyk. Explicit constructions for compressed sensing of sparse signals. In *Proc., SODA*, pages 30–33, 2008.

34 Hossein Jowhari, Mert Sağlam, and Gábor Tardos. Tight bounds for ℓ_p samplers, finding duplicates in streams, and related problems. In *Proc., ACM Symposium on Principles of Database Systems (PODS)*, pages 49–58, 2011.

35 John Kallaugher and Eric Price. Separations and equivalences between turnstile streaming and linear sketching. In *Proc., STOC*, pages 1223–1236, 2020.

36 Michael Kapralov, Yin Tat Lee, CN Musco, Christopher Paul Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. *SIAM Journal on Computing (SICOMP)*, 46(1):456–477, 2017.

37 Michael Kapralov, Jelani Nelson, Jakub Pachocki, Zhengyu Wang, David P Woodruff, and Mobin Yahyazadeh. Optimal lower bounds for universal relation, and for samplers and finding duplicates in streams. In *Proc., FOCS*, pages 475–486, 2017.

38 W Kautz and Roy Singleton. Nonrandom binary superimposed codes. *IEEE Transactions on Information Theory*, 10(4):363–377, 1964.

39 Yi Li, Huy L Nguyen, and David P Woodruff. Turnstile streaming algorithms might as well be linear sketches. In *Proc., STOC*, pages 174–183, 2014.

40 Bernt Lindström. On Möbius functions and a problem in combinatorial number theory. *Canadian Mathematical Bulletin*, 14(4):513–516, 1971.

41 Hanna Mazzawi. Optimally reconstructing weighted graphs using queries. In *Proc., SODA*, pages 608–615, 2010.

42 Jelani Nelson and Huacheng Yu. Optimal lower bounds for distributed and streaming spanning forest computation. In *Proc., SODA*, pages 1844–1860, 2019.

43 Hung Q Ngo and Ding-Zhu Du. A survey on combinatorial group testing algorithms with applications to dna library screening. *Discrete mathematical problems with medical applications*, 55:171–182, 2000.

44 Noam Nisan. The demand query model for bipartite matching. *Proc., SODA*, pages 592–599, 2021.

45 Ely Porat and Amir Rothschild. Explicit non-adaptive combinatorial group testing schemes. In *Proc., ICALP*, pages 748–759, 2008.

46 Cyrus Rashtchian, David P. Woodruff, and Hanlin Zhu. Vector-matrix-vector queries for solving linear algebra, statistics, and graph problems. In *Proc., International Workshop on Randomization and Computation (RANDOM)*, pages 26:1–26:20, 2020.

47 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.

48 Dana Ron and Gilad Tsur. The power of an example: Hidden set size approximation using group queries and conditional sampling. *ACM Transactions on Computation Theory (TOCT)*, 8(4):15, 2016.

49 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.

50 Miklós Ruszinkó and Peter Vanroose. How an Erdős-Rényi-type search approach gives an explicit code construction of rate 1 for random access with multiplicity feedback. *IEEE Transactions on Information Theory*, 43(1):368–373, 1997.

51 Xiaoming Sun, David P. Woodruff, Guang Yang, and Jialin Zhang. Querying a matrix through matrix-vector products. In *Proc., ICALP*, pages 94:1–94:16, 2019.