

Nearly optimal independence oracle algorithms for edge estimation in hypergraphs

Holger Dell 

University of Frankfurt, Germany

IT University of Copenhagen and Basic Algorithms Research Copenhagen (BARC), Denmark

John Lapinskas 

University of Bristol, UK

Kitty Meeks 

University of Glasgow, UK

Abstract

Consider a query model of computation in which an n -vertex k -hypergraph can be accessed only via its independence oracle or via its colourful independence oracle, and each oracle query may incur a cost depending on the size of the query. Several recent results (Dell and Lapinskas, STOC 2018; Dell, Lapinskas, and Meeks, SODA 2020) give efficient algorithms to approximately count the hypergraph’s edges in the colourful setting. These algorithms immediately imply fine-grained reductions from approximate counting to decision, with overhead only $\log^{\Theta(k)} n$ over the running time n^α of the original decision algorithm, for many well-studied problems including k -Orthogonal Vectors, k -SUM, subgraph isomorphism problems including k -Clique and colourful- H , graph motifs, and k -variable first-order model checking.

We explore the limits of what is achievable in this setting, obtaining unconditional lower bounds on the oracle cost of algorithms to approximately count the hypergraph’s edges in both the colourful and uncoloured settings. In both settings, we also obtain algorithms which essentially match these lower bounds; in the colourful setting, this requires significant changes to the algorithm of Dell, Lapinskas, and Meeks (SODA 2020) and reduces the total overhead to $\log^{\Theta(k-\alpha)} n$. Our lower bound for the uncoloured setting shows that there is no fine-grained reduction from approximate counting to the corresponding uncoloured decision problem (except in the case $\alpha \geq k - 1$): without an algorithm for the colourful decision problem, we cannot hope to avoid the much larger overhead of roughly $n^{(k-\alpha)^2/4}$. The uncoloured setting has previously been studied for the special case $k = 2$ (Peled, Ramamoorthy, Rashtchian, Sinha, ITCS 2018; Chen, Levi, and Waingarten, SODA 2020), and our work generalises the existing algorithms and lower bounds for this special case to $k > 2$ and to oracles with cost.

2012 ACM Subject Classification Theory of computation \rightarrow Fixed parameter tractability; Theory of computation \rightarrow Oracles and decision trees; Mathematics of computing \rightarrow Graph algorithms

Keywords and phrases Graph oracles, Fine-grained complexity, Approximate counting, Hypergraphs

Funding *Kitty Meeks*: Supported by EPSRC grant EP/V032305/1.

1 Introduction

Many decision problems in computer science, particularly those in NP, can naturally be expressed in terms of determining the existence of a witness. For example, solving SAT requires determining the existence of a satisfying assignment to a CNF formula. All such problems Π naturally give rise to a counting version $\#\Pi$, in which we ask for the number of witnesses. It is well-known that $\#\Pi$ is often significantly harder than Π ; for example, Toda’s theorem implies that it is impossible to solve $\#\text{P}$ -complete counting problems in polynomial time with access to an NP-oracle unless the polynomial hierarchy collapses. However, the same is not true for *approximately* counting witnesses (to within a factor of two, say). For example, it is known that: if Π is a problem in NP, then there is an FPRAS for $\#\Pi$ using

an NP-oracle [36]; if Π is a problem in $W[i]$, then there is an FPTRAS for $\#\Pi$ using a $W[i]$ -oracle [30]; and that the Exponential Time Hypothesis is equivalent to the statement that there is no subexponential-time approximation algorithm for $\#3\text{-SAT}$ [14].

In this paper we are concerned with analogous results in the fine-grained setting, which considers exact running times rather than coarse-grained classifications such as polynomial, FPT, or subexponential; such results turn out to be inextricably bound to graph oracle results of independent interest.

Past work in this area has focused on the family of *uniform witness problems* [15]. Roughly speaking, these are problems which can be expressed as counting edges in a k -hypergraph G in which the edges correspond to witnesses and induced subgraphs correspond to sub-problems. (See Section 1.2 for a detailed definition.) Many of the most important problems in fine-grained and parameterised complexity can be expressed as uniform witness problems including $k\text{-SUM}$, $k\text{-OV}$, $k\text{-CLIQUE}$, Hamming weight- k solutions to CNFs, $\text{SIZE-}k\text{ GRAPH MOTIF}$, most subgraph detection problems (including weighted problems such as $\text{ZERO-WEIGHT } k\text{-CLIQUE}$ and $\text{NEGATIVE-WEIGHT TRIANGLE}$), and first-order model-checking [15], in addition to certain database queries [19] and patterns in graphs [10]. Here k may be either a constant, as in the case of $k\text{-SUM}$, or a parameter, as in the case of $k\text{-CLIQUE}$. In this setting, invoking a decision algorithm on a sub-problem of the original problem corresponds to invoking an oracle to test, given a set of vertices S , whether the induced subgraph $G[S]$ contains any edges; this oracle is called an *independence oracle* for G and is well-studied in its own right (see Section 1.3 for an overview).

Surprisingly, there is a partial analogue of the above reductions from approximate counting to decision in this setting. If the vertices of G are coloured, given a set $S \subseteq V(G)$, a *colourful independence oracle* tests whether $G[S]$ contains any edges with one vertex of each colour. This typically corresponds to a natural colourful variant of the original decision problem — for example, for $k\text{-CLIQUE}$, it corresponds to deciding whether a k -coloured graph contains a size- k clique with one vertex of each colour. These oracles are again well-studied in their own right (see Section 1.3), and for many but not all uniform witness problems they can be efficiently simulated using the independence oracle. Given access to a colourful independence oracle for a graph G , we can count G 's edges to within a factor of $1 \pm \varepsilon$ using $\varepsilon^{-2} k^{\mathcal{O}(k)} \log^{\Theta(k)} n$ oracle queries [15]. (See [5] for an improvement to the log factor.) In fact, we can say more — if we can simulate the colourful independence oracle in time n^{α_k} with $\alpha_k \geq 1$, then these queries dominate the running time and we obtain an approximate counting algorithm with running time $n^{\alpha_k} \cdot \varepsilon^{-2} k^{\mathcal{O}(k)} \log^{\Theta(k)} n$ in the usual word-RAM model. Translating back out of the oracle setting, this means that if we simulate the oracle by running an algorithm for the colourful decision problem, then for constant k and ε , we obtain an approximate counting algorithm with only polylogarithmic overhead over that decision algorithm. This result has led to several improved approximate counting algorithms — see [15] for applications to $k\text{-OV}$ over finite fields and graph motifs, [19] for applications to database queries, and [10] for applications to patterns in graphs.

We are left with two major open problems of concern to researchers in fine-grained complexity, parameterised complexity and graph oracles, and we expect our paper to be of interest to all three communities. First, can the result of [15] be generalised from colourful independence oracles to independence oracles? This would imply, for example, a fine-grained reduction from approximate induced sub-hypergraph counting to induced sub-hypergraph detection. In this setting, efficiently simulating the colourful independence oracle using the independence oracle requires solving a long-standing open problem — see Section 1.2 — so the result of [15] does not straightforwardly apply. Second, in the parameterised setting, the

factor of $\log^{\Theta(k)} n$ is not truly polylogarithmic, but equivalent to a factor of $k^{\mathcal{O}(k)} n^{o(1)}$. Can it be improved to $\log^{\mathcal{O}(1)} n$?

In this paper, we answer both questions, and in the process substantially generalise recent graph oracle results for the $k = 2$ case [13]. In both the colourful and uncoloured settings, we pin down the optimal oracle algorithm almost exactly. In both cases this algorithm improves on the current state of the art, and it allows for the desired fine-grained reductions if and only if the cost of calling the oracle on an x -vertex set (corresponding to the run-time of a decision algorithm on an x -element instance) is close to x^k . Moreover, our lower bounds are unconditional — they do not rely on conjectures such as SETH or $\text{FPT} \neq \text{W}[1]$.

In a little more detail, suppose for the moment that $\varepsilon = 1/2$, and that the cost of calling the oracle on an x -vertex set is x^{α_k} for some $\alpha_k \in [0, k]$. In the uncoloured setting, we define a function $g(k, \alpha_k) \approx (k - \alpha)^2 / (4k)$ (see (1.1.1)) and show that an overhead of $2^{\mathcal{O}(k)} n^{g(k, \alpha_k) \pm o(1)}$ is both achievable and required; we have $g(k, \alpha_k) = 0$ when $\alpha_k \geq k - 1$, so in this regime we obtain a fine-grained reduction. In the colourful setting, we show that the $\log^{\Theta(k)} n$ overhead of [15, 5] can be improved to $\log^{\Theta(k - \alpha_k)} n$, but no further; thus polylogarithmic overhead is possible if and only if $k - \alpha_k \in \mathcal{O}(1)$ as $k \rightarrow \infty$. For general values of ε , both of our upper bounds have an additional multiplicative overhead of $\mathcal{O}(\varepsilon^{-2})$, which is common in approximate counting algorithms.

In the rest of the introduction, we state our results for graph oracles more formally in Section 1.1, followed by their (immediate) corollaries for uniform witness problems in Section 1.2. We then give an overview of related work in Section 1.3, followed by a brief description of our proof techniques in Section 1.4.

1.1 Oracle results

Our results are focused on two graph oracle models on k -hypergraphs: independence oracles and colourful independence oracles. Both oracles are well-studied in their own right from a theoretical perspective, as they are both natural generalisations of group testing from unary relations to k -ary relations, and the apparent separation between them in power is already a source of substantial interest. They also provide a point of comparison for a rich history of sublinear-time algorithms for oracles which provide more local information, such as degree oracles. See the introduction of [13] for a more detailed overview of the full motivation, and Section 1.3 for a survey of past results.

In both the colourful and uncoloured case, while formally the oracles are bitstrings and a query takes $\mathcal{O}(1)$ time, in order to obtain reductions from approximate counting problems to decision problems in Section 1.2 we will simulate oracle queries using a decision algorithm. As such, rather than focusing on the *number* of queries as a computational resource, we define a more general *cost function* which will correspond to the running time of the algorithm used to simulate the query; thus the cost of a query will scale with its size. In our application, this allows for more efficient reductions by exploiting cheap queries, while also substantially strengthening our lower bounds. Indeed, simulating an oracle query typically requires between $\text{poly}(k)$ and $\text{poly}(n)$ time, so a lower bound on the total number of queries required would tell us very little; meanwhile, setting the cost of all queries to 1 in our results yields tight bounds for the number of queries required.

We are also concerned with the *running times* of our oracle algorithms, again due to our applications in Section 1.2. We work in the standard RAM-model of computation with $\Theta(\log n)$ bits per word and access to the usual $\mathcal{O}(1)$ -time arithmetic and logical operations on these words; in addition, oracle algorithms can perform oracle queries, which are considered to take $\mathcal{O}(1)$ time.

As shorthand, for all real $x, y > 0$ and $\varepsilon \in (0, 1)$, we say that x is an ε -approximation to y if $|x - y| < \varepsilon y$. We define an ε -approximate counting algorithm to be an oracle algorithm that is given n and k as explicit input, is given access to an oracle representing an n -vertex k -hypergraph G , and outputs an ε -approximation to the number of edges of G , denoted by $e(G)$. We allow ε to be either part of the input (for upper bounds) or fixed (for lower bounds).

1.1.1 Our results for the uncoloured independence oracle.

Given a k -hypergraph G with vertex set $[n]$, the (uncoloured) independence oracle is the bitstring $\text{IND}(G)$ such that for all sets $S \subseteq [n]$, $\text{IND}(G)_S = 1$ if $G[S]$ contains no edges and 0 otherwise. Thus a query to $\text{IND}(G)_S$ allows us to test whether or not the induced subgraph $G[S]$ contains an edge. We define the cost of an oracle call $\text{IND}(G)_S$ to be a polynomial function of the form $\text{cost}_k(S) = |S|^{\alpha_k}$, where the map $k \mapsto \alpha_k$ satisfies $\alpha_k \in [0, k]$ but is otherwise arbitrary. (This upper bound is motivated by the fact that we can trivially enumerate all edges of G by using $\mathcal{O}(n^k)$ queries to all size- k subsets of $[n]$, incurring oracle cost at most $n^k \cdot k^{\alpha_k}$.)

It is not too hard to show that the naive $\mathcal{O}(n^k)$ -cost exact edge-counting algorithm of querying every possible edge and the naive $\mathcal{O}(n^{\alpha_k})$ -cost algorithm to decide whether any edge is present by querying $[n]$ are both essentially optimal. For approximate counting we prove the following, where for all real numbers x we write $\lfloor x \rfloor := \lfloor x + 1/2 \rfloor$ for the value of x rounded to the nearest integer, rounding up in case of a tie.

► **Theorem 1** (Uncoloured independence oracle, polynomial cost function). *Let $\alpha_k \in [0, k]$ for all $k \geq 2$, let $\text{cost}_k(x) = x^{\alpha_k}$, and let*

$$g(k, \beta) := \frac{1}{k} \cdot \left\lfloor \frac{k - \beta}{2} \right\rfloor \cdot \left(k - \beta - \left\lfloor \frac{k - \beta}{2} \right\rfloor \right). \quad (1.1.1)$$

There is a randomised ε -approximate counting algorithm $\text{UncoI}(\text{IND}(G), \varepsilon, \delta)$ with failure probability at most δ , worst-case running time

$$\mathcal{O}\left(\log(1/\delta)(k^{5k} + \varepsilon^{-2} 2^{5k} \log^5 n \cdot n^{g(k, 1)} \cdot n)\right),$$

and worst-case oracle cost

$$\mathcal{O}\left(\log(1/\delta)(k^{7k} + \varepsilon^{-2} 2^{5k} \log^5 n \cdot n^{g(k, \alpha_k)} \cdot n^{\alpha_k})\right)$$

under cost_k . Moreover, every randomised $(1/2)$ -approximate edge-counting IND-oracle algorithm with failure probability at most $1/10$ has worst-case expected oracle cost $\Omega((n^{g(k, \alpha_k)} / k^{3k}) \cdot n^{\alpha_k})$ under cost_k .

Observe that the polynomial overhead $n^{g(k, \alpha_k)}$ of approximate counting over decision is roughly equal to $n^{(k - \alpha_k)^2 / (4k)}$. If $\alpha_k = 0$, then the worst-case oracle cost of an algorithm is simply the worst-case number of queries that it makes. Thus Theorem 1 generalises known matching upper and lower bounds of $\tilde{\Theta}(\sqrt{n})$ queries in the graph case [13], both by allowing $k > 2$ and by allowing $\alpha_k > 0$. (See Section 1.3 for more details.) Moreover, if $\alpha_k \geq k - 1$, then $g(k, \alpha_k) = 0$; thus in this case, Theorem 1 shows that approximate counting requires the same oracle cost as decision, up to a polylogarithmic factor. Taking $k = 2$ and $\alpha_k = 1$, this implies that whenever we can simulate an edge-detection oracle for a graph in linear time, then we can also obtain a linear-time approximate edge-counting algorithm (up to polylogarithmic factors). Analogous upper bounds on the running time and oracle cost of UncoI also hold for any “reasonable” cost function of the form $\text{cost}_k(n) = n^{\alpha_k + o(1)}$; for details, see Section 2.1.3 and Theorem 39.

1.1.2 Our results for the colourful independence oracle.

Given a k -hypergraph G with vertex set $[n]$, the *colourful independence oracle* is the bitstring $\text{cIND}(G)$ such that for all disjoint sets $S_1, \dots, S_k \subseteq [n]$, $\text{cIND}(G)_{S_1, \dots, S_k} = 1$ if G contains no edge $e \in E(G)$ with $|S_i \cap e| = 1$ for all i , and 0 otherwise. We view S_1, \dots, S_k as colour classes in a partial colouring of $[n]$; thus a query to $\text{cIND}(G)_{S_1, \dots, S_k}$ allows us to test whether or not G contains an edge with one vertex of each colour. (Note that we do not require $S_1 \cup \dots \cup S_k = [n]$.) Analogously to the uncoloured case, we define the *cost* of an oracle call $\text{cIND}(G)_{S_1, \dots, S_k}$ to be a polynomial function of the form $\text{cost}_k(S_1, \dots, S_k) = \text{cost}_k(|S_1| + \dots + |S_k|) = (|S_1| + \dots + |S_k|)^{\alpha_k}$, where the map $k \mapsto \alpha_k$ satisfies $\alpha_k \in [0, k]$ but is otherwise arbitrary.

It is not too hard to show that the naive $\mathcal{O}(n^k)$ -cost exact edge-counting algorithm of querying every possible edge and the naive $\mathcal{O}((k^k/k!)n^{\alpha_k})$ -cost algorithm to decide whether any edge is present by randomly colouring the vertices are both essentially optimal, and indeed we prove as [Proposition 65](#) that any such decision algorithm requires cost $\Omega(n^{\alpha_k})$. For approximate counting, we prove the following.

► **Theorem 2** (Colourful independence oracle, polynomial cost function). *Let $\alpha_k \in [0, k]$ for all $k \geq 2$, let $\text{cost}_k(x) = x^{\alpha_k}$, and let $T := \log(1/\delta)\varepsilon^{-2}k^{27k} \log^{4(k-\lceil\alpha_k\rceil)+18} n$. There is a randomised ε -approximate edge counting algorithm $\text{Count}(\text{cIND}(G), \varepsilon, \delta)$ with worst-case running time $\mathcal{O}(T \cdot n^{\max(1, \alpha_k)})$, worst-case oracle cost $\mathcal{O}(T \cdot n^{\alpha_k})$ under cost_k , and failure probability at most δ . Moreover, every randomised $(1/2)$ -approximate edge counting cIND -oracle algorithm with failure probability at most $1/10$ has worst-case oracle cost*

$$\Omega\left(k^{-9k} \left(\frac{\log n}{\log \log n}\right)^{k-\lfloor\alpha_k\rfloor-3} \cdot n^{\alpha_k}\right)$$

under cost_k .

The upper bound replaces a $\log^{\Theta(k)} n$ term in the query count of the previous best-known algorithm ([6] for $\alpha_k = 0$) by a $\log^{\Theta(k-\alpha_k)} n$ term in the multiplicative overhead over decision, giving polylogarithmic overhead over decision when $k - \alpha_k = \mathcal{O}(1)$. The lower bound shows that this term is necessary and cannot be reduced to $\log^{\mathcal{O}(1)} n$; this is a new result even for $\alpha_k = 0$. (See [Section 1.3](#) for more details.) Analogous upper bounds on the running time and oracle cost of Count also hold for any “reasonable” cost function of the form $\text{cost}_k(n) = n^{\alpha_k + o(1)}$; for details, see [Section 2.1.3](#) and [Theorem 48](#).

1.1.3 Approximate sampling results.

There is a known fine-grained reduction from approximate sampling to approximate counting [15]. Strictly speaking this, reduction is proved for $\alpha_k = 1$ with a colourful independence oracle, but the only actual use of the oracle in the reduction is to enumerate all edges in a set X with $\mathcal{O}(|X|^k)$ size- k queries, so it transfers immediately to our setting. The upper bounds of [Theorems 1](#) and [2](#) therefore also yield approximate sampling algorithms with overhead $2^{\mathcal{O}(k)} \log^{\mathcal{O}(1)} n$ over approximate counting.

1.1.4 A parameterised complexity motivation for our lower bound results.

To understand an important motivation for the lower bounds in our results, consider as an example the longest path problem: Given (G, k) , does there exist a simple path of length k ?

A long sequence of works in parameterised complexity led to a spectacular algorithm [8] for this problem in undirected graphs that runs in time $1.66^k \cdot \text{poly}(n)$. There is a somewhat shorter sequence of works for the corresponding approximate counting version of the problem, which culminated in a $4^k \text{poly}(n)$ -time algorithm [9, 24].

Instead of designing ever-more sophisticated algorithms for approximately counting k -paths in order to get closer to the running time of the decision problem, our dream result would instead be a subexponential-time approximate-counting-to-decision reduction that uses the decision problem in a black-box fashion and causes only a factor $2^{o(k)} \text{poly}(n)$ overhead in the running time. This way, any improvements to the decision algorithm would automatically carry over. One way to formalise what the black-box can do is captured by defining the k -hypergraph whose edges are the k -paths of the underlying graphs; using an algorithm for the k -path decision problem, it is trivial to simulate the independence oracle and easy to simulate the colourful independence oracle for this hypergraph.

Theorem 2 implies that any decision algorithm for k -path can be turned into an approximate counting algorithm by paying a $\log^{\mathcal{O}(k)} n$ -factor overhead in the running time. While this is still a fixed-parameter tractable running time, it leads to a useless algorithm, since the running time is much worse than $c^k \text{poly}(n)$. The main consequence of **Theorem 2** for k -path stems not from this meaningless upper bound, but from the lower bound, which is new even for $\alpha_k = 0$: Our results imply that if the decision algorithm for k -path is formalized using the colourful independence oracle, then the overhead of the approximate-counting-to-decision reduction must be $\log^{\Omega(k)} n$, and so a subexponential-time reduction cannot exist. Conversely, if a useful approximate-counting-to-decision reduction exists, it cannot merely be based on the hypergraph whose edges consist of all k -paths; instead, the reduction would have to have access to and exploit the underlying structure of the original graph. We believe that this is a useful insight for the design of future algorithms for approximate counting.

1.2 Reductions from approximate counting to decision

Theorems 1 and **2** can easily be applied to obtain reductions from approximate counting to decision for many important problems in fine-grained and parameterised complexity. The following definition is taken from [15]; recall that a counting problem is a function $\#\Pi: \{0, 1\}^* \rightarrow \mathbb{N}$ and its corresponding decision problem is defined via $\Pi = \{x \in \{0, 1\}^*: \#\Pi(x) > 0\}$.

► **Definition 3.** *The decision problem Π is a uniform witness problem if there is a function that maps instances $x \in \{0, 1\}^*$ to uniform hypergraphs G_x such that the following statements hold:*

- (i) $\#\Pi(x)$ is equal to the number $e(G_x)$ of edges in G_x ;
- (ii) $V(G_x)$ and the size $k(G_x)$ of edges in $E(G_x)$ can be computed from x in time $\tilde{\mathcal{O}}(|x|)$;
- (iii) *there exists an algorithm which, given x and $S \subseteq V(G_x)$, in time $\tilde{\mathcal{O}}(|x|)$ prepares an instance $I_x(S) \in \{0, 1\}^*$ such that $G_{I_x(S)} = G_x[S]$ and $|I_x(S)| \in \mathcal{O}(|x|)$.*

The set $E(G_x)$ is the set of witnesses of the instance x .

Intuitively, we can think of a uniform witness problem as a problem of counting witnesses in an instance x that can be naturally expressed as edges in a hypergraph G_x , in such a way that induced subgraphs of G_x correspond to sub-instances of x . This allows us to simulate a query to $\text{IND}(G_x)_S$ by running a decision algorithm for Π on the instance $I_x(S)$, and if our decision algorithm runs on an instance y in time $T(|y|)$ then this simulation will require time $\tilde{\mathcal{O}}(T(|S|))$. Typically there is only one natural map $x \mapsto G_x$, and so we consider it to be a part of the problem statement. Simulating the independence oracle in this way, the statement of **Theorem 1** yields the following.

► **Theorem 4.** *Suppose $\alpha_k \in [1, k]$ for all $k \geq 2$. Let Π be a uniform witness problem. Suppose that given an instance x of Π , writing $n = |V(G_x)|$ and $k = k(G_x)$, there is an algorithm to solve Π on x with error probability at most $1/3$ in time $\tilde{O}(n^{\alpha_k})$. Then there is an ε -approximation algorithm for $\#\Pi(x)$ with error probability at most $1/3$ and running time*

$$k^{\mathcal{O}(k)} + \varepsilon^{-2} n^{\alpha_k} \cdot 2^{\mathcal{O}(k)} n^{g(k, \alpha_k)}.$$

Note that the running time of the algorithm for $\#\Pi$ is the sum of the oracle cost and the running time of the algorithm of [Theorem 1](#); by requiring $\alpha_k \geq 1$, we ensure this is dominated by the oracle cost. (Indeed, for most uniform witness problems it is very easy to prove that every decision algorithm must read a constant proportion of the input, and so we will always have $\alpha_k \geq 1$.) The lower bound of [Theorem 1](#) implies that the $n^{\alpha_k + g(k, \alpha_k)}$ term in the running time cannot be substantially improved with any argument that relativises; thus in simple terms, there is a generic fine-grained reduction from approximate counting to decision if and only if the decision algorithm runs in time $\Omega(n^{k-1})$.

It is instructive to consider an example. Take Π to be the problem INDUCED- H of deciding whether a given input graph G contains an induced copy of a fixed graph H . In this case, the hypergraph corresponding to an instance G will have vertex set $V(G)$ and edge set $\{X \subseteq V(G) : G[X] \simeq H\}$; thus the witnesses are vertex sets which induce copies of H in G . The requirements of [Definition 3](#)(i) and (ii) are immediately satisfied, and [Definition 3](#)(iii) is satisfied since deleting vertices from the hypergraph corresponds to deleting vertices of G . Thus writing $k = |V(H)|$, [Theorem 4](#) gives us a reduction from approximate $\#\text{INDUCED-}H$ to INDUCED- H with overhead $\varepsilon^{-2} 2^{\mathcal{O}(k)} n^{g(k, \alpha_k)}$ over the cost of the decision algorithm. Moreover, on applying the fine-grained reduction from approximate sampling to counting in [\[15\]](#) we also obtain an approximate uniform sampling algorithm with overhead $\varepsilon^{-2} 2^{\mathcal{O}(k)} n^{g(k, \alpha_k)}$. Many more examples of uniform witness problems to which [Theorem 4](#) applies can be found in the introduction of [\[15\]](#).

We now describe the corresponding result in the colourful oracle setting, which we now set out — again, the following definition is taken from [\[15\]](#).

► **Definition 5.** *Suppose Π is a uniform witness problem. COLOURFUL- Π is defined as the problem of, given an instance $x \in \{0, 1\}^*$ of Π and a partition of $V(G_x)$ into disjoint sets S_1, \dots, S_k , deciding whether $\text{cIND}_{G_x}(S_1, \dots, S_k) = 0$ holds.*

Continuing our previous example, in COLOURFUL-INDUCED- H , we are given a (perhaps improper) vertex colouring of our input graph G , and we wish to decide whether G contains an induced copy of H with exactly one vertex from each colour. Simulating an oracle call to $\text{cIND}(G_x)_{S_1, \dots, S_k}$ corresponds to running a decision algorithm for COLOURFUL- Π on the instance $I_x(S_1 \cup \dots \cup S_k)$ with colour classes S_1, \dots, S_k , and if this decision algorithm runs on an instance y in time $T(|y|)$ then this simulation will require time $\tilde{O}(T(|S_1| + \dots + |S_k|))$. Simulating the colourful independence oracle in this way, the statement of [Theorem 2](#) yields the following.

► **Theorem 6.** *Suppose $\alpha_k \in [1, k]$ for all $k \geq 2$. Let Π be a uniform witness problem. Suppose that given an instance x of Π , writing $n = |V(G_x)|$ and $k = k(G_x)$, there is an algorithm to solve COLOURFUL- Π on x with error probability at most $1/3$ in time $\mathcal{O}(n^{\alpha_k})$. Then there is an ε -approximation algorithm for $\#\Pi(x)$ with error probability at most $1/3$ and running time*

$$\varepsilon^{-2} n^{\alpha_k} \cdot k^{\mathcal{O}(k)} (\log n)^{\mathcal{O}(k - \alpha_k)}.$$

As in the uncoloured case, the requirement $\alpha_k \geq 1$ ensures that the dominant term in the running time is the time required to simulate the required oracle queries, and the lower bound of [Theorem 2](#) implies that the $\log^{\mathcal{O}(k-\alpha_k)} n$ term in the running time cannot be substantially improved with any argument that relativises. Again writing $k = |V(H)|$, [Theorem 6](#) gives us a reduction from approximate $\#$ INDUCED- H to COLOURFUL-INDUCED- H with overhead $\varepsilon^{-2} k^{\mathcal{O}(k)} (\log n)^{\mathcal{O}(k-\alpha_k)}$ over the cost of the decision algorithm. This result improves on the reduction of [[15](#), Theorem 1.7] by a factor of $\log^{\Omega(\alpha_k)} n$, and using the fine-grained reduction from approximate sampling to counting in [[15](#)] it can immediately be turned into an approximate uniform sampling algorithm.

Observe that in most cases, there is far less overhead over decision in applying [Theorem 6](#) to reduce $\#$ INDUCED- H to COLOURFUL-INDUCED- H than there is in applying [Theorem 4](#) to reduce $\#$ INDUCED- H to INDUCED- H . In some cases, such as the case where H is a k -clique, there are simple fine-grained reductions from COLOURFUL-INDUCED- H to INDUCED- H , and in this case [Theorem 6](#) is an improvement. However, it is not known whether the same is true of all choices of H , and indeed even an FPT reduction from COLOURFUL-INDUCED- H to INDUCED- H would imply the long-standing dichotomy conjecture for the embedding problem introduced in [[21](#)]. More generally, but still within the setting of uniform witness problems, the problem of detecting whether a graph contains a size- k set which either spans a clique or spans an independent set is in FPT by Ramsey’s theorem, but its colourful version is W[1]-complete [[27](#)].

While the distinction between colourful problems and uncoloured problems is already well-studied in subgraph problems, these results strongly suggest that it is worth studying in many other contexts in fine-grained complexity as well. Indeed, it is easy to simulate $\text{IND}(G)$ from $\text{cIND}(G)$ with random colouring; thus the lower bound of [Theorem 1](#) and the upper bound of [Theorem 2](#) imply that there is a fine-grained reduction from uncoloured approximate counting to colourful decision, but not to uncoloured decision. By studying the relationship between colourful problems and their uncoloured counterparts, we may therefore hope to shed light on the relationship between approximate counting and decision.

Finally, we observe that the set of running times allowed by [Theorems 4](#) and [6](#) may not be sufficiently fine-grained to derive meaningful results for some problems. In fine-grained complexity, even a subpolynomial improvement to a polynomial-time algorithm may be of significant interest — the classic example is the NEGATIVE-WEIGHT-TRIANGLE algorithm of [[37](#)], which runs in $n^3 / e^{\Omega(\sqrt{\log n})}$ time on an n -vertex instance, compared to the naive $\mathcal{O}(n^3)$ -time algorithm. In order to “lift” such improvements from decision problems to approximate counting, we must generalise the upper bounds of [Theorems 1](#) and [2](#) to cost functions of the form $\text{cost}_k(x) = x^{\alpha_k \pm o(1)}$ while maintaining low overhead. We do so in [Theorems 39](#) and [48](#) for all “reasonable” cost functions, including any function of the form $\text{cost}_k(x) = n^{\alpha_k} \log^{\beta_k} n$ and any function of the form $\text{cost}_k(x) = n^{\alpha_k} e^{\pm (\log n)^{\gamma_k}}$ where $\gamma_k < 1$. A full list of technical requirements is given in [Section 2.1.3](#), but the most important one is regular variation — this is a standard notion from probability theory for “almost polynomial” functions, and requiring it avoids pathological cases where (for example) we may have $\text{cost}_k(x) = \mathcal{O}(x)$ as $x \rightarrow \infty$, but $\text{cost}_k(2x_i) = \omega(\text{cost}_k(x_i))$ as $i \rightarrow \infty$ along some sequence $(x_i: i \geq 1)$.

1.3 Discussion of related work

In order to compare algorithms without excessive re-definition of notation, throughout this subsection we consider the problem of ε -approximating the number of edges in an m -edge, n -vertex k -hypergraph.

Colourful and uncoloured independence oracles were introduced in [[4](#)] in the graph setting,

then first generalised to hypergraphs in [7]. Edge estimation using these oracles was first studied in the graph setting (i.e. for $k = 2$) in [4], which gave an $\varepsilon^{-4} \log^{\mathcal{O}(1)} n$ -query algorithm for colourful independence oracles and an $(\varepsilon^{-4} + \varepsilon^{-2} \min\{\sqrt{m}, n^2/m\}) \log^{\mathcal{O}(1)} n = (\varepsilon^{-4} + \varepsilon^{-2} n^{2/3}) \log^{\mathcal{O}(1)} n$ -query algorithm for uncoloured independence oracles. The connection to approximate counting in fine-grained and parameterised complexity was first studied in [14].

For colourful independence oracles in the graph setting, [14] (independently from [4]) gave an algorithm using $\varepsilon^{-2} \log^{\mathcal{O}(1)} n$ cIND queries and $\varepsilon^{-2} n \log^{\mathcal{O}(1)} n$ adjacency queries. [1] subsequently gave a *non-adaptive* algorithm using $\varepsilon^{-6} \log^{\mathcal{O}(1)} n$ cIND queries.

The case of edge estimation in k -hypergraphs (i.e. for arbitrary $k \geq 2$) was first considered independently by [15, 5]; [15] gave an algorithm using $\varepsilon^{-2} k^{\mathcal{O}(k)} \log^{4k+\mathcal{O}(1)} n$ queries, while [5] gave an $\varepsilon^{-4} k^{\mathcal{O}(k)} \log^{4k+\mathcal{O}(1)}$ -query algorithm. [15] also introduced a reduction from approximate sampling to approximate counting in this setting (which also applies in the uncoloured setting) with overhead $k^{\mathcal{O}(k)} \log^{\mathcal{O}(1)} n$. [6] then improved the query count further to $\varepsilon^{-2} k^{\mathcal{O}(k)} \log^{3k+\mathcal{O}(1)} n$.

In this paper, we give an algorithm with total query cost $\varepsilon^{-2} k^{\mathcal{O}(k)} \log^{4(k-\alpha_k)+\mathcal{O}(1)} n$ under $\text{cost}_k(x) = x^{\alpha_k}$, giving polylogarithmic overhead when $\alpha_k \approx k$. We also give a lower bound which shows that a $\log^{\Theta(k-\alpha_k)}$ term is necessary; no lower bounds were previously known even for $\alpha_k = 0$ (i.e. the case where the total query cost equals the number of queries).

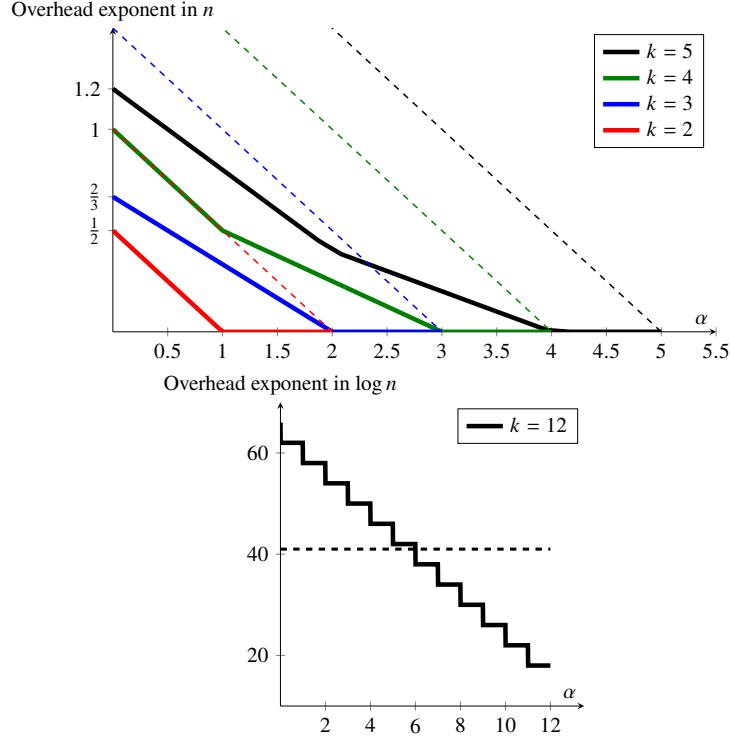
For uncoloured independence oracles of graphs, [13] improved the algorithm of [4] to use

$$\varepsilon^{-\Theta(1)} \min\{\sqrt{m}, n/\sqrt{m}\} \text{polylog } n = \varepsilon^{-\Theta(1)} \sqrt{n} \text{polylog } n \quad (1.3.1)$$

queries and gave a matching lower bound. (It is difficult to tell the exact value of the $\Theta(1)$ term from the proof, but it is at least 9 — see the definition of N in the proof of Lemma 3.9 on p. 15.) It is worth noting that the bound in (1.3.1) is stated as a function of both n and m . We believe that our results can be stated in such a way as well, but we defer doing so to the journal version of this paper.

To the best of our knowledge, no results on uncoloured edge estimation for $\alpha_k > 0$ or $k > 2$ have previously appeared in the literature. However, we believe it would be easy to partially generalise the proof of [4] to this setting. Very roughly speaking, their argument works by running a naive sampling algorithm and a more subtle branch-and-bound approximation algorithm in parallel, with the sampling algorithm running quickly on dense graphs and the branch-and-bound algorithm running quickly on sparse graphs. The main obstacle to generalising this approach would be the branch-and-bound approximation algorithm; however, by replacing it with a slower branch-and-bound *enumeration* algorithm for k -hypergraphs such as [28], we believe we would obtain worst-case oracle cost $k^{\mathcal{O}(k)} + \varepsilon^{-2} 2^{\mathcal{O}(k)} n^{\alpha_k + (k-\alpha_k)/2}$ under $\text{cost}_k(x) = x^{\alpha_k}$; this technique is well-known in the literature and also appears in e.g. [35]. By comparison (see Figure 1), the algorithm of Theorem 1 achieves a much smaller worst-case oracle cost of $k^{\mathcal{O}(k)} + \varepsilon^{-2} 2^{\mathcal{O}(k)} n^{\alpha_k + g(k, \alpha_k)}$, where $g(k, \alpha_k) \approx (k - \alpha_k)^2 / (4k) < (k - \alpha_k) / 2$ and where $g(k, \alpha_k) = 0$ for $\alpha_k \geq k - 1$. This substantially improves on the algorithm implicit in [4], and indeed is optimal up to a factor of $\varepsilon^{\Theta(1)} k^{\Theta(k)}$. Also, our algorithm has a better dependence on ε compared with [13] when $k = 2$; however, we bound the cost only in terms of n and not in terms of m .

Although a full survey is beyond the scope of this paper, there are natural generalisations of (colourful and uncoloured) independence oracles [33], and edge estimation problems are studied for other oracle models including neighbourhood access [34]. Other types of oracle are also regularly applied to fine-grained complexity in other models, notably including cut oracles [29, 3]. Perhaps surprisingly, we were unable to find any previous examples in the literature of unconditional oracle lower bounds relative to a general query cost function, and



■ **Figure 1** *Left:* If each IND-query of size x has cost x^{α_k} , then up to $k^{\mathcal{O}(k)} \log^{\mathcal{O}(1)} n$ factors, we show in Theorem 1 that $n^{g(k, \alpha_k) + \alpha_k}$ is the smallest possible IND-oracle cost to $(1/2)$ -approximate the number of edges. Plotted here in *thick lines* is the overhead $\alpha \mapsto g(k, \alpha)$ in the exponent of n for $k \in \{2, 3, 4, 5\}$, and in *dashed lines* is the larger overhead exponent $\alpha \mapsto (k - \alpha)/2$ obtained from a naive generalisation of the techniques of [4]. *Right:* If each cIND-query of size x has cost x^{α_k} , then up to $k^{\mathcal{O}(k)} (\log n)^{\mathcal{O}(1)} (\log \log n)^{\mathcal{O}(k - \alpha_k)}$ factors, we show in Theorem 2 that $n^{\alpha_k} \log^{4(k - \lceil \alpha_k \rceil) + 18} n$ is the smallest possible cIND-oracle cost to $(1/2)$ -approximate the number of edges. Plotted in *thick lines* is the overhead $\alpha \mapsto 4(k - \lceil \alpha \rceil) + 18$ in the exponent of $\log n$ for $k = 12$, and the *dashed line* depicts the overhead $\alpha \mapsto 3k + 5$ obtained by using the bound on the number of queries by [6]; our bound is better if $\lceil \alpha_k \rceil \geq k/4 + \Theta(1)$.

so our definitions and methods are novel in that sense. Of course, many existing works prove unconditional lower bounds in terms of query count [34], or consider algorithmic construction of graph oracles with bounded query times [23], or provide oracle algorithms with fast running times in addition to low query counts [31]. In our setting, however, a lower bound in terms of query cost is absolutely necessary. Recall that for us, query cost is the running time of an algorithm for the decision problem we are reducing to, and the algorithmic results of Theorems 4 and 6 all rely on smaller queries running faster; thus to prove they are “best possible” in any meaningful sense, we absolutely require the formal notion of query cost set out in Section 1.1.

Outside the oracle setting, it was recently proved [25] that any decision algorithm built around the representative family technique of [20] can be turned into an approximate counting algorithm with substantially lower overheads in k than those of [15]. More recently, several important decision problems in the fine-grained setting with $k = 3$ have been discovered to be “equivalent” to their *exact* counting versions [12]. This work is not directly comparable to ours, as they work in a substantially stronger setting and use a correspondingly weaker

notion of equivalence. Their equivalence is in the sense of equivalence of conjectures — for example, they prove that if there is an $O(n^{2-\varepsilon})$ -time algorithm for 3-SUM, then there exists $0 < \varepsilon' < \varepsilon$ such that there is an $O(n^{2-\varepsilon'})$ -time algorithm for #3-SUM. We stress that for exact counting problems as studied in [12], such equivalence results are genuinely deep and surprising. However, for the approximate counting problems we study, analogous results are typically quite easy to prove via the standard combination of sampling and branch-and-bound approaches discussed above, and so we instead study the stronger notion of fine-grained reductions from approximate counting to decision. Where no such reductions exist, we aim to nail down the exact value of ε' . (Recall also that there are uniform witness problems whose decision versions are in FPT but whose exact counting versions are W[1]-hard [27], so we cannot hope to extend our results to exact counting in this setting.)

1.4 Proof techniques

1.4.1 Colourful upper bound

We first discuss the proof of the upper bound of Theorem 2, our cIND-oracle algorithm for edge estimation using the colourful independence oracle of an n -vertex k -hypergraph G . In [15], it is implicitly proved that a cIND-oracle algorithm that computes a “coarse approximation” to $e(G)$, which is accurate up to multiplicative error b , can be bootstrapped into a full ε -approximation algorithm with overhead $\varepsilon^{-2} 2^{\mathcal{O}(k)} \log^{\mathcal{O}(1)} n \cdot b^2$. (See Theorem 49 of the present paper for details.) It therefore suffices to improve the coarse approximation algorithm of [15] from $k^{\mathcal{O}(k)} \log^{\Theta(k)} n$ queries and $k^{\mathcal{O}(k)} \log^{\Theta(k)} n$ multiplicative error to $k^{\mathcal{O}(k)} \log^{\Theta(k-\alpha_k)} n$ query cost and $k^{\mathcal{O}(k)} \log^{\Theta(k-\alpha_k)} n$ multiplicative error. Moreover, by a standard colour-coding argument, it suffices to make this improvement when G is k -partite with vertex classes V_1, \dots, V_k known to the algorithm.

Oversimplifying a little, and assuming n is a power of two, the algorithm of [15] works by guessing a probability vector $(Q_1, \dots, Q_k) \in \{1, 1/2, 1/4, \dots, 1/n\}^k$. It then deletes vertices from V_1, \dots, V_k independently at random to form sets $\mathcal{X}_1, \dots, \mathcal{X}_k$, so that for all $v \in V_j$ we have $\mathbb{P}(v \in \mathcal{X}_j) = Q_j$. After doing so, in expectation, $Q_1 Q_2 \dots Q_k$ proportion of the edges of G remain in the induced k -partite subgraph $G[\mathcal{X}_1, \dots, \mathcal{X}_k]$. If $e(G) \ll 1/(Q_1 \dots Q_k)$, it is easy to show with a union bound that no edges are likely to remain. What is more surprising is that there exist q_1, \dots, q_k with $q_1 \dots q_k \approx 1/e(G)$ such that if $\vec{Q} = \vec{q}$, then at least one edge is likely to remain in $G[\mathcal{X}_1, \dots, \mathcal{X}_k]$. Thus the algorithm of [15] iterates over all $\log^{\Theta(k)} n$ possible values of \vec{Q} , querying cIND(G) on $\mathcal{X}_1, \dots, \mathcal{X}_k$ for each, and then outputs the least value m such that $e(G[\mathcal{X}_1, \dots, \mathcal{X}_k]) > 0$ for some Q_1, \dots, Q_k with $1/(Q_1 \dots Q_k) = m$.

Our algorithm improves on this idea as follows. First, [15] does not actually prove the existence of the vector \vec{q} described above — it relies on a coupling between the different guesses of \vec{Q} . We require not only the existence of \vec{q} but also a structural result which may be of independent interest. For all $I \subseteq [k]$ and all $\zeta \in (0, 1]$, we define an (I, ζ) -core to be an induced subgraph $H = G[Y_1, \dots, Y_k]$ of G such that:

- (i) H contains at least $k^{-\mathcal{O}(k)}$ proportion of the edges of G .
- (ii) For all $i \in I$, the set Y_i is very small, containing at most $2/\zeta$ vertices.
- (iii) For all $i \notin I$, every vertex of Y_i is contained in at most ζ proportion of edges in H .

As an example, the most extreme core is the *rooted star*: It consists of some vertices $r_i \in Y_i$ for all $i \in I$ and all k -partite edges e with $e \supseteq \{r_i : i \in I\}$. We prove in Lemma 56 that, for all $\zeta \in (0, 1]$, there is some $I \subseteq [k]$ such that G contains an (I, ζ) -core H .

Suppose for the moment that we are given the value of I , but not Y_1, \dots, Y_k . By property (i), it would then suffice to approximate $e(H)$ using $k^{\mathcal{O}(k)} \log^{\mathcal{O}(k-\alpha_k)} n$ query cost. If $|I| \geq \alpha_k$,

then we can adapt the idea of the algorithm of [15], but taking $Q_j = 1$ for all $i \notin I$ to use only $\log^{\mathcal{O}(k-\alpha_k)} n$ queries in total; intuitively, this is possible due to property (ii), which implies that this is the “correct” setting. We set this algorithm out as **CoarseLargeCore** in Section 4.1.3.

If instead $|I| \leq \alpha_k$, then we will exploit the fact that query cost decreases polynomially with instance size by breaking H into smaller instances. For all $i \notin I$, we randomly delete vertices from V_i with a carefully-chosen probability p . Property (iii), together with a martingale bound (see Lemma 59), guarantees that the number of edges in the resulting hypergraph G' will be concentrated around its expectation of $p^k e(G)$. If we had access to Y_1, \dots, Y_k , we could then intersect V_i with Y_i for all $i \in I$ to obtain a substantially smaller instance, whose edges we could count with cheap queries; we could then divide the result by p^k to obtain an estimate for $e(G)$. Unfortunately we do not have access to Y_1, \dots, Y_k , but we can still break G' into smaller sub-hypergraphs by applying colour-coding to the vertex sets V_i with $i \in I$, and as long as $|I| \leq \alpha_k$ this still gives enough of a saving in query cost to prove the result. We set this algorithm out as **CoarseSmallCore** in Section 4.1.2.

Now, we are not in fact given the value of I in the (I, ζ) -core of G . But both **CoarseLargeCore** and **CoarseSmallCore** fail gracefully if they are given an incorrect value of I , returning an underestimate of $e(G)$ rather than an overestimate. We can therefore simply iterate over all 2^k possible values of I , applying **CoarseLargeCore** or **CoarseSmallCore** as appropriate, and return the maximum resulting estimate of $e(G)$. This proves the result.

1.4.2 Colourful lower bound

We now discuss the proof of the lower bound of Theorem 2. Using the minimax principle, to prove the bound for randomised algorithms, it is enough to give a pair of random graphs G_1 and G_2 with $e(G_2) \gg e(G_1)$ and prove that no *deterministic* algorithm A can distinguish between G_1 and G_2 with constant probability and worst-case oracle cost as in the bound. We base these random graphs on the main bottleneck in the algorithm described in the previous section: the need to check all possible values of \mathcal{Q} in a k -partite k -hypergraph with an (I, ζ) -core where $|I| \approx k - \alpha_k$.

We take G_1 to be an Erdős-Rényi k -partite k -hypergraph with edge density $p := t^{-(k-\lfloor \alpha_k \rfloor - 2)/2}$. We take the vertex classes V_1, \dots, V_k of G_1 to have equal size t , so that $t = n/k$. We then define a random complete k -partite graph H as follows. We first define a random vector \vec{Q} of probabilities, then take binomially random subsets $\mathcal{X}_1, \dots, \mathcal{X}_{k-\lfloor \alpha_k \rfloor - 2}$ of $V_1, \dots, V_{k-\lfloor \alpha_k \rfloor - 2}$, with $\mathbb{P}(v \in \mathcal{X}_j) = Q_j$ for all $v \in V_j$. For $j \geq k - \lfloor \alpha_k \rfloor - 1$, we take $\mathcal{X}_j \subseteq V_j$ to contain a single uniformly random vertex. We then define H to be the complete k -partite graph with vertex classes $\mathcal{X}_1, \dots, \mathcal{X}_k$, and form $G_2 = G_1 \cup H$ by adding the edges of H to G_1 . We choose \mathcal{Q} in such a way that $Q_1 \cdot \dots \cdot Q_{k-\lfloor \alpha_k \rfloor - 2}$ is guaranteed to be a bit larger than $pt^{\lfloor \alpha_k \rfloor + 2}$, so that $\mathbb{E}(e(H)) \gg \mathbb{E}(e(G_1))$. Intuitively, this corresponds to the situation of a randomly planted (I, ζ) -core where $I = \{k - \lfloor \alpha_k \rfloor - 1, \dots, k\}$ — we will show that the algorithm A needs to essentially guess the value of \mathcal{Q} using expensive queries.

To show that a low-cost deterministic algorithm A cannot distinguish G_1 from G_2 , suppose for simplicity that A is *non-adaptive*, so that its future oracle queries cannot depend on the oracle’s past responses. In this setting, it suffices to bound the probability of a fixed query $S = (S_1, \dots, S_k)$ distinguishing G_1 from G_2 .

It is not hard to show that without loss of generality we can assume $S_i \subseteq V_i$ for all $i \in [k]$. If $|S_j| \ll t$ for some $j \geq k - \lfloor \alpha_k \rfloor - 1$, then with high probability S_j will not contain the single “root” vertex of \mathcal{X}_j , so $H[S_1, \dots, S_k]$ will contain no edges and S will not distinguish G_1 from G_2 . With some effort (a simple union bound does not suffice), this idea allows us

to essentially restrict our attention to large, expensive queries S . However, if $|S_1| \dots |S_k|$ is large, then with high probability $G_1[S_1, \dots, S_k]$ will contain an edge, so again S will not distinguish G_1 from $G_2 = G_1 \cup H$. With some more effort, this allows us to essentially restrict our attention to queries where for some possible value \vec{q} of $\vec{\mathcal{Q}}$ we have $|S_j| \approx 1/q_j$ for all $j \leq k - \lfloor \alpha \rfloor - 2$; we choose these possible values to be far enough apart that such a query is only likely to distinguish G_1 from G_2 if $\vec{\mathcal{Q}} = \vec{q}$. There are roughly $((\log n)/\log \log n)^{k - \lfloor \alpha \rfloor - 2}$ possible values of $\vec{\mathcal{Q}}$, so the result follows.

Of course, in our setting A may be adaptive, and this breaks the argument above. Since the query A makes depends on the results of past queries, we cannot bound the probability of a fixed query distinguishing G_1 from G_2 in isolation — we must condition on the results of past queries. This is not a small technical point — it is equivalent to allowing A to be adaptive in the first place. The most damaging implication is that we could have a query $S = (S_1, \dots, S_k)$ with $|S_1| \dots |S_k|$ very large but such that $G_1[S_1, \dots, S_k]$ contains no edges, because most of the potential edges have already been exposed as not existing in past queries. We are able to deal with this while preserving the spirit of the argument above, by arguing based on the number of unexposed edges rather than $|S_1| \dots |S_k|$, but it requires significantly more effort and a great deal of care.

1.4.3 Uncoloured upper bound

We now discuss the proof of the upper bound of [Theorem 1](#). We adapt a classic framework for approximate counting algorithms that originated in [\[36\]](#), and that was previously applied to edge counting in [\[14\]](#). We first observe that by using an algorithm from [\[28\]](#), we can enumerate the edges in an n -vertex k -hypergraph G with $2^{\mathcal{O}(k)} \log^{\mathcal{O}(1)} n \cdot e(G)$ queries to an independence oracle. Suppose we form an induced subgraph G_i of G by deleting vertices independently at random, keeping each vertex with probability 2^{-i} ; then in expectation, we have $e(G_i) = 2^{-ki} e(G)$. If $e(G_i)$ is small, and $e(G_i) \approx \mathbb{E}(e(G_i))$, then we can efficiently count the edges of G_i using [\[28\]](#) and then multiply by 2^{ki} to obtain an estimate of $e(G)$. We can then simply iterate over all i from 0 to $\log n$ and return an estimate based on the first i such that $e(G_i)$ is small enough for [\[28\]](#) to return a value quickly.

Of course in general we do not have $e(G_i) \approx \mathbb{E}(e(G_i))$! One issue arises if, for some $r \in [k-1]$, every edge of G contains a common size- r set R — a “root”. Then with probability $1 - 2^{-ri}$, at least one vertex in R will be deleted and G_i will contain no edges. We address this issue in the simplest way possible: by taking more samples. Roughly speaking, suppose we are given i , and that we already know (based on the failure of previous values of i to return a result) that $e(G) > n^{k-r-1}$ for some $0 \leq r \leq k-1$. This implies that G cannot have any “roots” of size greater than r . Rather than taking a single random subgraph G_i , we take $t_i \approx 2^{ri}$ independent copies of G_i and sum their edge counts using [\[28\]](#); thus if G does contain a size- r root, we are likely to include it in the vertex set of at least one sample. Writing Σ_i for the sum of their edge counts, we then return $\Sigma_i/(t_i 2^{-ik})$ if the enumeration succeeds.

The exact expression we use for t_i is more complicated than 2^{ri} , due to the possibility of multiple roots — see [Section 3.2.2](#) for a more detailed discussion — but the idea is the same. The proof that Σ_i is concentrated around its expectation is an (admittedly somewhat involved) application of Chebyshev’s inequality, in which the rooted “worst cases” correspond to terms in the variance of Σ_i . We consider it surprising and interesting that such a conceptually simple approach yields an optimal upper bound, and indeed gives us a strong hint as to how we should prove the lower bound of [Theorem 1](#).

1.4.4 Uncoloured lower bound

We finally discuss the proof of the lower bound of [Theorem 1](#). As in the colourful case, we apply the minimax principle, so we wish to define random k -hypergraphs G_1 and G_2 with $e(G_2) \gg e(G_1)$ which cannot be distinguished by a low-cost deterministic algorithm A .

Our construction of G_1 and G_2 is natural from the above discussion, and the special case with $k = 2$ and $\alpha_k = 0$ is very similar to the construction used in [\[13\]](#). We take G_1 to be an Erdős-Rényi k -hypergraph with edge probability roughly $k!/n^r$. We choose a random collection of size- r sets in $V(G_1)$ to be “roots”, with a large constant number of roots present in expectation, and we define a k -hypergraph H to include every possible edge containing at least one of these roots as a subset. We then define $G_2 := G_1 \cup H$.

Similarly to the colourful lower bound, in the non-adaptive setting, any fixed query S_i with $|S_i|$ large is likely to contain an edge of G_1 , and any fixed query with $|S_i|$ small is unlikely to contain a root and therefore unlikely to contain any edges of H ; in either case, the query does not distinguish G_1 from G_2 . Also as in the colourful case, generalising this argument from the non-adaptive setting to the adaptive setting requires a significant amount of care and effort.

Organization.

In [Section 2.1](#), we set out some standard conventions and our formal definitions of running time, oracle costs, and the most general cost functions to which our algorithmic results apply. We then recall some standard results from the literature (and folklore) in [Section 2.2](#), and in [Section 2.3](#) we apply a more recent result from [\[11\]](#) to allow us to quickly sample binomially random subsets. We collect all our IND-oracle results in [Section 3](#), proving various necessary properties of $g(k, \alpha_k)$ in [Section 3.1](#), the upper bound in [Section 3.2](#), and the lower bound in [Section 3.3](#). Finally, we collect all our cIND-oracle results in [Section 4](#), proving the upper bound in [Section 4.1](#) and the lower bound in [Section 4.3](#).

2 Preliminaries

2.1 Notation and definitions

2.1.1 Basic notation and conventions

We take \mathbb{N} to be the set of natural numbers not including zero. For all $n \in \mathbb{N}$, we write $[n] := \{1, \dots, n\}$. We write \log for the base-2 logarithm, and \ln for the base- e logarithm. Given two sets X and Y , we write $X \subset Y$ to mean that X is a proper subset of Y and $X \subseteq Y$ to mean that X is a (possibly improper) subset of Y .

For all $x \in \mathbb{R}$, we write $\lfloor x \rfloor := \lfloor x + 1/2 \rfloor$ for the value of x rounded to the nearest integer, rounding up in the case of a tie. For all $\varepsilon > 0$, we say that x is an ε -approximation to y if $|x - y| \leq \varepsilon y$.

For all sets S and all $r \in \mathbb{N}$, we write $S^{(r)}$ for the set of all r -element subsets of S . If G is a k -uniform k -hypergraph with $S \subseteq V(G)$, we write $G[S]$ for the subgraph induced by S ; thus $E(G[S]) \subseteq S^{(k)}$. Analogously, if $S = (S_1, \dots, S_k)$ is a k -tuple of disjoint sets, we write $G[S] = G[S_1, \dots, S_k]$ for the induced k -partite subgraph, which has vertex set $S_1 \cup \dots \cup S_k$ and edge set $\{e \in E(G) : |e \cap S_i| = 1 \text{ for all } i \in [k]\}$. We write $S^{(k)}$ for the set of all possible edges of $G[S]$, i.e. $\{\{s_1, \dots, s_k\} : s_i \in S_i \text{ for all } i\}$. If G and H are graphs on the same vertex set, we write $G \cup H$ for the graph on that vertex set with edge set $E(G) \cup E(H)$.

2.1.2 Oracle algorithms

Independence oracle.

Let G be a hypergraph with vertex set $[n]$ and m edges. Let $\text{IND}(G) \in \{0, 1\}^{2^{[n]}}$ be the string that satisfies the following for all $S \subseteq [n]$:

$$\text{IND}(G)_S = \begin{cases} 1 & \text{if } G[S] \text{ contains no edge;} \\ 0 & \text{otherwise.} \end{cases} \quad (2.1.1)$$

The bitstring $\text{IND}(G)$ is called the *independence oracle* of G . Let us write $G_{\text{IND}}(O)$ for the hypergraph belonging to the independence oracle O .

Colourful independence oracle.

The *colourful independence oracle* $\text{cIND}(G)$ of G is the bitstring that is defined as follows for all disjoint sets $S_1, \dots, S_k \subseteq [n]$:

$$\text{cIND}(G)_{S_1, \dots, S_k} = \begin{cases} 1 & \text{if } G \text{ contains no edge } e \in E(G) \text{ with } \forall i: |S_i \cap e| = 1; \\ 0 & \text{otherwise.} \end{cases} \quad (2.1.2)$$

We write $G_{\text{cIND}}(O)$ for the hypergraph belonging to the colourful independence oracle O .

Oracle algorithm.

In this paper, an *oracle algorithm* is an algorithm with access to an oracle that represents the input n -vertex k -hypergraph implicitly, either as an independence oracle or as a colourful independence oracle. We assume that the numbers n and k are always given explicitly as input and thus known to the algorithm at run-time; recall that the graph's vertex set is always $[n]$, so this is also known. We consider IND -oracle algorithms in [Section 3](#) and cIND -oracle algorithms in [Section 4](#), and we strictly distinguish between the worst-case running time of the algorithm (which does not include oracle costs) and the worst-case oracle cost of the queries (which does not include the running time of the algorithm). Given an oracle O (such as $\text{IND}(G)$ for some n -vertex k -hypergraph G) we write $A(O, x)$ for the output produced by A when supplied with O and with additional input x . While n and k are always included as part of x , we typically do not write these out. The algorithm can prepare a query by writing a set S or a tuple (S_1, \dots, S_k) in the canonical encoding as an indicator string $q \in \{0, \dots, k\}^n$ to a special query array. It can then issue the oracle query via a primitive `query`(q) and receive the answer 0 or 1.

Induced subgraphs.

Observe that given $\text{IND}(G)$, it is trivial to simulate $\text{IND}(G[X])$ for all $X \subseteq [n]$, as we have $\text{IND}(G[X])_S = \text{IND}(G)_S$ for all $S \subseteq X$. We will frequently use this fact implicitly to pass induced subgraphs of G as arguments to subroutines without incurring overhead. To this end, we write $A(\text{IND}(G[X]), x)$ as a shorthand for $A'(\text{IND}(G), X, x)$, where A' is the oracle algorithm that runs A and simulates oracle queries to $\text{IND}(G[X])$ as just discussed using X and $\text{IND}(G)$. Likewise, given $\text{cIND}(G)$, it is trivial to simulate $\text{cIND}(G[S_1, \dots, S_k])$ for all disjoint $S_1, \dots, S_k \subseteq [n]$.

Randomness.

To model *randomised algorithms*, we augment the RAM-model with a primitive `rand`(R) that returns a uniformly random element from the set $\{1, \dots, R\}$ for any given $R \leq \text{poly}(n)$. We view a randomised algorithm as a discrete probability distribution \mathcal{A} over a set $\text{supp}(\mathcal{A})$ of deterministic algorithms based on the results of `rand`.

Running time.

We measure the *running time* in the standard RAM-model of computation with $\Theta(\log n)$ bits per word and the usual arithmetic and logical operations on these words. Writing $T(A, G, x)$ for the running time of $A(\text{IND}(G), x)$, the *worst-case running time* of a randomised algorithm \mathcal{A} is defined as the function $x \mapsto \max_G \sup_{A \in \text{supp}(\mathcal{A})} T(A, G, x)$, where G ranges over all n -vertex k -hypergraphs with n and k as specified in the input x . Similarly, for a randomised algorithm \mathcal{A} , the *worst-case expected running time* is defined as the function $x \mapsto \max_G (\mathbb{E}_{A \sim \mathcal{A}} T(A, G, x))$, where again the maximum is taken over all n -vertex k -hypergraphs G .

Oracle cost.

We measure the oracle cost by a *cost function* $\text{cost} = \{\text{cost}_k : k \geq 2\}$, where $\text{cost}_k : (0, \infty) \rightarrow (0, \infty)$. We will typically require cost to be regularly-varying with parameter k (see [Section 2.1.3](#)). In a k -uniform hypergraph G , the cost of an `IND`-oracle query to a set $S \subseteq [n]$ is given by $\text{cost}(S) := \text{cost}_k(|S|)$; note that this query cost depends only on the size of S . By convention, we set $\text{cost}_k(0) := 0$ for all k as empty queries cannot provide any information about the graph. Similarly, the cost of a `cIND`-oracle query to a tuple (S_1, \dots, S_k) is given by $\text{cost}(S_1, \dots, S_k) := \text{cost}_k(|S_1 \cup \dots \cup S_k|)$.

Let A be a deterministic oracle algorithm, and let X be the set of possible (non-oracle) inputs to A ; recall that this always includes n and k . Given an n -vertex k -uniform hypergraph G and $x \in X$, let $S_1(G, x), \dots, S_{t_{G,x}}(G, x)$ be the sequence of queries that A makes when supplied with input x and an oracle for G . Then for all n -vertex k -uniform hypergraphs G and all $x \in X$, the *oracle cost* of A on G (with respect to cost) is defined as $\text{cost}(A, G, x) = \sum_{i=1}^{t_{G,x}} \text{cost}_k(S_i(G, x))$. The oracle cost of a randomised oracle algorithm \mathcal{A} on G and x is denoted by $\text{cost}(\mathcal{A}, G, x)$, and is the random variable $\text{cost}(A, G, x)$ where $A \sim \mathcal{A}$.

The *worst-case oracle cost* of a randomised oracle algorithm \mathcal{A} (with respect to cost) is defined as the function $x \mapsto \max_G (\sup_{A \in \text{supp}(\mathcal{A})} \text{cost}(A, G, x))$; here the maximum is taken over all n -vertex k -hypergraphs G , where n and k are as defined in the input x . Similarly, the *worst-case expected oracle cost* of \mathcal{A} is defined as the function $x \mapsto \max_G (\mathbb{E}_{A \sim \mathcal{A}} (\text{cost}(A, G, x)))$, where again the maximum is taken over all n -vertex k -hypergraphs G .

2.1.3 Requirements on cost functions for upper bounds

In the introduction, we focused on polynomial cost functions of the form $\text{cost}_k(S) = |S|^{\alpha_k}$ for some $\alpha_k \in [0, k]$, and these functions are easy to work with. However, even subpolynomial improvements to an algorithm's running time can be of interest and by considering more general cost functions we can lift these improvements from decision algorithms to approximate counting algorithms. To take a well-known example, in the negative-weight triangle problem, we are given an n -vertex edge-weighted graph G and asked to determine whether it contains a triangle of negative total weight; this problem is equivalent to a set of other problems under subcubic reductions, including APSP [38]. The naive $\Theta(n^3)$ -time algorithm can be improved

by a subpolynomial factor of $e^{\Theta(\sqrt{\log n})}$ [37], but as stated in the introduction our result would not lift this improvement from decision to approximate counting — we would need to take $k = 3$ and $\text{cost}(n) = n^3/e^{\Theta(\sqrt{\log n})}$. (For clarity, in this specific case the problem is equivalent to its colourful version and a fine-grained reduction is already known [14, 15].)

While we cannot hope to say anything meaningful in the algorithmic setting with a fully general cost function, our results do extend to all cost functions which might reasonably arise as running times. A natural first attempt to formalise what we mean by “reasonable” would be to consider cost functions of the form $\text{cost}_k(n) = n^{\alpha_k + o(1)}$ as $n \rightarrow \infty$. Unfortunately, such cost functions can still have a pathological property which makes a fine-grained reduction almost impossible: the $o(1)$ term might vary wildly between different values of n . For example, if we take $\text{cost}_k(n) = n^{\alpha_k + \sin(\pi n/2)/\sqrt{\log(n)}}$, then we have $\text{cost}_k(n) = n^{\alpha_k + o(1)}$, but $\text{cost}_k(2n)/\text{cost}_k(n)$ could be $\omega(\text{polylog}(n))$, $\Theta(1)$ or $o(1/\text{polylog}(n))$ depending on whether n is congruent to 3, 2 or 1 modulo 4 (respectively). It is even possible to construct a similar example which is monotonically increasing. We therefore require something slightly stronger, borrowing a standard notion from the probability literature for distributions which are “almost power laws”.

► **Definition 7.** A function $f: \mathbb{R} \rightarrow \mathbb{R}$ is regularly-varying if, for all fixed $a > 0$,

$$\lim_{x \rightarrow \infty} |f(ax)/f(x)| \in (0, \infty).$$

Observe that any cost function likely to arise as a running time is regularly-varying under this definition. We will use the following standard facts about regularly-varying functions.

► **Lemma 8.** Let f be a regularly-varying function. Then there exists a unique $\alpha \in \mathbb{R}$, called the index of f , with the following properties.

- (i) For all fixed $A > 0$, $\lim_{x \rightarrow \infty} (f(Ax)/f(x)) = A^\alpha$. Moreover, for all closed intervals $I \subseteq \mathbb{R}$, this limit is uniform over all $A \in I$.
- (ii) For all $\delta > 0$, there exists x_0 such that for all $x \geq x_0$ and all $A_x \geq 1$,

$$A_x^{\alpha-\delta} \leq f(A_x x)/f(x) \leq A_x^{\alpha+\delta}.$$

- (iii) $f(x) = x^{\alpha+o(1)}$ as $x \rightarrow \infty$.

- (iv) If $\alpha > 0$, then there exists x_0 such that f is strictly increasing on $[x_0, \infty)$.

Proof. Part (i) is proved in e.g. Feller [17, Chapter VIII.8 Lemmas 1–2]. We now prove part (ii). Let $A_x \geq 1$. By part (i), there exists x_0 such that

$$\text{for all } x \geq x_0 \text{ and all } A \in [1, 2], A^{\alpha-\delta} \leq f(Ax)/f(x) \leq A^{\alpha+\delta}. \quad (2.1.3)$$

Suppose $x \geq x_0$, and let k_x be sufficiently large such that $A_x^{1/k_x} \in [1, 2]$ holds. Then breaking $f(A_x n)/f(n)$ into a telescoping product and applying (2.1.3) to each term yields

$$\frac{f(A_x n)}{f(n)} = \prod_{i=1}^{k_x} \frac{f(A_x^{i/k_x})}{f(A_x^{(i-1)/k_x})} \geq \prod_{i=1}^{k_x} A_x^{(\alpha-\delta)/k_x} = A_x^{\alpha-\delta},$$

as required. Similarly, $f(A_x n)/f(n) \leq A_x^{\alpha+\delta}$ as required. Part (iii) follows immediately on taking $x = 1$ in part (ii). To prove part (iv), take $x, y \in (0, \infty)$ with $x < y$ and observe from part (ii) that when x and y are sufficiently large, $\text{cost}(y) \geq \text{cost}(x)(y/x)^{\alpha/2} > \text{cost}(x)$. ◀

We will only need our cost functions to vary regularly in n , not k ; to facilitate this, we bring in the following standard result which allows us to “confine the regular variation to the $n^{o(1)}$ term”.

► **Definition 9.** A regularly-varying function with index 0 is called a slowly-varying function.

► **Lemma 10.** Let f be a regularly-varying function with index α . Then we have $f(x) = x^\alpha \sigma(x)$ for some slowly-varying function σ .

Proof. This follows easily from Lemma 8; see Feller [17, Chapter VIII.8 (8.5)–(8.6)]. ◀

We are now ready to state the technical requirements on our cost functions.

► **Definition 11.** For each $k \geq 2$, let $\text{cost}_k: (0, \infty) \rightarrow (0, \infty)$. We say that $\text{cost} = \{\text{cost}_k: k \geq 2\}$ is a regularly-varying parameterised cost function if $\text{cost}_k(x) \leq x^k$ for all k and x , and there exists a slowly-varying function $\sigma: (0, \infty) \rightarrow (0, \infty)$ and a map $k \mapsto \alpha_k$ satisfying the following properties:

- (i) for all k and x , $\text{cost}_k(x) = x^{\alpha_k} \sigma(x)$;
- (ii) for all k , $\alpha_k \in [0, k]$;
- (iii) for all k and x , $\text{cost}_k(x) \leq x^k$;
- (iv) either $\liminf_{k \rightarrow \infty} \alpha_k > 0$ or there exists x_0 such that for all k , cost_k is non-decreasing on (x_0, ∞) ;
- (v) there is an algorithm to compute $\lfloor \alpha_k \rfloor$ in time $\mathcal{O}(k^{9k})$.

We say that k is the parameter of cost , α is the index of cost , and σ is the slowly-varying component of cost .

Point (i) is the main restriction, and the one we have been discussing until now. Points (ii) and (iii) have already been discussed in the introduction. In the colourful case we will need to compute $\lfloor \alpha_k \rfloor$ in order to know which subroutines to use, so point (v) avoids an additive term in the running time. Finally, point (iv) is a technical convenience which (together with point (i)) guarantees monotonicity, as we show in Lemma 12 below. Requiring point (iv) is unlikely to affect applications of our results — typically such applications would either satisfy $\alpha_k = 0$ (as we are concerned only with query count and not with query cost) or $\alpha_k \geq 1$ (as the decision algorithm used to simulate the oracle needs to read the entire input).

► **Lemma 12.** Let cost be a regularly-varying parameterised cost function with parameter k , index α and slowly-varying component σ . Then there exists $x_0 \in (0, \infty)$ such that for all k , cost_k is non-decreasing on $[x_0, \infty)$.

Proof. If $\liminf_{k \rightarrow \infty} \alpha_k = 0$ then this is immediate from Definition 11(iv), so suppose instead $\liminf_{k \rightarrow \infty} \alpha_k = \delta > 0$. Then the function $x \mapsto x^\delta \sigma(x)$ is regularly-varying, and hence by Lemma 8(iv) there exists x_0 such that for all $y > x \geq x_0$ we have $y^\delta \sigma(y) > x^\delta \sigma(x)$. It follows that for all k ,

$$\frac{\text{cost}_k(y)}{\text{cost}_k(x)} = (y/x)^{\alpha_k} \frac{\sigma(y)}{\sigma(x)} \geq \frac{y^\delta \sigma(y)}{x^\delta \sigma(x)} > 1$$

as required. ◀

2.2 Collected standard results

2.2.1 Probabilistic results

We will need the following two standard Chernoff bounds.

► **Lemma 13** (Corollary 2.3 of Janson, Łuczak and Rucinski [22]). Let X be a binomial random variable, and let $0 < \delta < 3/2$. Then $\mathbb{P}(|X - \mathbb{E}(X)| \geq \delta \mathbb{E}(X)) \leq 2e^{-\delta^2 \mathbb{E}(X)/3}$.

► **Lemma 14** (Corollary 2.4 of Janson, Łuczak and Rucinski [22]). *Let X be a binomial random variable, and let $z \geq 7 \cdot \mathbb{E}(X)$. Then $\mathbb{P}(X \geq z) \leq e^{-z}$.*

We will also make use of the following form of the FKG inequality.

► **Lemma 15** (Theorem 6.3.2 of Alon and Spencer [2]). *Let X_1, \dots, X_n be independent Bernoulli variables, let \mathcal{A} and \mathcal{B} be events determined by these variables, let $1_{\mathcal{A}}$ be the indicator function of \mathcal{A} and let $1_{\mathcal{B}}$ be the indicator function of \mathcal{B} . If $1_{\mathcal{A}}$ and $1_{\mathcal{B}}$ are either both increasing or both decreasing as functions of X_1, \dots, X_n , then $\mathbb{P}(\mathcal{A} \cap \mathcal{B}) \geq \mathbb{P}(\mathcal{A}) \mathbb{P}(\mathcal{B})$. If $1_{\mathcal{A}}$ is increasing and $1_{\mathcal{B}}$ is decreasing or vice versa, then $\mathbb{P}(\mathcal{A} \cap \mathcal{B}) \leq \mathbb{P}(\mathcal{A}) \mathbb{P}(\mathcal{B})$.*

The following probability bound is folklore.

► **Lemma 16.** *Let t be an integer, let Z_1, \dots, Z_t be discrete random variables, let $\mathcal{E}_1, \dots, \mathcal{E}_t$ be events such that \mathcal{E}_i is a function of Z_1, \dots, Z_i , and let $\mathcal{E} = \mathcal{E}_1 \cup \dots \cup \mathcal{E}_t$. For all i , let \mathcal{Z}_i be the set of possible values of Z_1, \dots, Z_i under which $\mathcal{E}_1, \dots, \mathcal{E}_i$ do not occur, that is,*

$$\mathcal{Z}_i = \left\{ (z_1, \dots, z_i) : \mathbb{P}(\overline{\mathcal{E}_1}, \dots, \overline{\mathcal{E}_i} \mid (Z_1, \dots, Z_i) = (z_1, \dots, z_i)) = 1 \right\}.$$

Suppose that for all i , every $(z_1, \dots, z_i) \in \mathcal{Z}_i$ is a prefix to some $(z_1, \dots, z_t) \in \mathcal{Z}_t$. Then

$$\mathbb{P}(\mathcal{E}) \leq \max_{(z_1, \dots, z_{t-1}) \in \mathcal{Z}_{t-1}} \sum_{i=1}^t \mathbb{P}(\mathcal{E}_i \mid (Z_1, \dots, Z_{i-1}) = (z_1, \dots, z_{i-1})). \quad (2.2.1)$$

We will use Lemma 16 when proving lower bounds. In doing so, we will take Z_1, \dots, Z_t to be pairs of query results from running a deterministic algorithm on two random inputs, \mathcal{E}_i to be the event that we distinguish the two inputs on the i^{th} query, and \mathcal{E} to be the event that we distinguish the two inputs at all. Note the order of the maximum and the sum in (2.2.1); in a simple union bound over $\mathcal{E}_1, \dots, \mathcal{E}_t$, they would occur in the opposite order for a weaker result.

Proof. Consider the random variable

$$Y := \sum_{i=1}^t \mathbb{P}_{Z_i}(\overline{\mathcal{E}_1}, \dots, \overline{\mathcal{E}_{i-1}}, \mathcal{E}_i \mid Z_1, \dots, Z_{i-1}).$$

Note that the i^{th} term in this sum is a random variable that is a function of Z_1, \dots, Z_{i-1} . We first bound Y above. Deterministically, on exposing the values of Z_1, \dots, Z_t we obtain

$$Y \leq \max_{z_1, \dots, z_{t-1}} \sum_{i=1}^t \mathbb{P}_{Z_i}(\overline{\mathcal{E}_1}, \dots, \overline{\mathcal{E}_{i-1}}, \mathcal{E}_i \mid (Z_1, \dots, Z_{i-1}) = (z_1, \dots, z_{i-1})).$$

If $(z_1, \dots, z_{t-1}) \notin \mathcal{Z}_{t-1}$, then let $k = \min\{\ell \leq t-1 : (z_1, \dots, z_\ell) \notin \mathcal{Z}_\ell\}$. Thus $(z_1, \dots, z_k) \notin \mathcal{Z}_k$, but (if $k > 1$) we have $(z_1, \dots, z_{k-1}) \in \mathcal{Z}_{k-1}$; it follows that $\overline{\mathcal{E}_1}, \dots, \overline{\mathcal{E}_{k-1}}$ and \mathcal{E}_k all occur. The last $t-k$ probabilities in the corresponding sum are therefore zero (even if $k=1$), and we have

$$\begin{aligned} \sum_{i=1}^t \mathbb{P}_{Z_i}(\overline{\mathcal{E}_1}, \dots, \overline{\mathcal{E}_{i-1}}, \mathcal{E}_i \mid (Z_1, \dots, Z_{i-1}) = (z_1, \dots, z_{i-1})) \\ = \sum_{i=1}^k \mathbb{P}_{Z_i}(\overline{\mathcal{E}_1}, \dots, \overline{\mathcal{E}_{i-1}}, \mathcal{E}_i \mid (Z_1, \dots, Z_{i-1}) = (z_1, \dots, z_{i-1})). \end{aligned}$$

It follows that

$$Y \leq \max_{k \in [t]} \max_{(z_1, \dots, z_{k-1}) \in \mathcal{Z}_{k-1}} \sum_{i=1}^k \mathbb{P}_{\mathcal{Z}_i} \left(\overline{\mathcal{E}_1}, \dots, \overline{\mathcal{E}_{i-1}}, \mathcal{E}_i \mid (Z_1, \dots, Z_{i-1}) = (z_1, \dots, z_{i-1}) \right),$$

as we are maximising over the same terms — each $(z_1, \dots, z_{t-1}) \notin \mathcal{Z}_{t-1}$ corresponds to a shorter $(z_1, \dots, z_{k-1}) \in \mathcal{Z}_{k-1}$. Moreover, since every vector in \mathcal{Z}_{k-1} is a prefix to some vector in \mathcal{Z}_{t-1} by hypothesis, the first maximum must be attained at $k = t$, and so

$$Y \leq \max_{(z_1, \dots, z_{t-1}) \in \mathcal{Z}_{t-1}} \sum_{i=1}^t \mathbb{P}_{\mathcal{Z}_i} \left(\overline{\mathcal{E}_1}, \dots, \overline{\mathcal{E}_{i-1}}, \mathcal{E}_i \mid (Z_1, \dots, Z_{i-1}) = (z_1, \dots, z_{i-1}) \right). \quad (2.2.2)$$

We now calculate $\mathbb{E}(Y)$. By linearity of expectation and since each \mathcal{E}_i is a function of Z_1, \dots, Z_i , we have

$$\begin{aligned} \mathbb{E}(Y) &= \sum_{i=1}^t \mathbb{E}_{Z_1, \dots, Z_{i-1}} \left(\mathbb{P}_{\mathcal{Z}_i} \left(\overline{\mathcal{E}_1}, \dots, \overline{\mathcal{E}_{i-1}}, \mathcal{E}_i \mid Z_1, \dots, Z_{i-1} \right) \right) \\ &= \sum_{i=1}^t \mathbb{P}_{Z_1, \dots, Z_i} \left(\overline{\mathcal{E}_1}, \dots, \overline{\mathcal{E}_{i-1}}, \mathcal{E}_i \right) = \mathbb{P}(\mathcal{E}). \end{aligned}$$

Since $\mathbb{E}(Y)$ is bounded above by the maximum possible value of Y , the result follows from (2.2.2). \blacktriangleleft

2.2.2 Algorithmic results

In order to prove lower bounds on the cost of randomised oracle algorithms we will apply the minimax principle, a standard tool from decision tree complexity. There are many possible statements of this principle (see [39, Theorem 3] for the original one by Yao); for convenience, we provide a statement matched to our setting along with a self-contained proof.

► Theorem 17. *Let $n, k \in \mathbb{N}$ with $k \geq 2$. Let \mathcal{A} be a randomised oracle algorithm whose possible non-oracle inputs are drawn from a set X and whose possible outputs lie in a set Σ . Let $x \in X$, and let n and k be as determined by x . Let \mathcal{G} be the set of all k -hypergraphs on $[n]$. If \mathcal{D} is any probability distribution on \mathcal{G} , and F is any set of functions $\mathcal{G} \times X \rightarrow \Sigma$ with $\mathbb{P}_{A \sim \mathcal{A}}[A \in F] \geq p$, then*

$$\max_{G \in \mathcal{G}} \mathbb{E}_{A \sim \mathcal{A}} [\text{cost}(A, G, x)] \geq p \inf_{A \in F} \mathbb{E}_{G \sim \mathcal{D}} [\text{cost}(A, G, x)]. \quad (2.2.3)$$

The expectation $\mathbb{E}_{A \sim \mathcal{A}}[\text{cost}(A, G, x)]$ on the left side of Equation (2.2.3) is the expected cost of running a randomised algorithm \mathcal{A} on the graph G ; thus the left side of Equation (2.2.3) is the worst-case expected oracle cost of \mathcal{A} over all possible inputs G (with non-oracle input x). The set F will be application-dependent, but will typically contain all deterministic algorithms that with suitably high probability exhibit the “correct” behaviour on the specific input distribution \mathcal{D} — for example, returning an accurate approximation to the number of edges in the input graph. In our applications, the requirement $\mathbb{P}_{A \sim \mathcal{A}}[A \in F] \geq p$ will be immediate from the stated properties of \mathcal{A} . The point of Theorem 17, then, is that we can lower-bound the cost of an arbitrary *randomised* algorithm \mathcal{A} on a worst-case deterministic graph $G \in \mathcal{G}$ by instead lower-bounding the cost of any *deterministic* algorithm A on a randomised input distribution \mathcal{D} of our choice.

Proof. We derive the claim as follows:

$$\begin{aligned}
\max_{G \in \mathcal{G}} \mathbb{E}_{A \sim \mathcal{A}} [\text{cost}(A, G, x)] &\geq \mathbb{E}_{G \sim \mathcal{D}} \mathbb{E}_{A \sim \mathcal{A}} [\text{cost}(A, G, x)] = \mathbb{E}_{A \sim \mathcal{A}} \mathbb{E}_{G \sim \mathcal{D}} [\text{cost}(A, G, x)] \\
&\geq \mathbb{E}_{A \sim \mathcal{A}} \left[\mathbb{E}_{x \sim \mathcal{X}} [\text{cost}(A, G, x)] \mid A \in F \right] \cdot \mathbb{P}_{A \sim \mathcal{A}} [A \in F] \\
&\geq \frac{1}{2} \inf_{A \in F} \left[\mathbb{E}_{x \sim \mathcal{X}} [\text{cost}(A, G, x)] \right].
\end{aligned}$$

The first and last inequalities are trivial, and the second inequality follows by conditioning and dropping the terms for $A \notin F$, which is correct because the cost is non-negative. \blacktriangleleft

Finally, we set out two lemmas for passing between different performance guarantees on randomised algorithms. The ideas behind these lemmas (independent repetition and median boosting) are very standard, but we set them out in detail because we are working in the word-RAM model (see [Section 2.1.2](#)) and so we may not be able to efficiently compute our own stated upper bounds on oracle cost; thus there is more potential than usual for subtle errors.

We use the following standard lemma to pass from probabilistic guarantees on running time and oracle cost to deterministic guarantees. The lemma assumes an implicit problem definition in the form of a relation between inputs and outputs, modelling which input-output pairs are considered to be correct. In our application, an input-output pair $((G, x), A(\text{IND}(G), x))$ is considered correct if $A(\text{IND}(G), x)$ is an ε -approximation to $e(G)$.

► **Lemma 18.** *Let $\text{cost} = \{\text{cost}_k : k \geq 2\}$ be a cost function as in [Section 2.1.2](#). Let \mathcal{A} be a randomised oracle algorithm whose possible non-oracle inputs are drawn from a set X . Let $T_{\text{orig}}, C_{\text{orig}}, T_{\text{totals}}^{\text{comp}}, T_{\text{cost}}^{\text{comp}} : X \rightarrow (0, \infty)$, and let $\delta > 0$ be a constant. Suppose that:*

- *given any possible oracle input O and non-oracle input $x \in X$, with probability at least $1 - \delta$, $\mathcal{A}(O, x)$ has the correct input-output behaviour, runs in time $\mathcal{O}(T_{\text{orig}}(x))$, and has oracle cost $\mathcal{O}(C_{\text{orig}}(x))$;*
- *for all $x \in X$, $T_{\text{orig}}(x)$ and $C_{\text{orig}}(x)$ can be computed in time $\mathcal{O}(T_{\text{totals}}^{\text{comp}}(x))$; and*
- *for all $x \in X$, writing k_x and n_x for the values of k and n specified by x , for all $n \leq n_x$, $\text{cost}_{k_x}(n)$ can be computed in time $\mathcal{O}(T_{\text{cost}}^{\text{comp}}(x))$.*

Then there is a randomised oracle algorithm \mathcal{A}' such that:

- (i) \mathcal{A}' has the same inputs as \mathcal{A} together with a rational $\gamma \in (0, 1)$;
- (ii) \mathcal{A}' has worst-case running time $\mathcal{O}(T_{\text{totals}}^{\text{comp}}(x) + \log(1/\gamma) \cdot T_{\text{orig}}(x) \cdot T_{\text{cost}}^{\text{comp}}(x))$;
- (iii) \mathcal{A}' has worst-case oracle cost $\mathcal{O}(\log(1/\gamma) \cdot C_{\text{orig}}(x))$;
- (iv) \mathcal{A}' exhibits the same correct input-output behaviour as \mathcal{A} with probability $1 - \gamma - \delta$.

Proof. The algorithm $\mathcal{A}'(O, x, \gamma)$ works as follows:

- Compute $T_{\text{orig}}(x)$ and $C_{\text{orig}}(x)$.
- Repeat at most $\lceil \log_\delta(\gamma) \rceil$ times:
 - Run $\mathcal{A}(O, x)$ and keep track of the running time incurred. Compute the cost of each invocation of the oracle before it happens, and keep track of the total cost incurred.
 - If the total running time exceeds $T_{\text{orig}}(x)$ (not counting overhead for tracking the running time) or if the total oracle cost would exceed $C_{\text{orig}}(x)$ on the next oracle query, abort this run.
 - If $\mathcal{A}(O, x)$ halts within these resource constraints, return its output.
- Return the error RTE ('running time exceeded').

Part (i) is immediate. Observe that \mathcal{A}' runs $\mathcal{A}(O, x)$ at most $\mathcal{O}(\log(1/\gamma))$ times, each time incurring $\mathcal{O}(T_{\text{orig}}(x))$ running time and $\mathcal{O}(C_{\text{orig}}(x))$ oracle cost. Moreover, since each oracle query takes $\Theta(1)$ time, each run of \mathcal{A} makes $\mathcal{O}(T_{\text{orig}}(x))$ oracle queries; thus we spend $\mathcal{O}(T_{\text{orig}}(x) \cdot T_{\text{cost}}^{\text{comp}}(x))$ time per run on keeping track of the total running time and oracle cost. Finally, we spend $\mathcal{O}(T_{\text{totals}}^{\text{comp}}(x))$ time computing $T_{\text{orig}}(x)$ and $C_{\text{orig}}(x)$. Thus (ii) and (iii) follow.

Let \mathcal{E} be the event that at least one run of \mathcal{A} succeeds within the resource constraints. Since the runs are independent, we have $\mathbb{P}(\mathcal{E}) \geq 1 - \delta^{\lceil \log_{\delta}(\gamma) \rceil} \geq 1 - \gamma$. Moreover, conditioned on a given run being within the resource constraints, this run exhibits the correct input-output behaviour for \mathcal{A} with probability at least $1 - \delta$; thus conditioned on \mathcal{E} , the algorithm \mathcal{A}' also exhibits the correct input-output behaviour with probability at least $1 - \delta$. By a union bound, (iv) follows. \blacktriangleleft

We also use the following standard lemma to boost the success probability of approximate counting by taking the median output among independent repetitions.

► **Lemma 19.** *Let \mathcal{A} be a randomised oracle algorithm. Suppose that given non-oracle input x , the algorithm \mathcal{A} has worst-case running time $\mathcal{O}(T(x))$ and worst-case oracle-cost $\mathcal{O}(C(x))$. Suppose further that for some constant $\delta < 1/2$, for all possible oracle inputs O and non-oracle inputs x , with probability at least $1 - \delta$, the algorithm call $\mathcal{A}(O, x)$ outputs a real number which lies in some possibly-unbounded interval $I_{O,x}$. Then there is a randomised oracle algorithm \mathcal{A}' such that:*

- (i) \mathcal{A}' has the same inputs as \mathcal{A} together with a rational $\gamma \in (0, 1)$;
- (ii) \mathcal{A}' has worst-case running time $\mathcal{O}(\log(1/\gamma) \cdot T(x))$;
- (iii) \mathcal{A}' has worst-case oracle cost $\mathcal{O}(\log(1/\gamma) \cdot C(x))$;
- (iv) For all possible inputs (O, x) , with probability at least $1 - \gamma$, the algorithm call $\mathcal{A}'(O, x)$ outputs a real number which lies in $I_{O,x}$.

Proof. Write $\xi = 1 - 1/(2(1 - \delta))$, and note that $\xi \in (0, 1)$ since $\delta \in [0, 1/2)$. Given oracle input O and non-oracle inputs $x \in X$ and $\gamma \in (0, 1)$, the algorithm \mathcal{A}' runs $\mathcal{A}(O, x)$ a total of $N := \lceil 6 \ln(2/\gamma)/\xi^2 \rceil$ times using independent randomness and returns the median of the outputs, treating any non-numerical outputs of \mathcal{A} as -1 . Parts (i)–(iii) are immediate. Moreover, let M be the number of outputs of \mathcal{A} which lie in $I_{O,x}$. If $M > N/2$, then the output of \mathcal{A}' also lies in $I_{O,x}$. Since $\mathbb{E}(M) \geq (1 - \delta)N$, it follows that

$$\begin{aligned} \mathbb{P}(\mathcal{A}'(O, x, \gamma) \in I_{O,x}) &\geq \mathbb{P}(M \geq N/2) \geq \mathbb{P}(M \geq \mathbb{E}(M)/(2(1 - \delta))) \\ &\geq \mathbb{P}(|M - \mathbb{E}(M)| \geq \xi \mathbb{E}(M)) \end{aligned}$$

Since M is a binomial variable, by the Chernoff bound of [Lemma 13](#) it follows that

$$\mathbb{P}(\mathcal{A}'(O, x, \gamma) \in I_{O,x}) \geq 1 - 2e^{-\xi^2 \mathbb{E}(M)/3} \leq 1 - 2e^{-\xi^2 N/6} \geq 1 - \gamma,$$

as required. \blacktriangleleft

2.2.3 Algebraic results

In proving our lower bounds, we will lower-bound the cost of a deterministic approximate counting algorithm in terms of the sizes of the queries it uses; we will then need to maximise this bound over all possible sets of query sizes to put our lower bound into closed form. A key part of this argument will be [Corollary 22](#), stated below. In order to prove this [Corollary 22](#), we first state a standard generalisation of Jensen's inequality due to Karamata.

► **Definition 20.** Let t be a positive integer and let $\vec{x}, \vec{y} \in \mathbb{R}^t$. We say that \vec{y} majorises \vec{x} and write $\vec{y} \succ \vec{x}$ or $\vec{x} \prec \vec{y}$ if

- (i) $x_1 \geq \dots \geq x_t$ and $y_1 \geq \dots \geq y_t$;
- (ii) $x_1 + \dots + x_t = y_1 + \dots + y_t$; and
- (iii) for all $m \in [t-1]$, $x_1 + \dots + x_m \geq y_1 + \dots + y_m$.

► **Lemma 21** (Karamata's inequality, eg. see [32, Theorem 12.2]). Let I be an interval in \mathbb{R} and let $\phi: I \rightarrow \mathbb{R}$ be a convex function. Let t be a positive integer and let $\vec{x}, \vec{y} \in \mathbb{R}^t$. If $\vec{x} \prec \vec{y}$, then

$$\sum_{i=1}^t \phi(x_i) \leq \sum_{i=1}^t \phi(y_i).$$

► **Corollary 22.** Let $r \geq \alpha \geq 0$. Let $c > 0$, let t be a positive integer and let $\vec{s} \in [0, c]^t$. Let $W > 0$, and suppose that $\sum_{i=1}^t s_i^\alpha \leq W$. Then we have

$$\sum_{i=1}^t s_i^r \leq W c^{r-\alpha}.$$

Proof. First observe that if $\alpha = 0$ then $t \leq W$ by hypothesis; thus $\sum_i s_i^r \leq t c^r \leq W c^{r-\alpha}$ as required. For the rest of the proof, we assume that $\alpha > 0$. Further, without loss of generality, we may assume that $s_1 \geq \dots \geq s_t$ and that $\sum_i s_i^\alpha = W$ (by increasing \vec{s} and t if necessary).

We apply Lemma 21, taking $\phi(x) = x^{r/\alpha}$, $x_i = s_i^\alpha$ for all $i \in [t]$, and

$$y_i = \begin{cases} c^\alpha & \text{if } i \leq \lfloor W/c^\alpha \rfloor, \\ W - \lfloor W/c^\alpha \rfloor c^\alpha & \text{if } i = \lfloor W/c^\alpha \rfloor + 1, \\ 0 & \text{otherwise.} \end{cases}$$

Observe that ϕ is convex since $r \geq \alpha$ and that $\sum_i x_i = \sum_i y_i = W$ by hypothesis. Further, $\vec{y} \succ \vec{x}$ — indeed, (i) and (ii) of the definition are immediate, and for all $m \in [t-1]$ we have $x_1 + \dots + x_m \leq \min\{W, m c^\alpha\} = y_1 + \dots + y_m$. Thus Lemma 21 applies and we obtain

$$\sum_{i=1}^t s_i^r = \sum_{i=1}^t \phi(x_i) \leq \sum_{i=1}^t \phi(y_i) = \lfloor W/c^\alpha \rfloor c^r + (W - \lfloor W/c^\alpha \rfloor c^\alpha)^{r/\alpha}. \quad (2.2.4)$$

Observe that

$$(W - \lfloor W/c^\alpha \rfloor c^\alpha)^{r/\alpha} = (W/c^\alpha - \lfloor W/c^\alpha \rfloor)^{r/\alpha} \cdot c^r \leq (W/c^\alpha - \lfloor W/c^\alpha \rfloor) c^r,$$

where the inequality follows since $W/c^\alpha - \lfloor W/c^\alpha \rfloor < 1$ by definition and $\alpha \geq r$. The result therefore follows from (2.2.4). ◀

Finally, the following bound on binomial coefficients is folklore.

► **Lemma 23.** Let S be an arbitrary set, and let $|S| \geq a \geq b \geq 0$. Then we have

$$\binom{|S| - a}{b} \geq \frac{|S|^b}{(2a)^a}.$$

Proof. If $|S| \geq 2a$, then we have

$$\binom{|S| - a}{b} \geq \frac{(|S| - a)^b}{b^b} \geq \frac{(|S|/2)^b}{b^b} \geq \frac{|S|^b}{(2a)^a}.$$

If instead $|S| < 2a$, then we have

$$\binom{|S| - a}{b} \geq 1 \geq \frac{|S|^b}{(2a)^a}.$$

◀

2.3 Efficiently sampling small random subsets

Let $U = \{1, \dots, n\}$ be a universe of size n and let i be an integer with $0 \leq i \leq \mathcal{O}(\log n)$. How can we sample a random set $X \subseteq U$ such that each element $u \in U$ is picked with independent probability 2^{-i} ? Naively, we would do this by flipping a 2^{-i} -biased coin for each element $u \in U$, which overall takes time $\mathcal{O}(n)$. This will be too slow for our purposes, so we give an improved algorithm in `SampleSubset` with running time $\mathcal{O}(i + n/2^i)$ and prove its correctness in [Lemma 26](#) (the goal of this subsection).

As a key subroutine of `SampleSubset`, we will need to sample from a binomial distribution $\text{Bin}(n, 1/2^i)$. In the real-RAM model, there is a simple and highly-efficient algorithm for this (namely drawing a uniformly random element from $[0, 1]$ and using the inverse method), and more sophisticated methods are available (e.g., see Fishman [18]). However, as we work in the word-RAM model (see [Section 2.1.2](#)) and e.g. $\binom{n}{n/2}$ is a $\Theta(n)$ -bit number, the problem requires more thought. Bringmann [11, Theorem 1.19] provides a word-RAM algorithm to sample from $\text{Bin}(n, \frac{1}{2})$ in expected constant time.

► **Lemma 24** (Bringmann [11, Theorem 1.19]). *In the word-RAM model with $\Theta(\log n)$ bits per word, a binomial random variable $\text{Bin}(n, \frac{1}{2})$ can be sampled in expected time $\mathcal{O}(1)$.*

Farach-Colton and Tsai [16, Theorem 2] show how to turn a sampler for $\text{Bin}(n, \frac{1}{2})$ into a sampler for $\text{Bin}(n, p)$. We will only need the special case where $1/p$ is a power of two. In this case the methods of [16] yield a much faster sampler, but this is not formally stated. Since the $p = 1/2^i$ case is far simpler than the general case, for the convenience of the reader we prove our own self-contained corollary to [Lemma 24](#) rather than analysing [16] in depth.

► **Corollary 25.** *On the word-RAM with $w = \Omega(\log n)$, given n and i , a binomial random variable $\text{Bin}(n, 2^{-i})$ can be sampled in expected time $\mathcal{O}(i)$.*

Proof. We can sample from $\text{Bin}(n, 2^{-i})$ by tossing n fair coins in i rounds: In round $j = 1$, all coins are tossed, and in round $j > 1$, all coins that turned up heads in round $j - 1$ are tossed again. After round $j = i$, we count how many coins show heads and this is our sample from $\text{Bin}(n, 2^{-i})$, because the probability that any specific coin survives all rounds is exactly 2^{-i} .

We can efficiently simulate this procedure as follows: In round $j = 1$, we sample n_1 from $\text{Bin}(n, \frac{1}{2})$. In round $j > 1$, we sample n_j from $\text{Bin}(n_{j-1}, \frac{1}{2})$. We return n_i as our sample from $\text{Bin}(n, 2^{-i})$. Thus, all that is needed is to draw i samples from $\text{Bin}(n', \frac{1}{2})$ using [Lemma 24](#) for various values of $n' \leq n$. Therefore, the overall expected running time is $\mathcal{O}(i)$. ◀

We now apply [Corollary 25](#) to implement `SampleSubset`.

► **Lemma 26.** *`SampleSubset` is correct and runs in expected time $\mathcal{O}(i + 2^{-i}n)$.*

Proof. We first show that the random set X is indeed distributed as claimed for $i \geq 3$. To this end, let Y be a random subset $\{1, \dots, n\}$ that is sampled by including each element $u \in \{1, \dots, n\}$ with independent probability 2^{-i} . We claim that X and Y follow the same distribution. Indeed, $|Y|$ follows $\text{Bin}(n, 2^{-i})$. By symmetry, conditioned on $|Y|$, the set Y is equally likely to be any size- $|Y|$ subset of $\{1, \dots, n\}$. Hence, X and Y follow the same distribution, as required.

Now we describe how exactly the algorithm is implemented in order to achieve the expected running time. For $i \leq 2$, the algorithm runs in time $\mathcal{O}(n)$ as required. For $i \geq 3$, [line 4](#) takes time $\mathcal{O}(i)$ by [line 25](#). In order to achieve expected time $\mathcal{O}(2^{-i}n)$, [line 5](#) is implemented using a hash set: We sample uniformly random elements from $\{1, \dots, n\}$ (with

Algorithm 1 `SampleSubset`

Input : Integers n and i .**Output** : A random set $X \subseteq \{1, \dots, n\}$ that includes each element with independent probability 2^{-i} .

```

1 begin
2   if  $i \leq 2$  then
3     Sample  $X$  naively by independently flipping a  $2^{-i}$ -biased coin for each element
      of  $\{1, \dots, n\}$ 
4   else
5     Sample  $S$  from the binomial distribution  $\text{Bin}(n, 2^{-i})$  using Corollary 25.
6     Sample  $X$  as a uniformly random size- $S$  subset of  $\{1, \dots, n\}$ .

```

replacement) and one by one add them to the hash set until the hash set contains exactly S distinct elements. Each hash set operation takes $\mathcal{O}(1)$ expected time, so it remains to argue how many elements we have to sample.

Conditioned on $S \leq \frac{7}{8}n$, at any point it takes at most 8 samples in expectation until a new element is added to the hash set. Overall, the expected total running time to sample S distinct elements in line 5 is thus at most $8S \leq \mathcal{O}(2^{-i}n)$.

Conditioned on $S > \frac{7}{8}n$, a coupon collector argument shows that it takes an expected number $\mathcal{O}(n \log n)$ of samples until we have seen S distinct elements. This would be too large, so it remains to prove that the probability that $S > \frac{7}{8}n$ holds is vanishingly small. Indeed, S is binomially distributed with mean $\mathbb{E}(S) = 2^{-i}n$. Let $z = \frac{7}{8}n$ and note that $z \geq 7 \cdot 2^{-3} \geq 7 \mathbb{E}(S)$ holds by $i \geq 3$. Thus, we have $\mathbb{P}(S \geq z) \leq e^{-z} = e^{-7n/8}$ by [Lemma 14](#). By $\mathcal{O}(e^{-7n/8}n \log n) \leq \mathcal{O}(1)$, the running time in the event $S_{i,j} > \frac{7}{8}n$ vanishes in expectation.

To conclude, the overall expected time of lines 4 and 5 is $\mathcal{O}(i + 2^{-i}n)$ as claimed. \blacktriangleleft

Later, it will be technically convenient to make the constant in the expected running time of `SampleSubset` explicit.

► **Definition 27.** Let C_{sample} be such that for all n and i , `SampleSubset`(n, i) runs in expected time at most $C_{\text{sample}}(i + 2^{-i}n)$.

3 Independence oracle with cost

In this section, we study the edge estimation problem for IND-oracle algorithms, which are given access to the independence oracle $\text{IND}(G)$ in k -uniform hypergraphs G with a cost function $\text{cost}: 2^{V(G)} \rightarrow \mathbb{R}_{\geq 0}$. The cost can be thought of as the running time of a subroutine that detects the presence of an edge, but it can also be used to model other types of cost. We restrict our attention to cost functions that only depend on the size of the query, so we write $\text{cost}(S) = \text{cost}(|S|)$, and in this case we assume that the cost is regularly-varying with index $\alpha \in [0, k]$. In particular, in the case $\alpha = 0$, the cost corresponds to the number of queries and therefore yields bounds for independence oracles without costs.

In [Section 3.1](#), we prove some simple algebraic lemmas that we will use to express our bounds in terms of the parameter $g(k, \beta)$ already introduced in [Theorem 1](#). We then present an IND-oracle algorithm in [Section 3.2](#) that approximately counts edges of a given k -uniform hypergraph, proving the upper bound part of [Theorem 39](#) and, as a corollary, the upper bound part of [Theorem 1](#). Finally, in [Section 3.3](#) we prove that the total cost incurred by

our IND-oracle algorithm is optimal up to a polylogarithmic factor, establishing [Theorem 41](#) and, as a corollary, the lower bound part of [Theorem 1](#).

3.1 Algebraic preliminaries

Recall the following notation from the statement of [Theorem 1](#).

► **Definition 28.** For all real numbers k and β , we define

$$g(k, \beta) = \frac{1}{k} \cdot \left\lfloor \frac{k - \beta}{2} \right\rfloor \cdot \left(k - \beta - \left\lfloor \frac{k - \beta}{2} \right\rfloor \right).$$

In [Theorem 1](#), $g(k, \beta)$ will arise from a maximum over $\log n$ iterations of the main loop in our oracle algorithm; in [Theorem 41](#), it will arise from a maximum over k possible choices of input distribution for our lower bound. The arguments required are very similar in both cases, so we prove the necessary lemmas in this combined section. Our goals will be [Lemma 31](#) and [Corollary 33](#); these are bounds on algebraic expressions which arise naturally in bounding the total running time and oracle cost of [UncoLApprox](#) above in [Section 3.2](#) and in bounding the required oracle cost below in [Section 3.3](#). The proofs are standard applications of algebra and elementary calculus, and may be skipped on a first reading. We first prove some simple bounds on g .

► **Lemma 29.** For all $k \geq 0$ and $\beta \in [0, k]$, we have $g(k, \beta) \leq (k - \beta)^2 / (4k)$. In particular, $g(k, \beta) \leq k/4$.

Proof. Fix k and $\beta \in [0, k]$, and let $h(x) = x(k - \beta - x)/k$. Then since $\beta \in [0, k]$, h is a parabola maximised at $x = (k - \beta)/2$, so it follows that

$$h(k - \beta) \geq h(\lfloor (k - \beta)/2 \rfloor) = g(k, \beta).$$

We have $h(k - \beta) = (k - \beta)^2 / (4k)$, so the first part of the result follows. The second part is then immediate on observing that $(k - \beta)^2$ is maximised over $\beta \in [0, k]$ at $\beta = k/2$. ◀

► **Lemma 30.** For all integers $k \geq 2$ and all $\beta \in [0, k]$, we have $g(k, \beta) \geq g(k, 0) - \beta/2$.

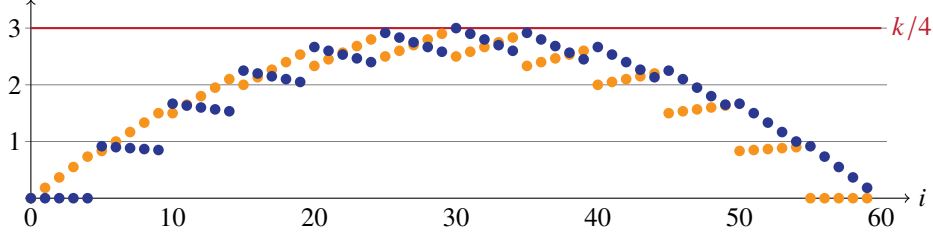
Proof. We first claim that for fixed k , $g(k, \beta)$ is continuous in β . This is immediate for all values of β except those at which the value of $\lfloor (k - \beta)/2 \rfloor$ “jumps”, i.e. those values of β such that $k - \beta$ is an odd integer. For such values of β , we must show that the limit of $g(k, x)$ is the same as x converges to β from above and below. Suppose $k - \beta$ is indeed an odd integer; then we have

$$\begin{aligned} \lim_{x \uparrow \beta} g(k, x) &= \frac{1}{k} \cdot \frac{k - \beta - 1}{2} \cdot \left(k - \beta - \frac{k - \beta - 1}{2} \right) = \frac{(k - \beta - 1)(k - \beta + 1)}{4k}, \\ \lim_{x \downarrow \beta} g(k, x) &= \frac{1}{k} \cdot \frac{k - \beta + 1}{2} \cdot \left(k - \beta - \frac{k - \beta + 1}{2} \right) = \frac{(k - \beta + 1)(k - \beta - 1)}{4k}. \end{aligned}$$

Thus $\lim_{x \uparrow \beta} g(k, x) = \lim_{x \downarrow \beta} g(k, x)$ as required, and $g(k, \beta)$ is continuous in β as claimed.

We next observe that for all non-negative integers x and all fixed $k \geq 2$, $g(k, \beta)$ is differentiable over $\beta \in (x, x + 1)$, since $\lfloor (k - \beta)/2 \rfloor$ is constant on this interval. The derivative is given by

$$\frac{\partial}{\partial \beta} g(k, \beta) = -\frac{1}{k} \left\lfloor \frac{k - \beta}{2} \right\rfloor \geq -\frac{1}{k} \cdot \lim_{\beta \downarrow 0} \left\lfloor \frac{k - \beta}{2} \right\rfloor \geq -\frac{1}{2},$$



■ **Figure 2** Depicted are the functions $i \mapsto \frac{1}{k} f_1(\hat{L}_i, \hat{\gamma}_i)$ (●) and $i \mapsto \frac{1}{k} f_2(\hat{L}_i, \hat{\gamma}_i)$ (●) from Lemma 31 for $n = 2^{60}$ and $k = 12$. For each fixed integer $L \in \{0, \dots, k-1\}$, the functions are linear in γ , which gives rise to the step artefacts; when γ is fixed, the functions are degree-2 polynomials in L , which causes the overall parabolic shape. When k and $\log n$ are even, the overall maximum is equal to $\frac{k}{4}$ and achieved at $i = \frac{\log n}{2}$.

where the final inequality follows since $k \geq 2$ is an integer and $\beta > x \geq 0$. Since we have already shown that on fixing k and viewing g as a function of β , g is continuous everywhere and only fails to be differentiable at integer values, it follows by the mean value theorem that for all $\beta \geq 0$, $g(k, \beta) \geq g(k, 0) - \beta/2$ as required. ◀

We now prove our first key upper bound. We will use this both directly to bound the running time of `UncolApprox` and in the proof of Lemma 32.

► **Lemma 31.** Fix integers $k \geq 2$ and $n \geq 1$. For all integers $i \geq 0$, let $\hat{L}_i \in \mathbb{N}$ and $\hat{\gamma}_i \in [0, 1)$ be the unique values with $2^{ik} = n^{\hat{L}_i + \hat{\gamma}_i}$. Let

$$F_i = n^{(\hat{L}_i + \hat{\gamma}_i)(k - \hat{L}_i)/k} \cdot \max\{2^{-i}, n^{-\hat{\gamma}_i}\} \quad (3.1.1)$$

For all non-negative reals β , we have

$$\max_i F_i \cdot 2^{-i\beta} = \max_{L \in \{0, \dots, k - \lceil \beta \rceil\}} n^{L(k - L - \beta)/k} = n^{g(k, \beta)}. \quad (3.1.2)$$

Moreover, if $\beta > k - 1$ and $i \geq (\log n)/k$, then we have

$$F_i \cdot 2^{-i\beta} \leq n^{-(\beta - (k-1))}.$$

Proof. Using $2^i = n^{(\hat{L}_i + \hat{\gamma}_i)/k}$ and taking base- n logarithms of both sides of Equation (3.1.1), we get $\log_n(F_i \cdot 2^{-i\beta}) = \frac{1}{k} \max\{f_1(\hat{L}_i, \hat{\gamma}_i), f_2(\hat{L}_i, \hat{\gamma}_i)\}$ where $f_1, f_2 : \mathbb{N} \times [0, 1] \rightarrow \mathbb{R}$ are the functions depicted in Figure 2 and defined via

$$\begin{aligned} f_1(L, \gamma) &= (L + \gamma)(k - L) - (L + \gamma) - (L + \gamma)\beta = (L + \gamma)(k - L - 1 - \beta), \\ f_2(L, \gamma) &= (L + \gamma)(k - L) - k\gamma - (L + \gamma)\beta = (L + \gamma)(k - L - \beta) - k\gamma. \end{aligned}$$

We now provide an upper bound for both functions f_1 and f_2 . Both functions are linear in γ , and thus the maximum is achieved at $\gamma = 0$ or $\gamma = 1$. Thus it remains to upper bound the following four functions of L :

$$\begin{aligned} f_1(L, 0) &= L(k - L - 1 - \beta), \\ f_1(L, 1) &= (L + 1)(k - L - 1 - \beta), \\ f_2(L, 0) &= L(k - L - \beta), \\ f_2(L, 1) &= (L + 1)(k - L - \beta) - k. \end{aligned}$$

Note that for all L , $f_1(L, 1) = f_2(L + 1, 0)$, $f_1(L, 0) \leq f_2(L, 0)$, and $f_2(L, 1) \leq f_2(L, 0)$ hold; thus

$$\max_i \log_n(F_i \cdot 2^{-i\beta}) = \frac{1}{k} \max_i \left(\max \{f_1(\hat{L}_i, \hat{\gamma}_i), f_2(\hat{L}_i, \hat{\gamma}_i)\} \right) = \frac{1}{k} \max_{L \in \{0, \dots, k\}} f_2(L, 0).$$

By elementary calculus, $f_2(L, 0)$ is maximised over integer values of L at $L = \lfloor (k - \beta)/2 \rfloor$. Moreover, this value of L is at most $k - \lceil \beta \rceil$; indeed, if $\beta \leq k - 1$ then we have $\lfloor (k - \beta)/2 \rfloor = \lfloor (k - \beta + 1)/2 \rfloor \leq (k - \beta + 1)/2 \leq k - \beta$, and if $\beta > k - 1$ then we have $\lfloor (k - \beta)/2 \rfloor = k - \lceil \beta \rceil = 0$. Thus we have

$$\max_i F_i \cdot 2^{-i\beta} = n^{\frac{1}{k} \max_{0 \leq L \leq k - \lceil \beta \rceil} (f_2(L, 0))} = n^{\frac{1}{k} f_2(\lfloor \frac{k - \beta}{2} \rfloor, 0)} = n^{g(k, \beta)},$$

as required.

Finally, suppose $\beta > k - 1$ and $i \geq (\log n)/k$, so that $L \geq 1$. Arguing as before,

$$\begin{aligned} \log_n(F_i \cdot 2^{-i\beta}) &\leq \frac{1}{k} \max_{L \geq 1} f_2(L, 0) = \max_{L \geq 1} L(k - L - \beta) \\ &= \max_{L \geq 1} (L(1 - L) - L(\beta - (k - 1))) . \end{aligned}$$

The function inside the maximum is decreasing in L for $L \geq 1$, so we have $\log_n(F_i \cdot 2^{-i\beta}) \leq -(\beta - (k - 1))$ and the result follows. \blacktriangleleft

We now state what is essentially our second key lemma of the section; however, for technical convenience, we replace cost functions by their associated polynomials. We will then use this lemma to prove the actual result we need in [Corollary 33](#).

► **Lemma 32.** *Fix integers $k \geq 2$ and $n \geq 1$. For all integers $i \geq 0$, let $\hat{L}_i \in \mathbb{N}$ and $\hat{\gamma}_i \in [0, 1)$ be the unique values with $2^{ik} = n^{\hat{L}_i + \hat{\gamma}_i}$. Let*

$$F_i = n^{(\hat{L}_i + \hat{\gamma}_i)(k - \hat{L}_i)/k} \cdot \max\{2^{-i}, n^{-\hat{\gamma}_i}\}.$$

For all $\beta \in [0, k]$, we have

$$\max_i F_i \cdot (\max\{n/2^i, n^{1/4}\})^\beta = \max_{r \in \{\lceil \beta \rceil, \dots, k\}} n^{r(k - r + \beta)/k} = n^{\beta + g(k, \beta)}.$$

Moreover, if $\beta > k - 1$ and $i \geq (\log n)/k$, then we have

$$F_i \cdot (\max\{n/2^i, n^{1/4}\})^\beta \leq n^{\max\{3\beta/4, k - 1\}}.$$

Proof. Observe that by [Lemma 31](#) applied with $\beta = 0$, $F_i \leq n^{g(k, 0)}$. By [Lemma 30](#), it follows that $F_i \leq n^{g(k, \beta) + \beta/2}$, and so

$$F_i n^{\beta/4} \leq n^{g(k, \beta) + 3\beta/4} < n^{\beta + g(k, \beta)}. \quad (3.1.3)$$

On the other hand, by [Lemma 31](#) we have

$$\max_i F_i (n/2^i)^\beta = n^{g(k, \beta) + \beta}.$$

It follows that

$$\max_i F_i \cdot (\max\{n/2^i, n^{1/4}\})^\beta = n^{g(k, \beta) + \beta},$$

as required. Moreover, by substituting $L = k - r$ and applying [Lemma 31](#), we have

$$\begin{aligned} \max_{r \in \{\lceil \beta \rceil, \dots, k\}} n^{r(k-r+\beta)/k} &= \max_{L \in \{0, \dots, k - \lceil \beta \rceil\}} n^{(k-L)(L+\beta)/k} \\ &= \max_{L \in \{0, \dots, k - \lceil \beta \rceil\}} n^{\beta + L(k-L-\beta)/k} = n^{\beta + g(k, \beta)}, \end{aligned}$$

again as required.

Finally, suppose $\beta > k - 1$ and $i \geq (\log n)/k$. Then by [Lemma 31](#) we have

$$F_i(n/2^i)^\beta \leq n^{\beta - (\beta - (k-1))} = n^{k-1}.$$

Moreover, in this case we have $g(k, \beta) = 0$ and so (3.1.3) implies that $F_i n^{\beta/4} \leq n^{3\beta/4}$. The result therefore follows. \blacktriangleleft

Finally, we restate [Lemma 32](#) in terms of a general slowly-varying cost function; this is the form of the result we will actually use. Here $C^+(n)$ will be the upper bound on oracle cost we derive in [Section 3.2](#).

► **Corollary 33.** *Let cost be a regularly-varying parameterised cost function with parameter k and slowly-varying component σ , and let $\alpha_k \in [0, k]$ be the index of cost_k for all $k \geq 2$. For each integer i , let $\hat{L}_i(n) \in \mathbb{N}$ and $\hat{\gamma}_i(n) \in [0, 1)$ be the unique values with $2^{ik} = n^{\hat{L}_i(n) + \hat{\gamma}_i(n)}$, and let*

$$\begin{aligned} F_i(n) &= n^{(\hat{L}_i(n) + \hat{\gamma}_i(n))(k - \hat{L}_i(n))/k} \cdot \max\{2^{-i}, n^{-\hat{\gamma}_i(n)}\}, \\ C_k^+(n) &= \max_{0 \leq i \leq \log n - 1} \left(F_i(n) \cdot \text{cost}_k(\max\{n/2^i, n^{1/4}\}) \right). \end{aligned}$$

Then for all sequences $k = k(n)$ with $k(n) \in \{2, \dots, n\}$ for all n , we have:

- (i) $C_k^+(n) = n^{g(k, \alpha_k) + o(1)} \text{cost}_k(n)$ as $n \rightarrow \infty$; and
- (ii) if σ is eventually non-decreasing or if there exists $\eta > 0$ such that $\alpha_k \geq k - 1 + \eta$ for all k , then $C_k^+(n) = \mathcal{O}(n^{g(k, \alpha_k)} \text{cost}_k(n))$ as $n \rightarrow \infty$.

Proof. Let σ be the slowly-varying component of cost, so that $\text{cost}_k(n) = n^{\alpha_k} \ell(n)$. By [Lemma 32](#) applied with $\beta = \alpha_k$, we have

$$\begin{aligned} C_k^+(n) &= \max_i \left(F_i(n) \cdot (\max\{n/2^i, n^{1/4}\})^{\alpha_k} \cdot \sigma(\max\{n/2^i, n^{1/4}\}) \right) \\ &\leq n^{g(k, \alpha_k) + \alpha_k} \cdot \max_i \left(\sigma(\max\{n/2^i, n^{1/4}\}) \right) \\ &= n^{g(k, \alpha_k)} \text{cost}_k(n) \cdot \max_i \left(\frac{\sigma(\max\{n/2^i, n^{1/4}\})}{\sigma(n)} \right). \end{aligned} \tag{3.1.4}$$

Since σ is slowly-varying, part (i) follows immediately from (3.1.4). Moreover, if σ is eventually non-decreasing, then since $\max\{n/2^i, n^{1/4}\} \leq n$, (ii) follows from (3.1.4). Finally, suppose $\alpha_k \geq k - 1 + \eta$ for some fixed $\eta > 0$. We split into two cases depending on i .

If $i < (\log n)/k$, then we have $\hat{L}_i(n) = 0$, so $F_i(n) = 1$. By [Lemma 8](#)(ii) and (iii), for sufficiently large n we have $\text{cost}_k(n/2^i) \leq \text{cost}_k(n)/2^{i\alpha_k/2} = \mathcal{O}(\text{cost}_k(n))$ and $\text{cost}_k(n^{1/4}) \leq n^{\beta/2} = o(\text{cost}_k(n))$; it follows that

$$F_i(n) \text{cost}_k(\max\{n/2^i, n^{1/4}\}) = \mathcal{O}(\text{cost}_k(n)).$$

If instead $i \geq (\log n)/k$, then by [Lemma 32](#) we have

$$\begin{aligned} F_i(n) \text{cost}_k(\max\{n/2^i, n^{1/4}\}) &= F_i(n) \cdot (\max\{n/2^i, n^{1/4}\})^{\alpha_k} \cdot \sigma(\max\{n/2^i, n^{1/4}\}) \\ &\leq n^{\max\{3\alpha_k/4, k-1\}} \cdot \sigma(\max\{n/2^i, n^{1/4}\}) \\ &= \text{cost}_k(n) \cdot n^{-\alpha_k + \max\{3\alpha_k/4, k-1\}} \cdot \frac{\sigma(\max\{n/2^i, n^{1/4}\})}{\sigma(n)}. \end{aligned}$$

Observe that $\alpha_k/4 \geq (k-1)/4 \geq 1/4$, and that $\alpha_k - k - 1 \geq \eta$, which is a constant; by [Lemma 8\(iii\)](#), it follows that the $n^{-\alpha_k + \max\{3\alpha_k/4, k-1\}}$ term dominates the $\sigma(\cdot)$ terms and so the left-hand side is $\mathcal{O}(\text{cost}_k(n))$. We have therefore shown that $C_k^+(n) = \mathcal{O}(\text{cost}_k(n))$ in all cases, as required. \blacktriangleleft

3.2 Oracle algorithm for edge estimation

Our IND-oracle algorithm for the edge estimation problem has two components:

1. A randomised IND-oracle algorithm [SparseCount](#) by Meeks [28, Theorem 6.1] to enumerate all edges of a given hypergraph when given access to the independence oracle. [SparseCount](#) is an exact enumeration algorithm, and its running time can be tightly controlled by setting a threshold parameter M and aborting it after M edges are seen. This version of the algorithm requires only roughly $\mathcal{O}(M)$ queries, but may return [TooDense](#) to indicate that the hypergraph has more than M edges.
2. A randomised IND-oracle algorithm [UncolApprox](#) that invokes [SparseCount](#) on smaller and smaller random subgraphs until they are sparse enough so that [SparseCount](#) succeeds at enumerating all edges under the time constraints. We describe [UncolApprox](#) in [Section 3.2.2](#). From the exact numbers of edges that were observed in several independently sampled random subgraphs, the algorithm then calculates and returns an approximation to the number of edges in the whole hypergraph.

We give a self-contained overview of [SparseCount](#) in [Section 3.2.1](#). We then set out [UncolApprox](#) in [Section 3.2.2](#), describe the intuition behind it, and sketch a proof of its correctness. We then prove its correctness formally in [Section 3.2.3](#), and bound its running time in [Sections 3.1](#) and [3.2.4](#).

3.2.1 SparseCount: Enumerate all edges in sparse hypergraphs

All the properties of [SparseCount](#) we need will follow from [28], but for completeness we sketch how [SparseCount](#) works and give some intuition. Note that the version of [SparseCount](#) described in [28, Algorithm 1] gives a deterministic guarantee of correctness when supplied with a deterministic oracle. For our purposes this is unnecessary, and so our sketch will be of a slightly simpler version; in [28] our uniformly random sets are replaced with a deterministic equivalent using k -perfect hash functions.

[SparseCount](#) has access to a k -uniform hypergraph through its independence oracle. Moreover, [SparseCount](#) is given a set $U \subseteq V(G)$, the integer k , and a threshold parameter $M \geq 1$ as input. [SparseCount](#)(IND(G), U , k , M) works as follows:

1. Sample t uniformly random colourings $c_1, \dots, c_t: U \rightarrow [k]$ for $t = \Theta(e^{2k} \log|U|)$.
2. For each i from 1 to t , run a subroutine [RecEnum](#) to recursively enumerate and output all edges e that are colourful with respect to c_i . Keep track of the number of unique edges seen so far by incrementing a counter each time [RecEnum](#) outputs an edge e that is colourful with respect to c_i and *not* colourful with respect to any c_1, \dots, c_{i-1} .

3. If at any point the counter exceeds M , abort the execution immediately and return **TooDense**. Otherwise, return the final contents of the counter.

Next, we describe the missing subroutine **RecEnum** that is called by **SparseCount**. The input for **RecEnum** is a tuple (U_1, \dots, U_k) of disjoint subsets of U as well as the threshold parameter $M \geq 1$. In the initial call, U_j is the j^{th} colour class of some colouring c_i from step 1 of **SparseCount**. **RecEnum**($\text{IND}(G), U_1, \dots, U_k$) then proceeds as follows:

1. If the set e with $e = U_1 \cup \dots \cup U_k$ has size at most k , we have $|U_1| = \dots = |U_k| = 1$, and e is an edge (as determined by a call $\text{IND}(G)(e)$ to the independence oracle of G), then output e ; otherwise, do nothing.
2. Otherwise, independently and uniformly at random split each part U_j into two disjoint parts U_{j0} and U_{j1} of near-equal size, and recurse on all tuples $(U_{0b_0}, \dots, U_{kb_k})$ for all bit vectors $b \in \{0, 1\}^k$ for which $G[U_{0b_0} \cup \dots \cup U_{kb_k}]$ contains at least one edge (as determined by a call to the independence oracle on $G[U_{0b_0} \cup \dots \cup U_{kb_k}]$).

The intuition for the correctness of the algorithm is as follows: By standard colour-coding arguments, with high probability every edge is colourful with respect to at least one of the t random colourings c_i and so **RecEnum** will find it (regardless of how parts are split in step 2). The running time of **RecEnum** is not affected too much by edges that are not colourful, because with high probability a large proportion of edges are colourful and uncoloured edges get deleted quickly when the subsets are sampled. We encapsulate the relevant properties of **SparseCount** in the following lemma.

► **Lemma 34.** *There is a randomised IND-oracle algorithm **SparseCount**($\text{IND}(G), U, k, M, \delta$) with the following behaviour:*

- **SparseCount** takes as input a set $U \subseteq V(G)$, integers k and M , and a rational number $\delta > 0$.
- **SparseCount** may output the integer $e(G[U])$, **TooDense** (‘The hypergraph $G[U]$ definitely has more than M edges’), or **RTE** (‘Allowed running time exceeded’).
- If $e(G[U]) \leq M$ holds, then **SparseCount**(U, k, M, δ) outputs either $e(G[U])$ or **RTE**; otherwise, it outputs either **TooDense** or **RTE**.
- **SparseCount** invokes the uncoloured independence oracle of G at most

$$\mathcal{O}\left(\log \frac{1}{\delta} \cdot e^{2k} \log^2 |U| \cdot \min\{M, e(G[U])\}\right)$$

times, and runs in time at most

$$\mathcal{O}\left(\log \frac{1}{\delta} \cdot e^{2k} \log^2 |U| \cdot \min\{M, e(G[U])\} \cdot |U|\right)$$

aside from that.

- The probability that **SparseCount** outputs **RTE** is at most δ .

We omit a formal proof of **Lemma 34**, since it follows easily from Meeks [28, Theorem 1.1] with a very similar argument to [28, Theorem 6.1], with the following minor caveats. First, we have applied the standard probability amplification result of **Lemma 18** to pass from bounds on the expected running time to deterministic bounds that hold with probability at least $2/3$, and from there we have applied **Lemma 19** to pass to bounds that hold with probability $1 - \delta$. (We take the cost function to be the number of queries, and the interval of **Lemma 19** to be $\{e(G[U])\}$.) Second, the theorem statements in [28] do not explicitly separate bounds on the number of oracle queries from the total running time without oracle queries, and in particular this means the dependence on n in their running time bounds is stated as $n^{\mathcal{O}(1)}$. However, from the proof of [28, Theorem 1.1] in Section 4 we see that there are $\mathcal{O}(e^{k+o(k)} e(G[U]) \log^2 n)$ total oracle calls, and that the running time without oracle calls is $\mathcal{O}(e^{k+o(k)} e(G[U]) n \log^2 n)$.

3.2.2 UncolApprox: Approximately count all edges in hypergraphs

In this section, we analyse our main algorithm, **UncolApprox**, which is laid out as [Algorithm 2](#). We write n for the number of vertices of G and $m = e(G)$ for the number of edges.

The basic idea of the algorithm is both simple and standard. In the main loop over $i = 0, \dots, \log n$ of lines 6–23, for a suitably-chosen integer t_i , we sample t_i independent random subsets $U_{i,j}$ of $V(G)$, including each element with probability $p_i = 1/2^i$. We then count the edges in each $e(G[U_{i,j}])$ using **SparseCount** with a suitably-chosen threshold $M_{i,j}$. It is easy to see by linearity of expectation that $\mathbb{E}(e(G[U_{i,j}])) = p_i^k e(G)$, so if our calls to **SparseCount** succeed then in expectation we return $e(G)$ in line 23.

The main subtlety of **UncolApprox** is in optimising our choice of parameters to minimise the running time while still ensuring correctness with high probability. The intuition here will tie into our lower bound proof in [Section 3.3](#), so we go into detail rather than simply presenting calculations.

By line 5, we can assume without loss of generality that n is a power of two, which simplifies the notation. Moreover, we write $m = n^\delta$ and split δ into its integral part $L = \lfloor \delta \rfloor$ and its rational part $\gamma = \delta - L$, and we remark that $m \leq \binom{n}{k} < n^k$ and thus $0 \leq L \leq k - 1$ and $\gamma \in [0, 1)$ holds.

The first parameter in **UncolApprox** is $p_i = 2^{-i}$, which is the probability of independently including each vertex v in the set $U_{i,j}$. Note that each edge $e \in E(G)$ has k elements and thus survives in $G[U_{i,j}]$ with probability $p_i^k = 2^{-ik}$. Therefore, the expected number of edges in $G[U_{i,j}]$ is $p_i^k m$, and for a given iteration i , the expected total number of edges $T_i := \sum_j e(G[U_{i,j}])$ is $t_i p_i^k m$. In the following definition, we define i^* as the largest integer such that this expected value remains at least 1.

► **Definition 35.** *Given an m -edge graph G , let $i^* = i^*(G)$ be the largest value of i such that $p_i^k m \geq 1$ holds (where $p_i = 2^{-i}$ as in **UncolApprox**).*

Note that $i^* \leq \log n - 1$ follows from $m < n^k$ and $p_{i^*}^k m \leq 2^{k+1}$. We will show in [Lemma 36](#) using Chebyshev’s inequality that T_i is very likely to be an ε -approximation of $\mathbb{E}(T_i)$ whenever $i \leq i^*$. Observe that if this holds, then whenever $T_i \leq M_i$, all iterations of **SparseCount** succeed and **UncolApprox** outputs a valid ε -approximation of m .

Given [Lemma 36](#), the reason **UncolApprox** is correct will be as follows: Whenever $i \leq i^*$, if all calls to **SparseCount** succeed in iteration i , we output an ε -approximation to m as required. The parameter M_i is chosen in such a way that if the number of edges is roughly 2^{ik} , then all calls to **SparseCount** are indeed likely to succeed. When $i = i^*$ we do indeed have $m \approx 2^{i^*k}$, so we are very likely to output a valid ε -approximation in this iteration if we have not already done so.

For the correctness of **UncolApprox**, the concentration analysis of [Lemma 36](#) is therefore crucial. The values \hat{L}_i and $\hat{\gamma}_i$ with $2^{ik} = n^{\hat{L}_i + \hat{\gamma}_i}$ arise naturally in this analysis. Recall that t_i is the number of random graphs $G[U_{i,j}]$ we count to estimate m ; to minimise the running time, we want to choose t_i as small as possible while maintaining concentration of T_i , so we choose it to scale with the maximum possible value of $\text{Var}(T_i)$ on the assumption that $m \lesssim 2^{ik}$. This maximum value always arises when $m \approx 2^{ik}$, and there are two possible “extreme cases” of input graphs G with this edge count for which $\text{Var}(T_i)$ could be near-maximum. The value of $\hat{\gamma}_i$ determines which case dominates, and hence which value we should take for t_i . (This is the source of the maximum in line 9.)

Suppose for simplicity that m is a power of 2^k , writing $m = 2^{ik} = n^{\hat{L}_i + \hat{\gamma}_i}$. The first extreme case is a *single-rooted star graph*. Such a graph G contains a single size- $(k - \hat{L}_i - 1)$ root $R \subseteq V(G)$ such that every edge of G contains R . Note that there are roughly $n^{\hat{L}_i + 1}$

■ **Algorithm 2** `UncolApprox`

This IND-oracle algorithm applies `SparseCount` to carefully-chosen numbers of successively sparser random subgraphs of G until the samples become sparse enough so that `SparseCount` stops returning `TooDense` and starts consistently returning accurate edge counts. At this point, the algorithm takes the average of the edge counts over all samples at that density and renormalises to obtain the final estimate.

Oracle: Independence oracle $\text{IND}(G)$ of an n -vertex k -hypergraph G .

Input: $n, k \in \mathbb{N}$ and $\varepsilon \in (0, 1)$.

Output: $x \in \mathbb{Q}$ that, with probability at least $2/3$, is an ε -approximation to $e(G)$.

```

1 begin
2   if  $n^k \leq \varepsilon^{-2}$  or  $n \leq k^5$  then
3     Enumerate all size- $k$  subsets of  $V(G)$ , check the independence oracle for each
      to compute  $e(G)$  by brute force, and return this value.
4   if  $n$  is not a power of two then
5     Set  $n = 2^{\lceil \log n \rceil}$  to add at most  $n - 1$  padding vertices. Before sending any
      future query to the oracle, we remove all padding vertices from the query.
6   for  $i = 0, \dots, \log n - 1$  do
7     Set the vertex survival probability  $p_i = 1/2^i$ .
8     Set  $\hat{L}_i \in \{0, 1, \dots, k\}$  and  $\hat{\gamma}_i \in \{\frac{0}{\log n}, \frac{1}{\log n}, \dots, \frac{\log n - 1}{\log n}\}$  to be the unique
      values that satisfy  $n^{\hat{L}_i + \hat{\gamma}_i} = 2^{ik}$ .
9     Set  $F_i = n^{(\hat{L}_i + \hat{\gamma}_i)(k - \hat{L}_i)/k} \cdot \max\{2^{-i}, n^{-\hat{\gamma}_i}\}$ .
10    Set  $t_i = \lceil \varepsilon^{-2} 10k^2 2^k \log n \cdot F_i \rceil$  and set  $M_i = 2^{k+1} \cdot t_i$ .
11    for  $j = 1, \dots, t_i$  do
12      Use SampleSubset (see Lemma 26) to efficiently sample a random
        subset  $U_{i,j} \subseteq V(G)$  that includes each element with independent
        probability  $p_i$ . If the total running time of all SampleSubset calls ever
        exceeds  $20C_{\text{sample}} \sum_{i=0}^{\log n - 1} t_i(i + n/2^i)$ , then abort and return RTE.
13      if  $|U_{i,j}| > \max\{7p_i n, 7k \ln n\}$  then
14        Return RTE.
15      Let  $C_{i,j} = \text{SparseCount}(\text{IND}(G), U_{i,j}, k, M_{i,j}, n^{-5k}/120)$ .
16      if  $C_{i,j} = \text{TooDense}$  then
17        Continue in line 7 with the next iteration of the outer for-loop.
18      else if  $C_{i,j} = \text{RTE}$  then
19        return RTE.
20      else
21        Set  $M_{i,j+1}$  to  $M_{i,j} - C_{i,j}$ .
22    if none of  $C_{i,1}, \dots, C_{i,t_i}$  are TooDense then
23      return  $\frac{1}{p_i^k t_i} \sum_{j=1}^{t_i} C_{i,j}$ .
```

possible edges containing R , so G can indeed have roughly m edges since $0 \leq \hat{\gamma}_i < 1$. In order for T_i to be an ε -approximation to $\mathbb{E}(T_i)$, it is necessary that the algorithm finds *some* edge during iteration i , so at least one set $U_{i,j}$ must contain R . Since $R \subseteq U_{i,j}$ happens with probability $p_i^{k-\hat{L}_i-1}$, we need $1/p_i^{k-\hat{L}_i-1}$ samples in expectation until we see even a single edge, where

$$1/p_i^{k-\hat{L}_i-1} = 2^{i(k-\hat{L}_i-1)} = n^{(L+\hat{\gamma}_i)(k-\hat{L}_i)/k} \cdot 2^{-i} \quad (3.2.1)$$

This shows why F_i in line 9 can't be much smaller if the first term of the maximum dominates. (Our choice of t_i then includes some additional minor factors to amplify the probability of producing an ε -approximation.)

The second extreme case is a *many-rooted star graph*. Such a graph G contains $\lceil n^{\hat{\gamma}_i} \rceil$ disjoint *roots* R_x of $k - \hat{L}_i$ vertices each. The edges of the graph are precisely the size- k sets e that contain at least one R_x . Observe that each root has roughly $n^{\hat{L}_i}$ incident edges, so G has roughly $n^{\hat{L}_i+\hat{\gamma}_i} = m$ edges. Again, in order for T_i to be an ε -approximation to $\mathbb{E}(T_i)$ the algorithm must find some edge, so some set $U_{i,j}$ must contain a root. Each set $U_{i,j}$ has probability $p_i^{k-\hat{L}_i}$ to contain any particular root, and thus probability at most $p_i^{k-\hat{L}_i} \cdot n^{\hat{\gamma}_i}$ to contain at least one root. Thus, the expected number of samples until we have seen a single edge is at least:

$$1/(p_i^{k-\hat{L}_i} \cdot n^{\hat{\gamma}_i}) = 2^{i(k-\hat{L}_i)} \cdot n^{-\hat{\gamma}_i} = n^{(\hat{L}_i+\hat{\gamma}_i)(k-\hat{L}_i)/k} \cdot n^{-\hat{\gamma}_i} \quad (3.2.2)$$

This shows why F_i in line 9 can't be much smaller if the second term of the maximum dominates.

We refrain from making these claims on the optimality of the choice of t_i (and thus the running time of [UncolApprox](#)) any more formal, because we provide lower bounds for *any* algorithm in [Section 3.3](#). However, we think it is worth giving the intuition since the proof of the lower bound uses the first extreme case as a source of hard input graphs. (The calculations in the proof of [Lemma 31](#) show that while the second case is necessary for correctness — without it we will take too few samples to obtain concentration on our output when $\hat{\gamma}_i$ is close to 1 — it does not end up affecting the running time.)

3.2.3 Correctness of UncolApprox

In the following lemma, we show that, for every iteration $i \leq i^*$ and with high probability, the average and normalised number of edges in the subgraphs $G[U_{i,j}]$ is a good approximation to the number of edges in G .

► **Lemma 36.** *In [UncolApprox](#), suppose $i \leq i^*(G)$ and $\varepsilon \geq n^{-k/2}$. We have*

$$\mathbb{P}\left(\left|e(G) - \frac{1}{p_i^{k t_i}} \sum_{j=1}^{t_i} e(G[U_{i,j}])\right| \geq \varepsilon \cdot e(G)\right) \leq \frac{1}{10k \log n}.$$

Proof. We first set out some notation. For convenience, for all $0 \leq i \leq \log n - 1$ and all $j \in [t_i]$, let $Z_{i,j} = e(G[U_{i,j}])$, let $Z_i = \frac{1}{p_i^{k t_i}} \sum_{j=1}^{t_i} Z_{i,j}$, and let $m = e(G)$. In this notation, our goal is to prove that $|Z_i - m| \leq \varepsilon m$ with probability at least $1 - 1/(10k \log n)$. We will prove the result by bounding the variance of Z_i above and applying Chebyshev's inequality; observe by linearity of expectation that $\mathbb{E}(Z_i) = m$.

For all $e \in E(G)$, let 1_e be the indicator random variable of the event $e \subseteq U_{i,j}$. By linearity of expectation, for all j , we have

$$\text{Var}(Z_{i,j}) = \mathbb{E}(Z_{i,j}^2) - \mathbb{E}(Z_{i,j})^2 \leq \sum_{\substack{(e,f) \in E(G)^2 \\ e \cap f \neq \emptyset \\ e \neq f}} \mathbb{E}(1_e 1_f) = \sum_{\substack{(e,f) \in E(G)^2 \\ e \cap f \neq \emptyset \\ e \neq f}} p_i^{|e \cup f|}. \quad (3.2.3)$$

For any set $A \subseteq V(G)$, we write d_A for the number of edges in G which contain A . For all $e, f \in E(G)$ we have $|e \cup f| = 2k - |e \cap f|$, so it follows from (3.2.3) that

$$\text{Var}(Z_{i,j}) \leq \sum_{\ell=1}^{k-1} \sum_{\substack{A \subseteq V(G) \\ |A|=\ell}} \sum_{\substack{(e,f) \in E(G)^2 \\ e \cap f = A}} p_i^{2k-|A|} = \sum_{\ell=1}^{k-1} \sum_{\substack{A \subseteq V(G) \\ |A|=\ell}} d_A^2 p_i^{2k-\ell}. \quad (3.2.4)$$

Observe that for any set A of size $\ell \in [k-1]$, we have $d_A \leq \min\{m, n^{k-\ell}\}$; thus by (3.2.4),

$$\text{Var}(Z_{i,j}) \leq \sum_{\ell=1}^{k-1} \min\{m, n^{k-\ell}\} p_i^{2k-\ell} \sum_{\substack{A \subseteq V(G) \\ |A|=\ell}} d_A. \quad (3.2.5)$$

For all $\ell \in [k-1]$, every edge of G contains exactly $\binom{k}{\ell}$ sets A of size ℓ , and every set A of size ℓ is contained in precisely d_A edges of G . Thus by double-counting,

$$\sum_{\substack{A \subseteq V(G) \\ |A|=\ell}} d_A = \binom{k}{\ell} m \leq 2^k m. \quad (3.2.6)$$

It therefore follows from (3.2.5) that

$$\text{Var}(Z_{i,j}) \leq 2^k p_i^{2k} m \sum_{\ell=1}^{k-1} \min\{m, n^{k-\ell}\} p_i^{-\ell}. \quad (3.2.7)$$

Write $m = n^{L+\gamma}$, where $0 \leq \gamma < 1$ and L is an integer. If $\gamma = 0$, then the two terms in the minimum are equal. Otherwise, the first term in the minimum is achieved for $L + \gamma < k - \ell$, which is equivalent to $\ell \leq k - L - 1$, and the second term is achieved for $\ell \geq k - L$. Substituting in $p_i = 2^{-i}$, it follows from (3.2.7) that

$$\text{Var}(Z_{i,j}) \leq 2^{k-2ik} m^2 \sum_{\ell=1}^{k-L-1} 2^{i\ell} + 2^{k-2ik} m n^k \sum_{\ell=k-L}^{k-1} (2^i/n)^\ell. \quad (3.2.8)$$

Recall $0 \leq i \leq \log n$. We observe that the terms in the first sum in (3.2.8) are non-decreasing in ℓ , so the largest term is achieved for $\ell = k - L - 1$. Conversely, the terms of the second sum are non-increasing in ℓ , which means that the largest term is achieved for $\ell = k - L$. Thus, (3.2.8) implies that

$$\text{Var}(Z_{i,j}) \leq k 2^{-iL-ik+k} m \cdot \max\{2^{-i} m, n^L\} = k 2^{-iL-ik+k} m^2 \max\{2^{-i}, n^{-\gamma}\}. \quad (3.2.9)$$

For all i , since the $Z_{i,j}$'s are i.i.d. and $\mathbb{E}(Z_i) = m$ holds, by (3.2.9) we have

$$\text{Var}(Z_i) = \frac{\text{Var}(Z_{i,1})}{t_i p_i^{2k}} \leq \frac{k 2^{i(k-L)+k}}{t_i} \cdot \mathbb{E}(Z_i)^2 \cdot \max\{2^{-i}, n^{-\gamma}\}. \quad (3.2.10)$$

Now, recall from line 8 of [UncolApprox](#) that $\hat{L} := \hat{L}_i \in \mathbb{N}$ and $\hat{\gamma} := \hat{\gamma}_i \in [0, 1)$ are defined in such a way that $n^{\hat{L}+\hat{\gamma}} = 2^{ik}$ holds. Since $i \leq i^*$ we have $2^{ik} \leq m$, so either we have $\hat{L} = L$ and $\hat{\gamma} \leq \gamma$ or we have $\hat{L} \leq L-1$. We now claim that in either case, $2^{i(k-L)} \cdot \max\{2^{-i}, n^{-\gamma}\} \leq F_i$, where F_i is defined as in line 9 of [UncolApprox](#).

In the first case, where $\hat{L} = L$ and $\hat{\gamma} \leq \gamma$, we have:

$$2^{i(k-L)} \cdot \max\{2^{-i}, n^{-\gamma}\} \leq 2^{i(k-\hat{L})} \cdot \max\{2^{-i}, n^{-\hat{\gamma}}\} = F_i \quad (3.2.11)$$

as claimed. In the second case, we have $\hat{L} \leq L-1$ and $2^i = n^{(\hat{L}+\hat{\gamma})/k}$, so

$$\begin{aligned} 2^{i(k-L)} \cdot \max\{2^{-i}, n^{-\gamma}\} &\leq 2^{i(k-\hat{L}-1)} \cdot 1 = n^{(\hat{L}+\hat{\gamma})(k-\hat{L})/k} \cdot 2^{-i} \\ &\leq n^{(\hat{L}+\hat{\gamma})(k-\hat{L})/k} \cdot \max\{2^{-i}, n^{-\hat{\gamma}}\} = F_i \end{aligned} \quad (3.2.12)$$

again as claimed. Combining (3.2.11) and (3.2.12) and using our choice of t_i in line 10 of [UncolApprox](#), we can continue our calculation from (3.2.10) to arrive at our final variance bound of

$$\text{Var}(Z_i) \leq \frac{k2^k F_i}{t_i} \cdot \mathbb{E}(Z_i)^2 \leq \frac{\varepsilon^2 \cdot \mathbb{E}(Z_i)^2}{10k \log n}.$$

By Chebyshev's inequality, it follows that

$$\mathbb{P}\left(|Z_i - \mathbb{E}(Z_i)| \geq \varepsilon \mathbb{E}(Z_i)\right) \leq \frac{\text{Var}(Z_i)}{\varepsilon^2 \mathbb{E}(Z_i)^2} \leq \frac{1}{10k \log n}.$$

Since $\mathbb{E}(Z_i) = m = e(G)$, the claim follows from the definition of Z_i . \blacktriangleleft

We now prove correctness of [UncolApprox](#).

► Lemma 37. *With probability at least $2/3$, [UncolApprox](#)($\text{IND}(G), \varepsilon$) returns an ε -approximation to $e(G)$.*

Proof. If $\varepsilon < n^{-k/2}$ or $n \leq k^5$, then the exhaustive search in line 3 produces the exact number $e(G)$. Now suppose $\varepsilon \geq n^{-k/2}$ and $n \geq k^5$. We consider the following events for an execution of [UncolApprox](#)($\text{IND}(G), \varepsilon$):

\mathcal{E}_1 : For all $i \leq i^*$, either [UncolApprox](#) does not reach iteration i or the value $\frac{1}{p_i^k t_i} \sum_{j=1}^{t_i} e(G[U_{i,j}])$ is an ε -approximation to $e(G)$.

\mathcal{E}_2 : [SparseCount](#) never returns RTE, so that [UncolApprox](#) does not return RTE in line 19.

\mathcal{E}_3 : All calls to [SampleSubset](#) have combined runtime at most $R := C_{\text{sample}} \sum_{i=0}^{\log n-1} t_i (i + n/2^i)$, so that [UncolApprox](#) does not return RTE in line 12.

\mathcal{E}_4 : All sets $U_{i,j}$ satisfy $|U_{i,j}| \leq z_i := \max\{7p_i n, 7k \ln n\}$, so that [UncolApprox](#) does not return RTE in line 14.

Let $\mathcal{E} = \mathcal{E}_1 \cap \mathcal{E}_2 \cap \mathcal{E}_3 \cap \mathcal{E}_4$. We will first prove that $\mathbb{P}(\mathcal{E}) \geq 2/3$, and then show that if \mathcal{E} occurs then [UncolApprox](#) halts by iteration $i^*(G)$ and returns a valid ε -approximation of $e(G)$ as required.

For all $i \leq \log n - 1$, observe that $(k - \hat{L}_i)/k \leq 1$ and $\max\{2^{-i}, n^{-\hat{\gamma}_i}\} \leq 1$; thus $F_i \leq e(G) \leq n^k$. Since $\varepsilon \geq n^{-k/2}$ and $k \leq n^{1/5}$, it follows that

$$t_i \leq 12\varepsilon^{-2} k^2 2^k n^k \log n \leq 12n^{3k+2/5} \log n.$$

Thus [SparseCount](#) is called at most $12n^{5k}$ times in total. Since we call [SparseCount](#) with parameter $\delta = 1/(120n^{5k})$, it follows from Lemma 34 and a union bound that

$$\mathbb{P}(\mathcal{E}_1) \geq 9/10. \quad (3.2.13)$$

By Lemma 36 and a union bound over all i with $0 \leq i \leq i^* \leq \log n - 1$,

$$\mathbb{P}(\mathcal{E}_2) \geq 1 - \frac{i^* + 1}{10 \log n} \geq \frac{9}{10}. \quad (3.2.14)$$

Recall from Definition 27 that the expected running time of a call to `SampleSubset`(n, i) is at most $C_{\text{sample}}(i + n/2^i)$; thus the expected running time of all such calls is at most $C_{\text{sample}} \sum_{i=0}^{\log n - 1} t_i(i + n/2^i)$, so by Markov's inequality we have

$$\mathbb{P}(\mathcal{E}_3) \geq 19/20. \quad (3.2.15)$$

The number of queries that the algorithm makes to `SparseCount` and the number of sets $U_{i,j}$ is at most $\sum_{i=0}^{\log n - 1} t_i \leq n^k \log n \leq n^{2k}$. Moreover, by a Chernoff bound (Lemma 14), the probability that an individual $|U_{i,j}|$ is larger than z_i is at most $e^{-z_i} \leq n^{-7k}$. Thus a union bound yields

$$\mathbb{P}(\mathcal{E}_4) \geq 1 - n^{2k} \cdot n^{-7k} \geq 11/12.$$

It follows from (3.2.13), (3.2.14), (3.2.15) and a union bound that

$$\mathbb{P}(\mathcal{E}) \geq 2/3. \quad (3.2.16)$$

If \mathcal{E} occurs, then any iteration $i \leq i^*$ in which `UncolApprox` halts will return an ε -approximation of $e(G)$. (Indeed, `UncolApprox` cannot return RTE, so it halts in a given iteration if and only if `SparseCount` does not return TooDense. In this case we have $C_{i,j} = e(G[U_{i,j}])$ for all j , so since \mathcal{E}_1 occurs it outputs a valid ε -approximation.) We now claim that if \mathcal{E} occurs and `UncolApprox` reaches iteration i^* , then `SparseCount` does not return TooDense in iteration i^* . Given this claim, it follows immediately that `UncolApprox` halts by iteration $i^*(G)$ and returns a valid ε -approximation; thus the result follows from (3.2.16).

Suppose \mathcal{E} occurs, and that `UncolApprox` reaches iteration i^* . Then by the definition of i^* , we have $2^{-i^*k}m = p_{i^*}^k m \geq 1$ and $2^{-(i^*+1)k}m < 1$. This implies $p_{i^*}^k m \leq 2^k$. Since \mathcal{E}_1 occurs and $\varepsilon \leq 1$, we have

$$\sum_{j=1}^{t_{i^*}} e(G[U_{i^*,j}]) \leq p_{i^*}^k t_{i^*} \cdot 2m \leq 2^{k+1} t_{i^*} = M_{i^*}.$$

In iteration j of the inner for loop of `UncolApprox`, we will therefore have

$$M_{i^*,j} = M_{i^*} - \sum_{\ell=1}^{j-1} e(G[U_{i^*,\ell}]) \geq e(G[U_{i^*,j}]),$$

Thus the j^{th} `SparseCount` call sees at most $M_{i^*,j}$ edges and will thus avoid returning TooDense as required. This concludes the proof. \blacktriangleleft

3.2.4 Running time and oracle cost of `UncolApprox`

We now proceed to bound the running time and oracle cost of `UncolApprox`. In the following, $\text{cost}(|S|)$ will give the oracle cost of a set $S \subseteq V(G)$. Note that we do not require cost to be regularly-varying in the lemma below.

► **Lemma 38.** *Let $\text{cost} = \{\text{cost}_k : k \geq 2\}$ be an arbitrary cost function with parameter k . Suppose that $\text{cost}_k(x) \leq x^k$ for all k and x and that there exists x_0 such that for all k ,*

cost_k is non-decreasing on $[x_0, \infty)$. Let $T(G, \varepsilon)$ be the worst-case running time required to run `UncolApprox`($\text{IND}(G), \varepsilon$) and let $C(G, \varepsilon)$ be the worst-case oracle cost incurred under cost. Then for every n -vertex k -hypergraph G we have

$$T(G, \varepsilon) \leq \mathcal{O}(k^{5k} + \varepsilon^{-2} 2^{5k} \log^5 n \cdot n^{1+g(k,1)}), \quad (3.2.17)$$

$$C(G, \varepsilon) \leq \mathcal{O}\left(k^{7k} + \varepsilon^{-2} 2^{5k} \log^5 n \cdot \max_i (F_i \text{cost}_k(\max\{n/2^i, n^{1/4}\}))\right). \quad (3.2.18)$$

As a special case, if $\text{cost}_k(n) = 1$ for all $k, n \in \mathbb{N}$, then the bound on the cost implies that `UncolApprox` makes at most $\mathcal{O}(\varepsilon^{-2} 2^{5k} \log^5 n \cdot n^{k/4})$ queries to the oracle in the worst case.

Proof. We first consider the resource requirements of line 3 by splitting into three subcases.

Case 1: $n \leq k^5$. In this case, the running time and query count of exhaustive search are both $\mathcal{O}(\binom{n}{k}) \leq \mathcal{O}((en/k)^k) = \mathcal{O}(k^{5k})$, and each query has cost given by $\text{cost}_k(k) = \mathcal{O}(k^k)$. This is accounted for by the additive k^{5k} and k^{7k} terms on the right sides of Equations (3.2.17) and (3.2.18).

Case 2: $n^k \leq \varepsilon^{-2}$. In this case, the running time and query count of exhaustive search are both $\mathcal{O}(n^k) = \mathcal{O}(\varepsilon^{-2})$, and each query is of size k . The running time is therefore dominated by the second summand on the right side of Equation (3.2.17). Since $k \leq n$ and cost is eventually non-decreasing, we have $\text{cost}_k(k) = \mathcal{O}(\text{cost}_k(n))$; since $F_0 = 1$, it follows that the cost is dominated by the second summand on the right side of Equation (3.2.18).

Thus in all cases our bounds on running time and cost are satisfied, and we may assume $n^k > \varepsilon^{-2}$ and $n > k^5$ so that `UncolApprox` does not halt in line 3.

Observe that since n is a power of two, we can quickly calculate \hat{L}_i and $F_i = 2^{i(k-\hat{L}_i)}$, and hence also t_i and M_i ; we do not need to calculate $\hat{\gamma}_i$. The running time and cost are therefore dominated by lines 12 and 15.

For brevity, let $I = \log(n) - 1$. In line 12, we abort execution in line 12 if the running time gets too large, so the total running time of all calls to `SampleSubset` is at most

$$\sum_{i=0}^I t_i \cdot \mathcal{O}(ip_i n) = \mathcal{O}\left(n \log^2 n \cdot \max \{t_i p_i : i \in \{0, \dots, I\}\}\right).$$

Expanding via $t_i p_i = \lceil \varepsilon^{-2} 10 k^2 2^k \log n \cdot F_i \rceil 2^{-i}$ and using the bound $F_i 2^{-i} \leq n^{g(k,1)}$ from Lemma 31 (applied with $\beta = 1$), we immediately see that this is at most

$$\mathcal{O}(\varepsilon^{-2} k^2 2^k \log^3 n \cdot n^{1+g(k,1)}),$$

which is dominated by the second summand on the right side of Equation (3.2.17).

It remains to analyse line 15, starting with the $\min\{M_{i,j}, e(G[U_{i,j}])\}$ term in both the cost and the running time of `SparseCount` according to Lemma 34. This running time is dominated by the case in which `UncolApprox` does not halt early by returning RTE, so assume this is the case. Line 15 calls `SparseCount`($\text{IND}(G), U_{i,j}, k, M_{i,j}, n^{-5k}/120$), and the inner loop over j aborts once such a call returns TooDense. Let $j_i^* \in \{1, \dots, t_i\}$ be the last iteration of the inner loop. For all j with $j < j_i^*$, the return value is $C_{i,j} = e(G[U_{i,j}])$. It follows that

$$\begin{aligned} \sum_{j=0}^{j_i^*} \min\{M_{i,j}, e(G[U_{i,j}])\} &\leq \sum_{j=0}^{j_i^*-1} e(G[U_{i,j}]) + M_{i,j_i^*} \\ &= \sum_{j=0}^{j_i^*-1} e(G[U_{i,j}]) + \left(M_i - \sum_{j=0}^{j_i^*-1} C_{i,j}\right) = M_i. \end{aligned}$$

Moreover, since we do not return RTE, we have $|U_{i,j}| \leq \max\{7p_i n, 7k \ln n\} \leq n$ for all i . Thus by Lemma 34, the total running time of all iterations of line 15 is

$$\mathcal{O}\left(\sum_{i=0}^I \sum_{j=0}^{j^*} \log(n^{5k}/120) e^{2k} z_i \log^2 z_i \min\{M_{i,j}, e(G[U_{i,j}])\}\right) = \mathcal{O}\left(k e^{2k} \log^4 n \cdot \max_i(z_i M_i)\right).$$

Substituting in the definitions of z_i and M_i , we obtain a running time of at most

$$\mathcal{O}\left(\varepsilon^{-2} 2^{5k} \log^5 n \cdot \max_i (F_i \max\{n/2^i, k \ln n\})\right).$$

Recall that $k \leq n^{1/5}$, so $k \ln n \leq n^{1/4}$; it follows by Lemma 32, applied with $\beta = 1$, that this expression is dominated by the right side of Equation (3.2.17).

Observe that line 12 has no oracle cost, so we now bound the total oracle cost of line 15. Arguing exactly as with the running time, the total number of queries in the i^{th} iteration is at most

$$\mathcal{O}\left(\sum_{j=0}^{j^*} \log(n^{5k}/120) e^{2k} \log^2 z_i \min\{M_{i,j}, e(G[U_{i,j}])\}\right) = \mathcal{O}\left(\varepsilon^{-2} 2^{5k} \log^4 n \cdot F_i\right).$$

Since cost_k is eventually non-decreasing and $|U_{i,j}| \leq z_i$ for all j , the cost of any query in the i^{th} iteration is at most

$$\max_j (\text{cost}_k(|U_{i,j}|)) = \mathcal{O}(\text{cost}_k(z_i)) = \mathcal{O}(\text{cost}_k(\max\{n/2^i, k \ln n\})).$$

It follows that the total oracle cost over all iterations is at most

$$\begin{aligned} \mathcal{O}\left(\sum_{i=0}^I \varepsilon^{-2} 2^{5k} \log^4 n \cdot F_i \text{cost}_k(\max\{n/2^i, k \ln n\})\right) \\ = \mathcal{O}\left(\varepsilon^{-2} 2^{5k} \log^5 n \max_i (F_i \text{cost}_k(\max\{n/2^i, n^{1/4}\}))\right). \end{aligned}$$

This is dominated by the right side of Equation (3.2.18) as required. \blacktriangleleft

► **Theorem 39.** *Let $\text{cost} = \{\text{cost}_k : k \geq 2\}$ be a regularly-varying parameterised cost function with parameter k , index $\alpha_k \in [0, k]$, and slowly-varying component σ . There is a randomised IND-oracle algorithm $\text{Unco1}(\text{IND}(G), \varepsilon, \delta)$ with worst-case running time*

$$\mathcal{O}\left((k^{5k} + \varepsilon^{-2} 2^{5k} \log^5 n \cdot n^{1+g(k,1)}) \log(1/\delta)\right), \quad (3.2.19)$$

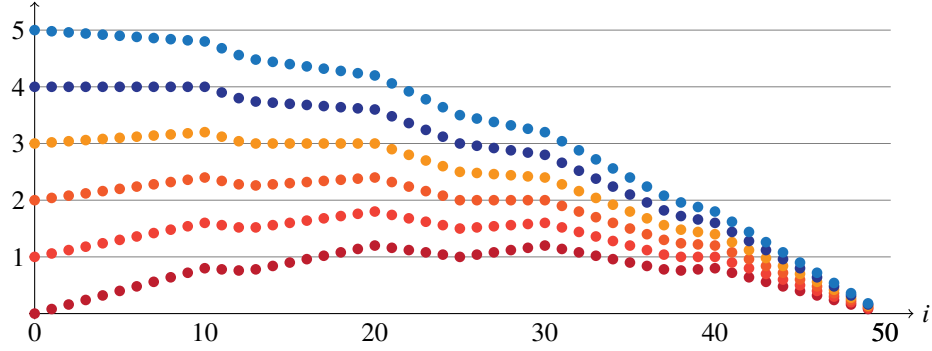
worst-case oracle cost

$$\mathcal{O}\left((k^{7k} + \varepsilon^{-2} 2^{5k} \log^5 n \cdot n^{g(k, \alpha_k) + o(1)} \text{cost}_k(n)) \log(1/\delta)\right), \quad (3.2.20)$$

and the following behaviour: Given an n -vertex k -hypergraph G and rationals $\varepsilon, \delta \in (0, 1)$, the algorithm outputs an integer m that, with probability at least $1 - \delta$, is an ε -approximation to $e(G)$.

Moreover, if either σ is eventually non-decreasing or there exists $\eta > 0$ such that $\alpha_k \geq k - 1 + \eta$ for all k , then the worst-case oracle cost is

$$\mathcal{O}\left((k^{7k} + \varepsilon^{-2} 2^{5k} \log^5 n \cdot n^{g(k, \alpha_k)} \text{cost}_k(n)) \log(1/\delta)\right). \quad (3.2.21)$$



■ **Figure 3** The expected overall cost of the i^{th} iteration of `UncolApprox` is depicted in the scenario where $n = 2^{50}$, $k = 5$, and the cost function is one of six functions with $\text{cost}(\tau) = \tau^\alpha$ for $\alpha \in \{0, \dots, k\}$. Plotted is the exponent g of the main n^g term of the cost of iteration i ; we have $g = \log_n(F_i 2^{-i\alpha} n^\alpha)$. As can be readily seen from the figure, increasing α shifts the most expensive iteration to the left. Thus, if we have prior knowledge about the rough number of edges, we may be able to improve the cost of our algorithm. For example, if we know that the graph is very dense so that $i^* \geq 0.8 \log n$ holds, then we can start the algorithm in iteration $i = 0.8 \log n$, which improves the cost. We may similarly improve the cost analysis if α is small compared to k and the graph is sparse, so that, for example, $i^* \leq 0.1 \log n$ holds. Interestingly, when $\alpha \geq k - 1$, the hardest instances seem to be the ones that are very sparse.

Proof. By Lemma 37, `UncolApprox` has success probability at least $2/3$. Applying standard boosting (Lemma 19) to `UncolApprox`, yields the algorithm with success probability $1 - \delta$ claimed here. By Lemma 12, the eventually non-decreasing cost assumption in Lemma 38 is satisfied and thus the worst-case bounds on running time (Equation (3.2.17)) and cost (Equation (3.2.18)) follow. The time bound in Equation (3.2.19) is identical to the one in Equation (3.2.17), the cost bound Equation (3.2.20) follows from Equation (3.2.18) by applying Corollary 33(i), and the cost bound Equation (3.2.21) follows from Equation (3.2.18) by applying Corollary 33(ii). ◀

3.3 Lower bounds on oracle algorithms for edge estimation

In this section, we unconditionally prove that the oracle cost achieved by `UncolApprox` is essentially optimal. We do this by proving a lower bound on the IND-decision tree complexity of approximate edge-counting. More specifically, we construct two correlated random n -vertex k -hypergraphs \mathcal{G}_1 and \mathcal{G}_2 that, with high probability, have a significantly different number of edges. This means that any approximate edge-counting algorithm \mathcal{A} will distinguish them with high probability. On the other hand, we also prove that any deterministic IND-oracle algorithm A that can distinguish between \mathcal{G}_1 and \mathcal{G}_2 must incur a large cost. We formalize this discussion as follows:

► **Theorem 40.** *Let $\text{cost}_k(n) = n^{\alpha_k}$, where $\alpha_k \in [0, k]$. Let n, k, r be integers with $\sqrt{n}/10^4 \geq k \geq r \geq \alpha_k \geq 1$. Let $\varepsilon \in (0, 1)$ satisfy $240k!/n^r \leq \varepsilon$. There exist two correlated distributions \mathcal{G}_1 and \mathcal{G}_2 on n -vertex k -hypergraphs with the following properties:*

- (i) *We have $\mathbb{P}_{(G_1, G_2) \sim (\mathcal{G}_1, \mathcal{G}_2)}[e(G_2) \geq (1 + \varepsilon)e(G_1)] \geq 0.95$.*

(ii) If A is a deterministic IND-oracle algorithm with

$$\mathbb{P}_{(G_1, G_2) \sim (\mathcal{G}_1, \mathcal{G}_2)} \left(A(\text{IND}(G_1)) \neq A(\text{IND}(G_2)) \right) \geq 2/3, \quad (3.3.1)$$

then the expected oracle cost of A (with respect to cost) under random inputs $G_1 \sim \mathcal{G}_1$ satisfies

$$\mathbb{E}_{G_1 \sim \mathcal{G}_1} [\text{cost}(A, G_1)] \geq n^{r(k-r+\alpha_k)/k} / (1080k^{3k} \varepsilon^{(r-\alpha_k)/k}).$$

Before we prove this theorem, let us apply the minimax theorem (Theorem 17) to it, in order to derive our main lower bound for IND-oracle algorithms.

► **Theorem 41.** Let $\text{cost}_k(n) = n^{\alpha_k}$, where $\alpha_k \in [0, k]$. Let \mathcal{A} be an IND-oracle algorithm such that for all k -hypergraphs G , $\mathcal{A}(\text{IND}(G))$ is a $\frac{1}{2}$ -approximation to $e(G)$ with probability at least $9/10$. Then \mathcal{A} has worst-case expected oracle cost at least $\Omega(n^{\alpha_k + g(k, \alpha_k)} / k^{3k})$ as $n \rightarrow \infty$, where k may depend on n in an arbitrary fashion.

Observe that unlike in Section 3.2, we do not require $\{\text{cost}_k : k \geq 2\}$ to be a regularly-varying parameterised cost function.

Proof. First suppose that $k > \sqrt{n}/10^4$, so that Theorem 40 does not apply. In this case, we have $k^{3k} = \Omega(n^{3k/2})$. Moreover, by Lemma 29 and since $\alpha_k \leq k$, we have

$$\alpha_k + g(k, \alpha_k) \leq \alpha_k + \frac{(k - \alpha_k)^2}{4k} = \frac{(k + \alpha_k)^2}{4k} \leq \frac{(2k)^2}{4k} = k.$$

It follows that $n^{\alpha_k + g(k, \alpha_k)} / k^{3k} = o(1)$. Since $\alpha_k \geq 0$, even a single query to $\text{IND}(G)$ requires $\Omega(1)$ cost and so the result is vacuously true.

For the rest of the proof, we may assume $k \leq \sqrt{n}/10^4$. Suppose G_1, G_2 are n -vertex k -hypergraphs that satisfy $e(G_2) \geq 3e(G_1)/2$. With probability at least $(9/10)^2 \geq 4/5$ over the random choices of \mathcal{A} , the algorithm \mathcal{A} will correctly output a $(1/2)$ -approximation for both graphs G_1 and G_2 . This implies that \mathcal{A} distinguishes G_1 and G_2 in the sense that the following holds:

$$\mathbb{P}_{A \sim \mathcal{A}} \left(A(\text{IND}(G_1)) \neq A(\text{IND}(G_2)) \right) \geq 4/5.$$

By a union bound, taking $\varepsilon = 1/2$ in Theorem 40i implies

$$\mathbb{P}_{(G_1, G_2) \sim (\mathcal{G}_1, \mathcal{G}_2)} \left(A(\text{IND}(G_1)) \neq A(\text{IND}(G_2)) \right) \geq 3/4. \quad (3.3.2)$$

Let F be the family of IND-oracle algorithms A that satisfy (3.3.1). By (3.3.2), we have

$$\begin{aligned} \frac{3}{4} &\leq \mathbb{P}_{(G_1, G_2) \sim (\mathcal{G}_1, \mathcal{G}_2)} \left(A(\text{IND}(G_1)) \neq A(\text{IND}(G_2)) \right) \leq \mathbb{P}_{A \sim \mathcal{A}} (A \in F) + \frac{2}{3} \mathbb{P}_{A \sim \mathcal{A}} (A \notin F) \\ &= \frac{1}{3} \mathbb{P}_{A \sim \mathcal{A}} (A \in F) + \frac{2}{3}, \end{aligned}$$

and so $\mathbb{P}_{A \sim \mathcal{A}} (A \in F) \geq 1/4$. Thus by minimax (i.e. Theorem 17 taking $p = 1/4$), we can lower-bound the worst-case expected oracle cost of \mathcal{A} by the worst-case expected oracle cost of any deterministic $A \in F$:

$$\max_G \mathbb{E}_{A \sim \mathcal{A}} [\text{cost}(A, G)] \geq \frac{1}{4} \inf_{A \in F} \mathbb{E}_{G_1 \sim \mathcal{G}_1} [\text{cost}(A, G_1)].$$

By [Theorem 40ii](#), it follows that

$$\max_G \mathbb{E}_{A \sim \mathcal{A}} [\text{cost}(A, G)] = \Omega(n^{r(k-r+\alpha_k)}/k^{3k}).$$

The result now follows immediately from [Lemma 32](#). ◀

3.3.1 \mathcal{G}_1 and \mathcal{G}_2 : Choosing a hard input distribution

Our first step in the proof of [Theorem 40](#) shall be to define the random graphs \mathcal{G}_1 and \mathcal{G}_2 and prove that $e(\mathcal{G}_2) \geq (1 + \varepsilon)e(\mathcal{G}_1)$ holds with probability at least 0.95.

Recall from the discussion in [Section 3.2.2](#) that our algorithmic approach is to randomly sample induced subgraphs and apply the independence oracle to these induced subgraphs. Recall also that one of the possible “worst cases” in which the number of samples required to see an edge is maximised is a single-rooted star graph, in which all edges intersect in a single r -vertex “root”. Consider the effects of combining an Erdős-Rényi k -hypergraph H_1 with a single-rooted star graph H_2 of similar density. Intuitively, we would expect that any large random set S is likely to contain an edge from H_1 , so independence oracles will return the same result for $H_1[S]$ and $(H_1 \cup H_2)[S]$. At the same time, any small random set S is unlikely to contain all of R , and will therefore also not distinguish H_1 from $H_1 \cup H_2$. This motivates our choice of \mathcal{G}_1 and \mathcal{G}_2 , which we define as follows; note that for technical reasons we allow H_2 to contain multiple roots.

► **Definition 42.** Let n, k, r be integers with $\sqrt{n}/10^4 \geq k \geq r \geq \alpha_k \geq 1$. Let $\varepsilon \in (0, 1)$ satisfy $240k!/n^r \leq \varepsilon$. We define the following probabilities:

$$p_1(n, k, r, \varepsilon) := k!/(\varepsilon n^r), \quad p_2(n, k, r, \varepsilon) := 240r!/n^r.$$

We define $H_1(n, k, r, \varepsilon)$ to be an Erdős-Rényi random k -hypergraph on the vertex set $[n]$, where each edge occurs independently with probability $p_1(n, k, r, \varepsilon)$. We define $\mathcal{R}(n, k, r, \varepsilon)$ to be a random subset of $[n]^{(r)}$, where each size- r set is included independently with probability $p_2(n, k, r, \varepsilon)$; we refer to these size- r sets as roots. We define $H_2(n, k, r, \varepsilon)$ to be the k -hypergraph with edge set

$$\left\{ e \in [n]^{(k)} : e \supseteq R \text{ for some } R \in \mathcal{R} \right\}.$$

Finally, we define

$$\begin{aligned} \mathcal{G}_1(n, k, r, \varepsilon) &= H_1(n, k, r, \varepsilon), \\ \mathcal{G}_2(n, k, r, \varepsilon) &= H_1(n, k, r, \varepsilon) \cup H_2(n, k, r, \varepsilon). \end{aligned}$$

Generally the values of n, k, r and ε will be clear from context, and in this case we will omit them from the notation.

The following lemma establishes [Theorem 40i](#) for \mathcal{G}_1 and \mathcal{G}_2 .

► **Lemma 43.** Let n, k, r be integers with $\sqrt{n}/10^4 \geq k \geq r \geq \alpha_k \geq 1$. Let $\varepsilon \in (0, 1)$ satisfy $240k!/n^r \leq \varepsilon$. Then with probability at least 0.95, we have $e(\mathcal{G}_2) > (1 + \varepsilon)e(\mathcal{G}_1)$.

Proof. Observe that

$$\frac{|e(\mathcal{G}_2) - e(\mathcal{G}_1)|}{e(\mathcal{G}_1)} = \frac{e(H_2) - |E(H_1) \cap E(H_2)|}{e(H_1)}. \quad (3.3.3)$$

To prove the lemma, we must prove that the left side of (3.3.3) is larger than ε with probability at least 0.95; to this end, we use Chernoff bounds to bound $e(H_1)$ and $|E(H_1) \cap E(H_2)|$ above, and $e(H_2)$ below.

Upper bound on $e(H_1)$. By linearity of expectation, $\mathbb{E}(e(H_1)) = p_1 \binom{n}{k} \leq p_1 n^k / k! = n^{k-r} / \varepsilon$. Since $\varepsilon < 1$, we have $n^{k-r} / \varepsilon > n^{k-r} \geq 1$. By the Chernoff bound of Lemma 14 applied with $z = 7n^{k-r} / \varepsilon$, we have

$$\mathbb{P}(e(H_1) \leq 7n^{k-r} / \varepsilon) \geq 1 - e^{-7n^{k-r} / \varepsilon} \geq 1 - e^{-7} \geq 0.99. \quad (3.3.4)$$

Upper bound on $|E(H_1) \cap E(H_2)|$ when $e(H_2) \geq 30n^{k-r}$. Any edge of H_2 is included in $E(H_1)$ independently with probability p_1 . By assumption on n, r, ε , we have $p_1 \leq 1/240 \leq 1/14$, so conditioned on $e(H_2) = M$ for any $M \in \mathbb{N}$, we have

$$\mathbb{E}[|E(H_1) \cap E(H_2)| \mid e(H_2) = M] = p_1 M \leq M/14.$$

By the Chernoff bound of Lemma 14 applied with $z = M/2$, we have for all $M \geq 30n^{k-r}$:

$$\mathbb{P}(|E(H_1) \cap E(H_2)| \leq e(H_2)/2 \mid e(H_2) = M) \geq 1 - e^{-7M/14} \geq 1 - e^{-15n^{k-r}} \geq 0.99. \quad (3.3.5)$$

Lower bound on $e(H_2)$. Let Z be the set of all edges of H_2 that contain exactly one root $R \in \mathcal{R}$, and let \mathcal{E} be the event that all roots $R \in \mathcal{R}$ are disjoint from each other. We have

$$\text{when } \mathcal{E} \text{ occurs, } e(H_2) \geq |Z| = |\mathcal{R}| \cdot \binom{n-r}{k-r}, \quad (3.3.6)$$

Note that $|\mathcal{R}|$ is binomially distributed, and since $\sqrt{n}/10 \geq k \geq r$ we have

$$\mathbb{E}(|\mathcal{R}|) = p_2 \binom{n}{r} = \frac{p_2}{r!} \prod_{i=0}^{r-1} (n-i) \geq \frac{p_2}{r!} \left(n^r - \sum_{i=0}^{r-1} i \right) \geq \frac{p_2 n^r}{4r!} \geq 30.$$

Moreover, $\mathbb{E}(|\mathcal{R}|) \leq p_2 n^r / r! = 240$. By the Chernoff bound of Lemma 13 applied with $\delta = 1/2$, it follows that

$$\mathbb{P}(30 \leq |\mathcal{R}| \leq 360) \geq 1 - 2e^{-\mathbb{E}(|\mathcal{R}|)/12} \geq 1 - 2e^{-5} \geq 0.98. \quad (3.3.7)$$

Moreover, to bound $\mathbb{P}(\mathcal{E})$, we observe that there are exactly $\binom{n}{r} \binom{r}{i} \binom{n-r}{r-i}$ pairs (S, T) of size- r sets S and T with $|S \cap T| = i$. A union bound over all possible intersecting pairs of distinct roots yields

$$\mathbb{P}(\mathcal{E}) \leq \sum_{i=1}^{r-1} \binom{n}{r} \binom{r}{i} \binom{n-r}{r-i} \cdot p_2^2 \leq \sum_{i=1}^{r-1} \frac{n^r \cdot r^i \cdot n^{r-i}}{r! \cdot i! \cdot (r-i)!} \cdot \frac{(240r!)^2}{n^{2r}} = 240^2 \sum_{i=1}^{r-1} \frac{r^i r!}{n^i i! (r-i)!}.$$

Observe that $r^2/n \leq k^2/n < 10^{-8}$; thus

$$\mathbb{P}(\mathcal{E}) \leq 240^2 \sum_{i=1}^{r-1} (r^2/n)^i \leq 240^2 \sum_{i=1}^{\infty} (r^2/n)^i = 240^2 \cdot \left(\frac{1}{1 - r^2/n} - 1 \right) \leq 240^2 \cdot 2r^2/n < 0.01.$$

Combining this with Equations (3.3.6) and (3.3.7) and a union bound, we obtain

$$\mathbb{P}\left(e(H_2) \geq 30 \binom{n-360r}{k-r}\right) \geq \mathbb{P}(\mathcal{E} \text{ and } 30 \leq |\mathcal{R}| \leq 360) \geq 1 - (0.01 + 0.02) = 0.97.$$

Combined with $\binom{n-360r}{k-r} \leq n^{k-r}$, we arrive at

$$\mathbb{P}(e(H_2) \geq 30n^{k-r}) \geq 0.97. \quad (3.3.8)$$

Conclusion of proof. Combining Equations (3.3.5) and (3.3.8), we get

$$\mathbb{P}(e(H_2) \geq 30n^{k-r} \text{ and } |E(H_1) \cap E(H_2)| \leq e(H_2)/2) \geq 0.99 \cdot 0.97 \geq 0.96.$$

Combined with Equation (3.3.4) and a union bound, all three bounds are likely to hold:

$$\mathbb{P}(e(H_2) \geq 30n^{k-r} \text{ and } |E(H_1) \cap E(H_2)| \leq e(H_2)/2 \text{ and } e(H_1) \leq 7n^{k-r}/\varepsilon) \geq 0.95.$$

By Equation (3.3.3), we arrive at the following with probability at least 0.95:

$$\frac{|e(\mathcal{G}_2) - e(\mathcal{G}_1)|}{e(\mathcal{G}_1)} = \frac{e(H_2) - |E(H_1) \cap E(H_2)|}{e(H_1)} \geq \frac{15n^{k-r}}{7n^{k-r}/\varepsilon} > \varepsilon.$$

The result therefore follows. \blacktriangleleft

3.3.2 Bounding the cost of separating \mathcal{G}_1 and \mathcal{G}_2

Throughout this section, let n, k, r, ε and A be as in the statement of Theorem 40, and let $\mathcal{G}_1, \mathcal{G}_2, H_1$ and H_2 be as in Definition 42. Note that the number of queries carried out by A on \mathcal{G}_1 is a random variable. This would lead to some uninteresting and unpleasant technicalities in the probability bounds, so we first make some easy observations that allow us to assume this number is deterministic.

► Remark 44.

- (i) Without loss of generality, A only ever queries sets which are either empty or have size at least k .
- (ii) For all vertex sets S with $|S| \geq k$, we have:
 - $\text{IND}(\mathcal{G}_1)_S = 1$ if and only if S contains no edge of H_1 .
 - $\text{IND}(\mathcal{G}_2)_S = 0$ if and only if S contains an edge of H_1 or a root from \mathcal{R} .
- (iii) Without loss of generality, we may assume that there is some $t = t_{n,k} \in \mathbb{N}$ such that A makes exactly the same number t of queries for each n -vertex k -hypergraph oracle input.

Proof. To see i, recall from Section 2.1.2 that the IND-oracle algorithm A receives n and k as explicit inputs. Any time A is about to query a set S of size less than k , it could instead avoid performing the query and correctly assume that the answer is $\text{IND}(G)_S = 1$. Modifying A in this way can only reduce the cost of A .

ii then follows directly from the definitions of $\mathcal{G}_1, \mathcal{G}_2$, and IND, even when $S = \emptyset$.

To see iii, first observe that we can enumerate the edges of an n -vertex k -hypergraph G using the oracle by querying all size- k subsets of $[n]$ with total oracle cost $n^k k^{\alpha_k} \leq (nk)^k$. Thus without loss of generality, we may assume $\text{cost}(A, G) \leq 3(nk)^k$ for all n -vertex k -hypergraphs G , by running A in parallel with the simple enumeration algorithm and returning the result of whichever algorithm finishes first. Under this assumption, A will always carry out $x_G < 4(nk)^k$ queries on G before returning, and we simply “pad” A by inserting $4(nk)^k - x_G$ zero-cost queries to the empty set before outputting the final answer. \blacktriangleleft

We now set out some notation to describe the sequence of oracle queries executed by A on a given input graph. Recall from Section 2.1.2 that A is explicitly given n and k as input, and can access G only by querying individual bits of $\text{IND}(G)$.

► **Definition 45.** Let G be an arbitrary n -vertex k -hypergraph. Let $q(G)$ be the sequence of oracle queries that A makes when given input $\text{IND}(G)$, and write $q(G) = (S_1(G), \dots, S_t(G))$. For all $i \in [t]$, let $b_i(G) = \text{IND}(G)_{S_i}$ be the bit returned by the i^{th} query, and let $b(G) = (b_1(G), \dots, b_t(G))$. We refer to $(q(G), b(G))$ as the transcript of A on input G . For all $i \in [t]$, let $q_{<i}(G) = (S_1(G), \dots, S_{i-1}(G))$ and $b_{<i}(G) = (b_1(G), \dots, b_{i-1}(G))$.

Observe that since A is a general deterministic oracle algorithm, the i^{th} query it makes will be a function of the responses to its first $i - 1$ queries. As such, the total oracle cost of running A on $\text{IND}(G)$ is a function of $b(G)$. We now define notation for these concepts.

► **Definition 46.** For all $i \leq t$ and all bit strings $\vec{b} \in \{0, 1\}^t$, let $S_i(\vec{b})$ be the i^{th} query that A makes after it obtained the responses b_1, \dots, b_{i-1} to the first $i - 1$ queries. (Note in particular that $S_i(\vec{b})$ is a deterministic function of \vec{b} .) For all $C > 0$, let X_C be the set of all possible response vectors that lead to a total query cost of at most C for A , that is,

$$X_C = \left\{ \vec{b} \in \{0, 1\}^t : \sum_{i=1}^t \text{cost}_k(S_i(\vec{b})) \leq C \text{ and } \vec{b} = b(G) \text{ for some } G \right\}.$$

Finally, write $\mathcal{I}(\vec{b}) = \{i \in [t] : S_i(\vec{b}) \neq \emptyset\}$, so that $|S_i(\vec{b})| \geq k$ for all $i \in \mathcal{I}(\vec{b})$ by Remark 44i.

We now state and prove Lemma 47, which is the heart of the proof of Theorem 40ii and bounds the probability of distinguishing between \mathcal{G}_1 and \mathcal{G}_2 with queries of a given cost C above in terms of the sizes of these queries. Turning this into an upper bound in terms of C then requires optimising over possible query sizes; we do this in the final proof of Theorem 40 at the end of this section.

The intuition behind Lemma 47 is that A can only distinguish between \mathcal{G}_1 and \mathcal{G}_2 if it ever makes a query that distinguishes them, that is, if $b(\mathcal{G}_1) \neq b(\mathcal{G}_2)$ holds. We will consider queries individually. If a query contains too many vertices, then it is very likely to include an edge of H_1 , which is thus contained in both \mathcal{G}_1 and \mathcal{G}_2 ; if a query contains too few vertices, then it is unlikely to pick up a root of H_2 , in which case it will have the same result in both \mathcal{G}_1 and \mathcal{G}_2 .

It is important to note that these queries are heavily dependent on each other — indeed, independence would correspond to the special case of a *non-adaptive* algorithm A , which must always use the same sequence of queries regardless of the responses. As such, turning this idea into a proof requires careful handling of the conditioning on past queries.

► **Lemma 47.** For all $C > 0$, we have

$$\begin{aligned} \mathbb{P} \left(A(\text{IND}(\mathcal{G}_1)) \neq A(\text{IND}(\mathcal{G}_2)) \text{ and } \sum_{i=1}^t \text{cost}_k(S_i(\mathcal{G}_1)) \leq C \right) \\ \leq 4^k k^k p_2 \max_{\vec{b} \in X_C} \sum_{i \in \mathcal{I}(\vec{b})} \min \left\{ |S_i(\vec{b})|^r, \frac{1}{p_1 |S_i(\vec{b})|^{k-r}} \right\}. \end{aligned}$$

Proof. For convenience, we define shorthand for the event of the lemma statement:

$$\mathcal{E} \text{ occurs when } A(\text{IND}(\mathcal{G}_1)) \neq A(\text{IND}(\mathcal{G}_2)) \text{ and } \sum_{j=1}^t \text{cost}_k(S_j(\mathcal{G}_1)) \leq C.$$

Recall that $\sum_{j=1}^t \text{cost}_k(S_j(\mathcal{G}_1)) \leq C$ if and only if $b(\mathcal{G}_1) \in X_C$, and that $A(\text{IND}(\mathcal{G}_1)) \neq A(\text{IND}(\mathcal{G}_2))$ can only happen when $b(\mathcal{G}_1) \neq b(\mathcal{G}_2)$ since A is deterministic. Thus

$$\mathbb{P}(\mathcal{E}) \leq \mathbb{P} \left(b(\mathcal{G}_1) \in X_C \text{ and } \bigvee_{i=1}^t \{b_i(\mathcal{G}_1) \neq b_i(\mathcal{G}_2)\} \right).$$

If $\mathbb{P}(b(\mathcal{G}_1) \in X_C) = 0$ then the result is immediate, so we may assume without loss of generality that $\mathbb{P}(b(\mathcal{G}_1) \in X_C) > 0$. It follows that

$$\mathbb{P}(\mathcal{E}) \leq \mathbb{P}\left(\bigvee_{i=1}^t \left\{b_i(\mathcal{G}_1) \neq b_i(\mathcal{G}_2)\right\} \mid b(\mathcal{G}_1) \in X_C\right). \quad (3.3.9)$$

We next bound $\mathbb{P}(\mathcal{E})$ in terms of the probabilities of distinguishing between \mathcal{G}_1 and \mathcal{G}_2 at each query. We apply [Lemma 16](#) to the event of (3.3.9), taking $Z_i = (b_i(\mathcal{G}_1), b_i(\mathcal{G}_2))$, taking \mathcal{E}_i to be the event that $b_i(\mathcal{G}_1) \neq b_i(\mathcal{G}_2)$, and working conditioned on $\vec{b}(\mathcal{G}_1) \in X_C$. Observe that the condition on the sets \mathcal{Z}_i of the lemma is satisfied — indeed, any transcript with $b_{<i}(\mathcal{G}_1) = b_{<i}(\mathcal{G}_2)$ has non-zero probability to be extended to a transcript with $b_{<t}(\mathcal{G}_1) = b_{<t}(\mathcal{G}_2)$, since we have $\mathcal{R} = \emptyset$ with non-zero probability and, conditioned on $\mathcal{R} = \emptyset$, we have $b(\mathcal{G}_1) = b(\mathcal{G}_2)$ with certainty. Thus [Lemma 16](#) applies, and we have

$$\mathbb{P}(\mathcal{E}) \leq \max_{\vec{b} \in X_C} \sum_{i=1}^t \mathbb{P}\left(b_i(\mathcal{G}_1) \neq b_i(\mathcal{G}_2) \mid b_{<i}(\mathcal{G}_1) = b_{<i}(\mathcal{G}_2) = (b_1, \dots, b_{i-1})\right).$$

For brevity, for any bit string $\vec{b} = (b_1, \dots, b_i)$, we define the event $\mathcal{B}_i(\vec{b})$ to occur when $b_{<i}(\mathcal{G}_1) = b_{<i}(\mathcal{G}_2) = (b_1, \dots, b_{i-1})$. Moreover, conditioned on $\mathcal{B}_i(\vec{b})$ we have $S_i(\mathcal{G}_1) = S_i(\mathcal{G}_2) = S_i(\vec{b})$, so since $E(\mathcal{G}_1) \subseteq \mathcal{E}(\mathcal{G}_2)$ the event $b_i(\mathcal{G}_1) \neq b_i(\mathcal{G}_2)$ can only occur if $b_i(\mathcal{G}_1) = 1$ and $b_i(\mathcal{G}_2) = 0$. Hence,

$$\mathbb{P}(\mathcal{E}) \leq \max_{\vec{b} \in X_C} \sum_{i=1}^t \mathbb{P}\left(b_i(\mathcal{G}_1) = 0 \text{ and } b_i(\mathcal{G}_2) = 1 \mid \mathcal{B}_i(\vec{b})\right).$$

Further, note that conditioned on $\mathcal{B}_i(\vec{b})$, if $S_i(\vec{b}) = \emptyset$ then we have $b_i(\mathcal{G}_1) = b_i(\mathcal{G}_2) = b_i = 1$ with certainty. It follows that

$$\mathbb{P}(\mathcal{E}) \leq \max_{\vec{b} \in X_C} \sum_{i \in \mathcal{I}(\vec{b})} \mathbb{P}\left(b_i(\mathcal{G}_1) = 0 \text{ and } b_i(\mathcal{G}_2) = 1 \mid \mathcal{B}_i(\vec{b})\right).$$

By [Remark 44ii](#), we have $b_i(\mathcal{G}_1) = 1$ if and only if $S_i(\vec{b})^{(k)} \cap E(H_1) = \emptyset$, and we have $b_i(\mathcal{G}_2) = 0$ if and only if either $S_i(\vec{b})^{(k)} \cap E(H_1) \neq \emptyset$ or $S_i(\vec{b})^{(r)} \cap \mathcal{R} \neq \emptyset$. This implies

$$\mathbb{P}(\mathcal{E}) \leq \max_{\vec{b} \in X_C} \sum_{i \in \mathcal{I}(\vec{b})} \mathbb{P}\left((S_i(\vec{b})^{(k)} \cap E(H_1) = \emptyset) \text{ and } (S_i(\vec{b})^{(r)} \cap \mathcal{R} \neq \emptyset) \mid \mathcal{B}_i(\vec{b})\right). \quad (3.3.10)$$

We next decompose $\mathcal{B}_i(\vec{b})$ into a conjunction of events depending either only on H_1 or only on \mathcal{R} ; this will allow us to split each summand of (3.3.10) into two independent events. We define

$$\begin{aligned} \mathcal{B}_{i,1}^-(\vec{b}) &:= \bigwedge_{\substack{j \leq i-1 \\ b_j=1}} (S_i(\vec{b})^{(k)} \cap E(H_1) = \emptyset), \\ \mathcal{B}_{i,1}^+(\vec{b}) &:= \bigwedge_{\substack{j \leq i-1 \\ b_j=0}} (S_i(\vec{b})^{(k)} \cap E(H_1) \neq \emptyset), \\ \mathcal{B}_{i,2}(\vec{b}) &:= \bigwedge_{\substack{j \leq i-1 \\ b_j=1}} (S_i(\vec{b})^{(r)} \cap \mathcal{R} = \emptyset). \end{aligned}$$

Then we have $\mathcal{B}_i(\vec{b}) = \mathcal{B}_{i,1}^-(\vec{b}) \wedge \mathcal{B}_{i,1}^+(\vec{b}) \wedge \mathcal{B}_{i,2}(\vec{b})$, where $\mathcal{B}_{i,1}^-(\vec{b}) \wedge \mathcal{B}_{i,1}^+(\vec{b})$ depends only on H_1 and $\mathcal{B}_{i,2}(\vec{b})$ depends only on H_2 . By (3.3.10), it follows that

$$\mathbb{P}(\mathcal{E}) \leq \max_{\vec{b} \in X_C} \sum_{i \in \mathcal{I}(\vec{b})} \mathbb{P}\left(S_i(\vec{b})^{(k)} \cap E(H_1) = \emptyset \mid \mathcal{B}_{i,1}^-(\vec{b}) \wedge \mathcal{B}_{i,1}^+(\vec{b})\right) \mathbb{P}\left(S_i(\vec{b})^{(r)} \cap \mathcal{R} \neq \emptyset \mid \mathcal{B}_{i,2}(\vec{b})\right). \quad (3.3.11)$$

We next deal with the “negative” conditioning. For all $x \in \{0, \dots, k\}$, we define

$$F_i^x(\vec{b}) = S_i(\vec{b})^{(x)} \setminus \bigcup_{\substack{j \in [i-1] \\ b_j=1}} S_j(\vec{b})^{(x)}.$$

Observe that $F_i^k(\vec{b})$ is precisely the set of size- k subsets of $S_i(\vec{b})$ which can still be edges of H_1 conditioned on $\mathcal{B}_{i,1}^-(\vec{b})$, and that $F_i^r(\vec{b})$ is precisely the set of size- r subsets of $S_i(\vec{b})$ which can still be roots in \mathcal{R} conditioned on $\mathcal{B}_{i,2}(\vec{b})$. It follows from (3.3.11) that

$$\mathbb{P}(\mathcal{E}) \leq \max_{\vec{b} \in X_C} \sum_{i \in \mathcal{I}(\vec{b})} \mathbb{P}\left(F_i^k(\vec{b}) \cap E(H_1) = \emptyset \mid \mathcal{B}_{i,1}^+(\vec{b})\right) \mathbb{P}\left(F_i^r(\vec{b}) \cap \mathcal{R} \neq \emptyset\right). \quad (3.3.12)$$

We next deal with the “positive” conditioning. Observe that the indicator function of $\mathcal{B}_{i,1}^+(\vec{b})$ is a monotonically increasing function of the indicator variables of H_1 ’s edges, and the indicator function of $F_i^k(\vec{b}) \cap E(H_1) = \emptyset$ is a monotonically decreasing function of these variables. Thus the two events are negatively correlated and therefore, by the FKG inequality (Lemma 15) combined with (3.3.12), we obtain

$$\begin{aligned} \mathbb{P}(\mathcal{E}) &\leq \max_{\vec{b} \in X_C} \sum_{i \in \mathcal{I}(\vec{b})} \mathbb{P}\left(F_i^k(\vec{b}) \cap E(H_1) = \emptyset\right) \mathbb{P}\left(F_i^r(\vec{b}) \cap \mathcal{R} \neq \emptyset\right) \\ &= \max_{\vec{b} \in X_C} \sum_{i \in \mathcal{I}(\vec{b})} (1 - p_1)^{|F_i^k(\vec{b})|} \left(1 - (1 - p_2)^{|F_i^r(\vec{b})|}\right) \leq \max_{\vec{b} \in X_C} \sum_{i \in \mathcal{I}(\vec{b})} e^{-p_1 |F_i^k(\vec{b})|} p_2 |F_i^r(\vec{b})|. \end{aligned} \quad (3.3.13)$$

By definition, we have $|F_i^r(\vec{b})| \leq |S_i(\vec{b})^{(r)}| \leq |S_i(\vec{b})|^r$, and so we can bound each term in the right-hand side of (3.3.13) by

$$e^{-p_1 |F_i^k(\vec{b})|} p_2 |F_i^r(\vec{b})| \leq p_2 |S_i(\vec{b})|^r. \quad (3.3.14)$$

We also provide a second upper bound on each term which will be stronger when $|S_i(\vec{b})|$ is large. To this end, we double-count the set Z of pairs $(X, Y) \in F_i^r(\vec{b}) \times F_i^k(\vec{b})$ with $X \subseteq Y$. Each size- k set $Y \in F_i^k(\vec{b})$ contains at most $\binom{k}{r}$ size- r sets $X \in F_i^r(\vec{b})$, and each size- r set $X \in F_i^r(\vec{b})$ is contained in exactly $\binom{|S_i(\vec{b})|-r}{k-r}$ size- k sets $Y \in F_i^k(\vec{b})$; hence we obtain

$$\left(\binom{|S_i(\vec{b})|-r}{k-r}\right) |F_i^r(\vec{b})| = |Z| \leq \binom{k}{r} |F_i^k(\vec{b})| \leq 2^k |F_i^k(\vec{b})|. \quad (3.3.15)$$

(Recall that for all \vec{b} and all $i \in \mathcal{I}(\vec{b})$, we have $|S_i(\vec{b})| \geq k$.) By Lemma 23 (taking $a = k$ and $b = k - r$), we have $\binom{|S_i(\vec{b})|-r}{k-r} \geq |S_i(\vec{b})|^{k-r} / (2k)^k$. Rearranging terms in (3.3.15) yields

$$|F_i^r(\vec{b})| \leq \frac{4^k k^k |F_i^k(\vec{b})|}{|S_i(\vec{b})|^{k-r}}. \quad (3.3.16)$$

Using this, we can bound each term of (3.3.13) as follows:

$$e^{-p_1|F_i^k(\vec{b})|} p_2 |F_i^r(\vec{b})| \stackrel{(3.3.16)}{\leq} 4^k k^k p_2 \cdot \frac{e^{-p_1|F_i^k(\vec{b})|} \cdot |F_i^k(\vec{b})|}{|S_i(\vec{b})|^{k-r}} \leq 4^k k^k p_2 \cdot \frac{1}{p_1 |S_i(\vec{b})|^{k-r}}. \quad (3.3.17)$$

The second inequality here follows by observing that the expression xe^{-p_1x} is maximised by $1/(p_1e)$ at $x = 1/p_1$. Thus by applying the bounds of (3.3.14) and (3.3.17) to the terms of (3.3.13), we arrive at the claimed bound of

$$\mathbb{P}(\mathcal{E}) \leq 4^k k^k p_2 \max_{\vec{b} \in X_C} \sum_{i \in \mathcal{I}(\vec{b})} \min \left\{ |S_i(\vec{b})|^r, \frac{1}{p_1 |S_i(\vec{b})|^{k-r}} \right\}. \quad \blacktriangleleft$$

We are now ready to prove [Theorem 40](#), from which our main result [Theorem 41](#) follows as discussed earlier at the start of the section.

► **Theorem 40.** *Let $\text{cost}_k(n) = n^{\alpha_k}$, where $\alpha_k \in [0, k]$. Let n, k, r be integers with $\sqrt{n}/10^4 \geq k \geq r \geq \alpha_k \geq 1$. Let $\varepsilon \in (0, 1)$ satisfy $240k!/n^r \leq \varepsilon$. There exist two correlated distributions \mathcal{G}_1 and \mathcal{G}_2 on n -vertex k -hypergraphs with the following properties:*

- (i) *We have $\mathbb{P}_{(G_1, G_2) \sim (\mathcal{G}_1, \mathcal{G}_2)}[e(G_2) \geq (1 + \varepsilon)e(G_1)] \geq 0.95$.*
- (ii) *If A is a deterministic IND-oracle algorithm with*

$$\mathbb{P}_{(G_1, G_2) \sim (\mathcal{G}_1, \mathcal{G}_2)} \left(A(\text{IND}(G_1)) \neq A(\text{IND}(G_2)) \right) \geq 2/3, \quad (3.3.1)$$

then the expected oracle cost of A (with respect to cost) under random inputs $G_1 \sim \mathcal{G}_1$ satisfies

$$\mathbb{E}_{G_1 \sim \mathcal{G}_1} [\text{cost}(A, G_1)] \geq n^{r(k-r+\alpha_k)/k} / (1080k^{3k} \varepsilon^{(r-\alpha_k)/k}).$$

Proof. We take \mathcal{G}_1 and \mathcal{G}_2 as in [Definition 42](#); these satisfy [i](#) by [Lemma 43](#). It remains to prove [ii](#).

Observe that by Markov's inequality, with probability at least $2/3$, $A(\text{IND}(\mathcal{G}_1))$ uses queries with total cost at most $3Z$. Write \mathcal{E} for the event that $A(\text{IND}(\mathcal{G}_1)) \neq A(\text{IND}(\mathcal{G}_2))$ and $\sum_{i=1}^t \text{cost}(S_i(\mathcal{G}_1)) \leq 3Z$; it follows by a union bound that

$$\mathbb{P}(\mathcal{E}) \geq \mathbb{P}(A(\text{IND}(\mathcal{G}_1)) \neq A(\text{IND}(\mathcal{G}_2))) - 1/3 \geq 1/3.$$

We now bound $\mathbb{P}(\mathcal{E})$ above. By [Lemma 47](#), we have

$$\mathbb{P}(\mathcal{E}) \leq 4^k k^k p_2 \max_{\vec{b} \in X_{3Z}} \sum_{i \in \mathcal{I}(\vec{b})} \min \left\{ |S_i(\vec{b})|^r, \frac{1}{p_1 |S_i(\vec{b})|^{k-r}} \right\}.$$

Combining these two equations yields

$$4^k k^k p_2 \max_{\vec{b} \in X_{3Z}} \sum_{i \in \mathcal{I}(\vec{b})} \min \left\{ |S_i(\vec{b})|^r, \frac{1}{p_1 |S_i(\vec{b})|^{k-r}} \right\} \geq \frac{1}{3}. \quad (3.3.18)$$

The remainder of the proof consists of bounding the left side of (3.3.18) above in terms of Z . We now write $y_i = |S_i(\vec{b})|$ for all $i \in [t]$, and we define the set Y with

$$Y = \left\{ \vec{y} \in [0, n]^t : \sum_{i=1}^t y_i^{\alpha_k} \leq 3Z \right\},$$

where $[0, n]$ is the real interval. For all $\vec{b} \in X_{3Z}$, the vector $(|S_1(\vec{b})|, \dots, |S_t(\vec{b})|)$ is contained in Y , since $\text{cost}(y_i) = y_i^{\alpha_k}$ holds. Moreover, $|S_i(\vec{b})| \geq 1$ for all $i \in \mathcal{I}(\vec{b})$ by definition. By (3.3.18), we therefore have

$$\max_{\vec{y} \in Y} \sum_{\substack{i \in [t] \\ y_i \geq 1}} \min \left\{ y_i^r, \frac{1}{p_1 y_i^{k-r}} \right\} \geq \frac{1}{3p_2 4^k k^k} \geq \frac{1}{3p_2 k^{k+2}}.$$

Since $x \mapsto x^r$ is increasing and $x \mapsto 1/(p_1 x^{k-r})$ is decreasing, their minimum is maximised over the non-negative reals when they are equal. Over the interval $[0, n]$, the minimum is maximised for some $x \in [0, n]$ with $x^r \leq 1/(p_1 x^{k-r})$, which is equivalent to $0 \leq x \leq \min\{n, p_1^{-1/k}\}$. It follows that replacing the interval $[0, n]$ in the definition of Y by the interval $[0, p_1^{-1/k}]$ will not affect the value of the maximum; thus writing

$$Y' = \left\{ \vec{y} \in [0, p_1^{-1/k}]^t : \sum_{i=1}^t y_i^{\alpha_k} \leq 3Z \right\},$$

it follows that

$$\max_{\vec{y} \in Y'} \sum_{i=1}^t y_i^r \geq \max_{\vec{y} \in Y'} \sum_{\substack{i \in [t] \\ y_i \geq 1}} \min \left\{ y_i^r, \frac{1}{p_1 y_i^{k-r}} \right\} = \max_{\vec{y} \in Y} \sum_{\substack{i \in [t] \\ y_i \geq 1}} \min \left\{ y_i^r, \frac{1}{p_1 y_i^{k-r}} \right\} \geq \frac{1}{3p_2 k^{k+2}}.$$

We now apply Karamata's inequality in the form of [Corollary 22](#), taking $c = 1/p_1^{1/k}$ and $W = 3Z$. Since $r \geq \alpha_k$, this yields

$$\frac{3Z}{p_1^{(r-\alpha_k)/k}} \geq \frac{1}{3p_2 k^{2k}},$$

Substituting in the definitions of p_1 and p_2 then yields

$$Z \geq \frac{n^{r-(r/k)(r-\alpha_k)}}{1080 k^{3k} \varepsilon^{(r-\alpha_k)/k}} = \frac{n^{(kr+\alpha_k r-r^2)/k}}{1080 k^{3k} \varepsilon^{(r-\alpha_k)/k}}$$

as required. ◀

4 Colourful independence oracle with cost

In this section, we study the edge estimation problem for cIND -oracle algorithms, which are given access to the colourful independence oracle $\text{cIND}(G)$ of a k -uniform hypergraph G . Any query $\text{cIND}(G)_{X_1, \dots, X_k}$ incurs a cost of $\text{cost}_k(|X_1| + \dots + |X_k|)$, where $\text{cost} = \{\text{cost}_k : k \geq 2\}$ is regularly-varying with parameter k . In [Section 4.1](#), we prove the upper bound part of [Theorem 2](#) as a special case of [Theorem 48](#) by constructing a randomised cIND -oracle algorithm for edge estimation. In [Section 4.3](#), we prove the lower bound part of [Theorem 2](#) as a special case of [Theorem 68](#) by showing that no randomised cIND -oracle algorithm for edge estimation can have significantly smaller worst-case oracle cost than the one obtained in [Section 4.1](#).

4.1 Oracle algorithm for edge estimation

We will prove the following result.

Algorithm	Statement	Approximation	Condition
Count_DLM22	Theorem 57	ε	—
Count	Theorem 48	ε	—
ColourCoarse_DLM22	Lemma 51	$(4k \log n)^k$	large k
ColourCoarse_New	Lemma 50	$k^{\mathcal{O}(k)} \log^{k-\alpha'_k-1} n$	small k
CoarseSmallCoreHelper	Lemma 60	$k^{\mathcal{O}(k)}$	small cores
CoarseSmallCore	Lemma 54	boost of CoarseSmallCoreHelper	
CoarseLargeCoreHelper	Lemma 63	$(k^{\mathcal{O}(k)} \log^{k- I } n)$	large cores
CoarseLargeCore	Lemma 55	boost of CoarseLargeCoreHelper	
VerifyGuess_New	Lemma 62	distinguishes “ $e(G) \ll M$ ” from “core and $e(G) \geq M$ ”	

■ **Table 1** Overview of the algorithms used in [Section 4.1](#).

► **Theorem 48.** *Let $\text{cost} = \{\text{cost}_k : k \geq 2\}$ be a regularly-varying parameterised cost function with parameter k and index $\alpha_k \in [0, k]$, let $\alpha'_k := \lceil \alpha_k \rceil - 1$, and let $T := \log(1/\delta) \varepsilon^{-2} k^{27k} \log^{4(k-\alpha'_k)+14} n$. There is a randomised cIND-oracle algorithm $\text{Count}(\text{cIND}(G), \varepsilon, \delta)$ with worst-case running time $\mathcal{O}(T \cdot (\text{cost}_k(n) + n))$, worst-case oracle cost $\mathcal{O}(T \cdot \text{cost}_k(n))$, and the following behaviour: Given an n -vertex k -hypergraph G and rationals $\varepsilon, \delta \in (0, 1)$, the algorithm outputs an integer m that, with probability at least $1 - \delta$, is an ε -approximation to $e(G)$.*

Recall from [Section 2.1.2](#) that we strictly separate our running times from our oracle costs, and note that the running time in [Theorem 48](#) still depends on the cost function. Indeed, Count will exploit a tradeoff between oracle cost and running time, and this is why we require the index of the cost function to be efficiently computable in [Definition 11\(v\)](#).

The structure of the proof of [Theorem 48](#) is roughly similar to the proof of [[15](#), Theorem 1.1]: We construct a cIND-oracle algorithm ColourCoarse with a coarse multiplicative approximation guarantee and use the following theorem, proved implicitly in [[15](#)], to boost it into an ε -approximation algorithm. See also [[6](#), Lemma 5.2], where this reduction was formally stated for the worst-case number of queries; in our formalisation, the oracle cost can be general, and we carefully distinguish the running time and the oracle cost.

► **Theorem 49** (Turn coarse into fine approximation). *Let $b = b(n, k)$, $T = T(n, k) = \Omega(k^2 n)$ and $C = C(n, k)$, and suppose that $\text{ColourCoarse}(\text{cIND}(G), X_1, \dots, X_k)$ is a randomised cIND-oracle algorithm with worst-case running time T , worst-case oracle cost C , and the following behaviour: Given an n -vertex k -hypergraph G with n a power of two and disjoint sets $X_1, \dots, X_k \subseteq V(G)$, the algorithm outputs an integer m such that, with probability at least $2/3$, we have $m/b \leq e(G[X_1, \dots, X_k]) \leq mb$.*

Then there is a randomised cIND-oracle algorithm $\text{Count}(\text{cIND}(G), \varepsilon, \delta)$ with worst-case running time

$$\mathcal{O}(\log(1/\delta) \varepsilon^{-2} e^{3k} b^2 \log^4 n \cdot T),$$

worst-case oracle cost

$$\mathcal{O}(\log(1/\delta) \varepsilon^{-2} e^{3k} b^2 \log^4 n \cdot C),$$

and the following behaviour: Given an n -vertex k -hypergraph G and rationals $\varepsilon, \delta \in (0, 1)$, the algorithm outputs an integer m that, with probability at least $1 - \delta$, is an ε -approximation to $e(G)$.

Proof. We first reduce from approximate edge counting in the whole graph with multiplicative error $2b$ to approximate edge counting in induced k -partite subgraphs with multiplicative

error b ; that is, we give a randomised cIND -oracle algorithm $\text{Coarse}(\text{cIND}(G), \delta)$ with the following behaviour. Suppose G is an n -vertex k -hypergraph to which Coarse has (only) colourful oracle access, where n is a power of two, and suppose $0 < \delta < 1$. Then, in time $\mathcal{O}(\log(1/\delta)ke^{2k} \cdot T)$ and with oracle cost $\mathcal{O}(\log(1/\delta)ke^{2k} \cdot C)$, $\text{Coarse}(\text{cIND}(G), \delta)$ outputs a rational number \hat{e} . Moreover, with probability at least $1 - \delta$, $\hat{e}/2b \leq e(G) \leq \hat{e} \cdot 2b$.

To obtain this algorithm Coarse from ColourCoarse , we carry out an intermediate colour-coding procedure ($\text{HelperCoarse}(\text{cIND}(G))$ in [15]), whose running time is dominated by $\mathcal{O}(ke^{2k})$ invocations of ColourCoarse , then run HelperCoarse a total of $\mathcal{O}(\log(1/\delta))$ times and output the median result. The proofs of correctness of these steps are given as [15, Lemma 4.3] and [15, Lemma 3.3], respectively.

The main counting algorithm in [15] is denoted there by Count ; to avoid overloading notation, we will instead refer to it as Count_DLM22 . Having defined Coarse , we now define Count by running $\text{Count_DLM22}(\text{cIND}(G), \varepsilon, \delta)$, replacing the Coarse subroutine from [15] with this new version and replacing the value of b in [15] (which corresponds to the error bound in Coarse) with $b^* := 2b(n, k)$. By exactly the same argument as in [15], the output of Count has the desired properties with the desired probability; it remains to bound the running time and query cost. Thankfully, most of these bounds are already given in terms of b , and so we only need to follow the analysis through.

From the proof of [15, Theorem 1.1], Count_DLM22 is just a wrapper for the main algorithm $\text{HelperCount}(\text{cIND}(G), \varepsilon)$, which serves to remove the requirement that n be a power of two (by adding isolated vertices) and reduce the failure probability to δ by running HelperCount $\mathcal{O}(1/\delta)$ times and outputting the median value.

The running time of HelperCount is analysed in [15, Lemma 3.5]; from this analysis, both the running time and oracle cost are dominated by $\mathcal{O}(\log n)$ calls to a subroutine Refine .

In [15, Lemma 3.4], the running time of Refine is given in terms of some of its arguments L , ξ and δ' . From steps (A2) and (A5) of HelperCount in [15], in every invocation of Refine we have $\xi = \mathcal{O}(\varepsilon^{-1} \log n)$ and $1/\delta' = \mathcal{O}(\log n)$. The value of L is not fixed — it is a so-called (G, b, y) -list L of induced subgraphs whose weighted sum approximates $e(G)$ — but from invariant (iii) in the proof of [15, Lemma 3.5], in each invocation of Refine we have

$$|L| = \mathcal{O}(k \log(nb^*) + (b^*)^2 \xi^{-2} \log(1/\delta')) = \mathcal{O}(kb^2 \varepsilon^{-2} \log^3 n).$$

With these bounds in place, we now bound the running time and oracle cost of Refine . From [15, Lemma 3.4], the running time and oracle cost of Refine are dominated by calls to Coarse ; the total number of such calls is upper bounded in [15, Lemma 3.5] by $\mathcal{O}(\lambda)$, where

$$\lambda := |L| + 2^k \xi^{-2} (b^*)^2 \log(1/\delta') = \mathcal{O}(\varepsilon^{-2} 2^k b^2 \log^3 n).$$

Each call to Coarse has $\delta^{-1} = \mathcal{O}(\lambda/\delta')$, so the running time of each call is $\mathcal{O}(\log(\lambda \log n) \cdot ke^{2k}T)$ and the query cost of each call is $\mathcal{O}(\log(\lambda \log n)ke^{2k}C)$. We may assume that $\varepsilon^{-1} < n^k$ (otherwise the algorithm counts exactly by brute force) and similarly that $b < n^k$, so $\log(\lambda \log n) = \mathcal{O}(k \log n)$. It follows that the running time of each call to Refine is $\mathcal{O}(k^2 e^{2k} \log n \cdot T)$ and the oracle cost is $\mathcal{O}(k^2 e^{2k} \log n \cdot C)$. Multiplying by the number λ of total calls, we see that the running time of HelperCount is $\mathcal{O}(\varepsilon^{-2} e^{3k} b^2 \log^4 n \cdot T)$, and the oracle cost is $\mathcal{O}(\varepsilon^{-2} e^{3k} b^2 \log^4 n \cdot C)$. Since UncolApprox runs HelperCount $\mathcal{O}(\log(1/\delta))$ times, the result follows. \blacktriangleleft

In order to apply Theorem 49 to prove Theorem 48, we need to provide an algorithm ColourCoarse with suitable properties. In fact, we will use one of two different algorithms,

depending on the size of k relative to n . Most of the rest of the section is devoted to proving the following lemma, which gives the existence of a suitable algorithm when k is not too large.

► **Lemma 50.** *Let $\text{cost} = \{\text{cost}_k : k \geq 2\}$ be a regularly-varying parameterised cost function with parameter k and index α_k . Let*

$$\begin{aligned}\alpha'_k &:= \lceil \alpha_k \rceil - 1, & T &:= k^{9k+2} \log^{2(k-\alpha'_k)+9} n, \\ b &:= (2k)^{5k} (\log n)^{k-\alpha'_k-1} \log^k (8k \log^{40k(k-\alpha'_k)/(\alpha_k-\alpha'_k)}(n)).\end{aligned}$$

There is a randomised cIND-oracle algorithm `ColourCoarse_New`(cIND(G), X_1, \dots, X_k) with worst-case running time $\mathcal{O}(T(\text{cost}_k(n) + n))$, worst-case oracle cost $\mathcal{O}(T \text{cost}_k(n))$, and the following behaviour: Given an n -vertex k -hypergraph G with n a power of two, $k \leq (\log n)/(\log \log n)^2$, and disjoint sets $X_1, \dots, X_k \subseteq V(G)$, the algorithm outputs an integer m such that, with probability at least $2/3$, we have $m/b \leq e(G[X_1, \dots, X_k]) \leq mb$.

We defer the proof of this lemma to [Sections 4.1.1–4.1.3](#). When k is large enough relative to n that $\log^k n = k^{\mathcal{O}(k)}$, the algorithm of [Lemma 50](#) may break down, but in this regime we can simply use the coarse approximation algorithm from [\[15\]](#). We now state its properties.

► **Lemma 51** (Dell, Lapinskas, and Meeks [\[15, Lemma 4.2\]](#)). *Let $\text{cost} = \{\text{cost}_k : k \geq 2\}$ be a regularly-varying parameterised cost function with parameter k and index α_k , let $b := (4k \log n)^k$, and let $T := (8k \log n)^{2k+2}$. There is a randomised cIND-oracle algorithm*

$$\text{ColourCoarse_DLM22}(\text{cIND}(G), X_1, \dots, X_k)$$

with worst-case running time $\mathcal{O}(T(\text{cost}_k(n) + n))$, worst-case oracle cost $\mathcal{O}(T \text{cost}_k(n))$, and the following behaviour: Given an n -vertex k -hypergraph G with n a power of two and disjoint sets $X_1, \dots, X_k \subseteq V(G)$, the algorithm outputs an integer m such that, with probability at least $2/3$, we have $m/b \leq e(G[X_1, \dots, X_k]) \leq mb$.

Proof. This is immediate from [\[15, Lemma 4.2\]](#), bounding the cost of each query above by $\mathcal{O}(\text{cost}_k(n))$. This bound is valid by [Lemma 12](#) since the cost function is regularly-varying. ◀

We now set out the proof of [Theorem 48](#) from [Theorem 49](#) and [Lemmas 50](#) and [51](#) (which consists of easy algebra), before devoting the rest of the section to proving [Lemma 50](#).

► **Theorem 48.** *Let $\text{cost} = \{\text{cost}_k : k \geq 2\}$ be a regularly-varying parameterised cost function with parameter k and index $\alpha_k \in [0, k]$, let $\alpha'_k := \lceil \alpha_k \rceil - 1$, and let $T := \log(1/\delta) \varepsilon^{-2} k^{27k} \log^{4(k-\alpha'_k)+14} n$. There is a randomised cIND-oracle algorithm `Count`(cIND(G), ε, δ) with worst-case running time $\mathcal{O}(T \cdot (\text{cost}_k(n) + n))$, worst-case oracle cost $\mathcal{O}(T \cdot \text{cost}_k(n))$, and the following behaviour: Given an n -vertex k -hypergraph G and rationals $\varepsilon, \delta \in (0, 1)$, the algorithm outputs an integer m that, with probability at least $1 - \delta$, is an ε -approximation to $e(G)$.*

Proof. We apply [Theorem 49](#), taking our coarse approximate counting algorithm `ColourCoarse` to be as follows. Given an n -vertex instance (G, X_1, \dots, X_k) , if $k \leq (\log n)/(\log \log n)^2$ then we apply the algorithm `ColourCoarse_New` of [Lemma 50](#) and return the results, and otherwise we apply the algorithm `ColourCoarse_DLM22` of [Lemma 51](#) and return the results.

As in [Lemma 50](#), let

$$\begin{aligned}\alpha'_k &:= \lceil \alpha_k \rceil - 1, & T_1 &:= k^{9k+2} \log^{2(k-\alpha'_k)+9} n, \\ b_1 &:= (2k)^{5k} (\log n)^{k-\alpha'_k-1} \log^k (8k \log^{40k(k-\alpha'_k)/(\alpha_k-\alpha'_k)}(n)).\end{aligned}$$

As in [Lemma 51](#), let

$$T_2 := (8k \log n)^{2k+2}, \quad b_2 := (4k \log n)^k.$$

Observe that when $k \geq (\log n)/(\log \log n)^2$, we have

$$k^{2k} = e^{2k \log k} = e^{2k(1-o(1)) \log \log n} = \Omega(\log^k n),$$

and hence

$$T_2 = \mathcal{O}(k^{7k} \log^2 n) = \mathcal{O}(T_1), \quad b_2 = \mathcal{O}(k^{4k}) = \mathcal{O}(b_1).$$

Thus on an n -vertex instance, our coarse approximation algorithm uses oracle queries of combined cost $\mathcal{O}(T_1 \text{cost}_k(n))$, runs in time $\mathcal{O}(T(\text{cost}_k(n) + n))$, and has multiplicative error $\mathcal{O}(b_1)$.

We now take **Count** to be the algorithm of [Theorem 49](#). Let

$$T_3 = \log(1/\delta) \varepsilon^{-2} e^{3k} b_1^2 \log^4 n;$$

then **Count** uses oracle queries of combined cost $\mathcal{O}(T_1 T_3 \text{cost}_k(n))$ and runs in time $\mathcal{O}(T_1 T_3 n)$. Observe that

$$b_1^2 = \mathcal{O}(k^{13k} (\log n)^{2(k-\alpha'_k)} (\log \log n)^{2k}). \quad (4.1.1)$$

If $k \geq (\log \log n)/(\log \log \log n)^2$, then

$$k^{2k} = e^{2k \log k} = e^{2k(1-o(1)) \log \log \log n} = \Omega((\log \log n)^k);$$

if instead $k \leq (\log \log n)/(\log \log \log n)^2$, then

$$(\log \log n)^k \leq e^{(\log \log n)/(\log \log \log n)} = (\log n)^{o(1)}.$$

In either case, we have $(\log \log n)^{2k} = \mathcal{O}(k^{4k} \log n)$, and hence by (4.1.1), $b_1^2 = \mathcal{O}(k^{17k} \log^{2(k-\alpha'_k)+1} n)$. Thus

$$T_1 T_3 = \mathcal{O}(\log(1/\delta) \varepsilon^{-2} k^{27k} \log^{4(k-\alpha'_k)+14} n) = \mathcal{O}(T),$$

and the result follows. ◀

The rest of this section is dedicated to proving [Lemma 50](#).

4.1.1 Coarse approximation: an overview

In order to prove [Lemma 50](#), given oracle access to an n -vertex k -hypergraph G and vertex classes X_1, \dots, X_k , we will make use of two separate approximate counting algorithms. Writing $H := G[X_1, \dots, X_k]$ for brevity, each algorithm will provide an estimate that is very likely not to be much larger than $e(H)$. Depending on G , one of these estimates is very likely to also not be much smaller than $e(H)$, so by returning whichever result is largest we obtain a coarse approximation to $e(H)$ as required. In this section, we set out the properties of these algorithms and explain how to use them to prove [Lemma 50](#). We then set out the algorithms themselves and prove their stated properties in [Sections 4.1.2 and 4.1.3](#).

For motivation, we first sketch a possible proof of [Lemma 50](#) that does not work. Form a graph H' from H by deleting vertices independently at random, so that each vertex is retained with probability $p = 1/(\log n)^{k^2}$. Each edge of H' is preserved with probability

$1/(\log n)^{k^3}$, so in expectation, H' contains $e(H)/(\log n)^{k^3}$ edges and $n/(\log n)^{k^2}$ vertices. By Chernoff bounds, we in fact have $|V(H')| = (1 + o(1))n/(\log n)^{k^2}$ with high probability; suppose (incorrectly) that we also had $e(H') = (1 + o(1))e(H)/(\log n)^{k^3}$ with high probability. Then we could simply run the full approximate counting algorithm of [15] on H' and multiply the result by $(\log n)^{k^3}$. Indeed, the factor of $(\log |V(H')|)^{4k+7}$ in the query count of [15] would be dominated by the factor of $1/(\log n)^{\alpha k^2}$ saved in oracle cost by running it on an instance with fewer vertices.

Of course, we do not in general have such concentration of $e(H')$, as illustrated by the following definition.

► **Definition 52.** *Let H be a k -hypergraph, and let $0 < \zeta \leq 1$. We say that $v \in V(H)$ is a ζ -root of H if $d(v) \geq \zeta e(H)$.*

If X_1 contains a ζ -root v with ζ relatively large, then we should expect $e(H')$ to depend very strongly on whether or not $v \in V(H')$, and so we should not expect concentration of $e(H')$. It turns out, however, that in some sense this is the only barrier to concentration of $e(H')$: if H were to contain no ζ -roots for an appropriate choice of ζ , then the above proof would work.

Despite the possibility of ζ -roots, we can still recover part of this argument if we know enough about where ζ -roots occur in the graph: if some vertex classes do not contain any ζ -roots, then we can reasonably hope for concentration of the number of edges that survive when we delete vertices uniformly at random from these root-free classes. (See Lemma 59.) In fact, we will make use of subgraphs of H which contain a large proportion of the edges and in which all roots are contained in a collection of relatively small vertex classes; this notion is formalised in the following definition.

► **Definition 53.** *Let $H = (V, E)$ be a k -partite k -hypergraph with vertex classes X_1, \dots, X_k . Let $I \subseteq [k]$, and let $\zeta \in (0, 1)$. For each $i \in [k]$, let $Y_i \subseteq X_i$. Then we say that (Y_1, \dots, Y_k) is an (I, ζ) -core of H if the following properties hold:*

- (i) $e(H[Y_1, \dots, Y_k]) \geq e(H)/(2k)^k$;
- (ii) for all $i \in I$, we have $|Y_i| \leq 2/\zeta$;
- (iii) for all $i \in [k] \setminus I$, the set Y_i contains no ζ -roots of $H[Y_1, \dots, Y_k]$.

We will show in Lemma 56 that, for any $\zeta \in (0, 1)$, every k -partite k -hypergraph contains an (I, ζ) -core for some I . Before doing so, we set out the behaviour of two algorithms which exploit this fact.

For simplicity, suppose (Y_1, \dots, Y_k) is an (I, ζ) -core of the k -partite k -hypergraph H for some suitably-chosen ζ , and write $H' = H[Y_1, \dots, Y_k]$; we use this specific core to sketch proofs of correctness for the algorithms, but the algorithms themselves do not have access to the sets (Y_1, \dots, Y_k) . Our first algorithm is designed to run efficiently in the case where G has an (I, ζ) -core with $|I|$ small, and works by recursively applying the algorithm of [15] to a collection of smaller instances, roughly as described above: vertices from classes outside I are deleted randomly and classes in I are partitioned into appropriately-sized subsets, each combination of which is considered in one of the smaller instances. Formally, we will prove the following lemma in Section 4.1.2. (The parameter t in this algorithm is a technical convenience which we will optimise later.)

► **Lemma 54.** *Let $\text{cost} = \{\text{cost}_k : k \geq 2\}$ be a regularly-varying parameterised cost function with parameter k . Let*

$$b := (2k)^{k+2}, \quad T := \log(1/\delta) k^{6k+1} \log^{4k+8}(n) n^{|I|} / t^{|I|}.$$

There is a randomised cIND -oracle algorithm

$\text{CoarseSmallCore}(\text{cIND}(G), X_1, \dots, X_k, I, t, \delta)$

with worst-case running time $\mathcal{O}(n + T \cdot kt \log(n))$, worst-case oracle cost $\mathcal{O}(T \text{cost}_k(2kt))$, and the following behaviour: Given an n -vertex k -hypergraph G with n a power of two, disjoint sets $X_1, \dots, X_k \subseteq V(G)$, a set $I \subseteq [k]$, an integer t with $n \geq t \geq 12 \log k$, and a rational $\delta > 0$, the algorithm outputs a non-negative integer m such that, with probability at least $1 - \delta$:

- (i) $m \leq b \cdot e(G[X_1, \dots, X_k])$;
- (ii) if G has an (I, ζ) -core for some ζ satisfying $t \geq (8k\zeta)^{1/(2(k-|I|))}n$, then $m \geq e(G[X_1, \dots, X_k])/b$.

We note explicitly that [15] is concerned only with the total number of queries (that is, $\text{cost}_k(n) = 1$), and that in this setting Lemma 54 does not yield an improved oracle cost. Note also that CoarseSmallCore does not require the value of sets (Y_1, \dots, Y_k) of the (I, ζ) -core, only the value of I itself (which we will simply guess, as there are only 2^k possibilities).

Our second algorithm is designed to deal with the opposite situation, when there is an (I, ζ) -core with $|I|$ large. In this case, we can adapt the algorithm of [15] to perform significantly better. Very roughly speaking, the algorithm of [15] works by finding probabilities $p_1, \dots, p_k \in \{1, 1/2, 1/4, \dots, 1/\log n\}$ such that:

- (a) $p_1 p_2 \dots p_k$ is as small as possible;
- (b) on randomly deleting all but roughly p_i proportion of vertices from each vertex class X_i , the resulting graph still contains at least one edge with high probability.

We then conclude that H contains at least roughly $1/(p_1 \dots p_k)$ edges. (Note that this is inaccurate on a formal level, but it is close enough to give intuition.) The approach to finding such probabilities p_1, \dots, p_k in [15] is simply to check all $\log^k n$ possibilities. However, if H contains an (I, ζ) -core, then for all $i \in I$ we know that many edges are concentrated on a few ζ -roots in X_i . It follows that we must take p_i very close to 1 to satisfy property (b), substantially reducing the number of possible values of p_1, \dots, p_k we need to check; this translates into a reduction in both the running time and oracle cost that scales with $|I|$. Formally, we will prove the following lemma in Section 4.1.3.

► **Lemma 55.** Let $\text{cost} = \{\text{cost}_k : k \geq 2\}$ be a regularly-varying parameterised cost function with parameter k . Let

$$b := (2k)^{5k} \log^{k-|I|} n \log^{|I|}(1/\zeta), \quad T := 2^{7k} \log(1/\delta) \log^{2|I|}(1/\zeta) (\log n)^{2(k-|I|)+1}.$$

There is a randomised cIND -oracle algorithm

$\text{CoarseLargeCore}(\text{cIND}(G), X_1, \dots, X_k, I, \zeta, \delta)$

with worst-case running time $\mathcal{O}(Tn \log n)$, worst-case oracle cost $\mathcal{O}(T \text{cost}_k(n))$, and the following behaviour: Given an n -vertex k -hypergraph G with $n \geq 32$ a power of two, disjoint sets $X_1, \dots, X_k \subseteq V(G)$, a set $I \subseteq [k]$, rationals $\zeta, \delta \in (0, \frac{1}{32})$ such that the denominator of ζ is $\mathcal{O}(n^k)$, the algorithm outputs a non-negative integer m such that, with probability at least $1 - \delta$:

- (i) $m \leq b \cdot e(G[X_1, \dots, X_k])$; and
- (ii) if G has an (I, ζ) -core, then $m \geq e(G[X_1, \dots, X_k])/b$.

We now show that every k -hypergraph contains an (I, ζ) -core for some I , ensuring that one of these two algorithms is always guaranteed to perform well for a suitably-chosen value of ζ .

► **Lemma 56.** *Let $0 < \zeta \leq 1$, and let H be a k -partite k -hypergraph with vertex classes X_1, \dots, X_k . Then there exists $I \subseteq [k]$ such that H contains an (I, ζ) -core.*

Proof. Informally, we will prove that we can find a set I and an (I, ζ) -core (Y_1, \dots, Y_k) by applying the following algorithm. We start out with $I = \emptyset$ and $Y_i = X_i$ for all i . We look for a value of $i \notin I$ such that at least $1/(2k)$ proportion of edges of $J := H[Y_1, \dots, Y_k]$ are incident to a $(\zeta/2)$ -root of J in Y_i . If such an i exists, then we add i to I , delete all non- $(\zeta/2)$ -roots from Y_i , and repeat the process. Otherwise, for all $i \notin I$ we delete all $(\zeta/2)$ -roots from Y_i , then halt; as we will prove, (Y_1, \dots, Y_k) is then an (I, ζ) -core of H .

Formally, we proceed by induction, proving the following claim.

Claim: Suppose that there exist $J = H[Y_1, \dots, Y_k]$ and $I \subseteq [k]$ such that:

(C1) $|E(J)| \geq |E(H)|/(2k)^{|I|}$; and

(C2) for all $i \in I$, $|Y_i| \leq 2/\zeta$.

Then there exist $J' = H[Y'_1, \dots, Y'_k]$ and $I' \subseteq [k]$ such that either:

(D1) (Y'_1, \dots, Y'_k) is an (I', ζ) -core of H ; or

(D2) J' and I' satisfy (C1) and (C2) and $|I'| = |I| + 1$.

Proof of Lemma from Claim: Since (C1) and (C2) are satisfied for $J = H$ and $I = \emptyset$, we can apply the claim repeatedly until (D1) holds. This process must terminate after at most $k + 1$ applications, since we cannot have $|I| > k$.

Proof of Claim: Let $J = H[Y_1, \dots, Y_k]$ and $I \subseteq [k]$ be as in the Claim. For each $i \in [k] \setminus I$, let Z_i be the set of $(\zeta/2)$ -roots of J in X_i . We split into two cases corresponding to (D1) and (D2) of the Claim.

Case 1: For all $i \in [k] \setminus I$, we have $\sum_{v \in Z_i} d_J(v) \leq e(J)/(2k)$. In this case, we take $Y'_i = Y_i$ for all $i \in I$ and $Y'_i = Y_i \setminus Z_i$ for all $i \in [k] \setminus I$. We claim that (Y'_1, \dots, Y'_k) is an (I, ζ) -core of H (satisfying (i)–(iii) of Definition 53), as in (D1). If $I = [k]$ then this is immediate from (C1) and (C2), so suppose $|I| \leq k - 1$. Let $J' = H[Y'_1, \dots, Y'_k]$. To see that (i) holds, observe that by the definition of the sets Y'_i and by hypothesis,

$$e(J') \geq e(J) - \sum_{i \notin I} \sum_{v \in Z_i} d_J(v) \geq e(J) - (k - |I|) \frac{e(J)}{2k} \geq \frac{e(J)}{2}. \quad (4.1.2)$$

By (C1) this is at least $e(H)/(2k)^{|I|+1} \geq e(H)/(2k)^k$, so (i) holds. Moreover, (ii) follows from (C2). Finally, observe that for all $i \notin I$ and all $v \in Y'_i$, we have $v \notin Z_i$; it follows from the definition of Z_i and (4.1.2) that

$$d_{J'}(v) \leq d_J(v) < \zeta e(J)/2 \leq \zeta e(J');$$

thus v is not a ζ -root of J' , as required by (iii). We have shown that (D1) holds, so we are done.

Case 2: There exists $\ell \in [k] \setminus I$ such that $\sum_{v \in Z_\ell} d_J(v) > e(J)/(2k)$. In this case, we choose an arbitrary such ℓ , we take $Y'_\ell = Z_\ell$, $Y'_i = Y_i$ for all $i \neq \ell$, and $I' = I \cup \{\ell\}$. We claim that $J' := H[Y'_1, \dots, Y'_k]$ and I' satisfy (C1) and (C2) with $|I'| = |I| + 1$, as in (D2). We certainly have $|I'| = |I| + 1$. To see (C1), observe that by hypothesis,

$$e(J') = \sum_{v \in Z_\ell} d_J(v) > e(J)/(2k);$$

hence by (C1) of the inductive hypothesis, we have $|E(H')| \geq |E(G)|/(2k)^{|I|+1}$ as required. To see (C2), observe that, by the definition of Z_ℓ ,

$$e(J) \geq \sum_{v \in Z_\ell} d_J(v) \geq |Z_\ell| \zeta e(J)/2,$$

and hence $|Z_\ell| \leq 2/\zeta$. It follows by (C2) of the inductive hypothesis that $|Y'_i| \leq 2/\zeta$ for all $i \in I'$, as required by (C2). We have shown that (D2) holds, so we are done. \blacktriangleleft

With Lemmas 54–56 in hand, we now set out the algorithm `ColourCoarse_New` required by Lemma 50 as Algorithm 3. (Recall from the statement of Lemma 50 that $\alpha_k \in [0, k]$ is the index of our regularly-varying cost function.) We now restate Lemma 50 and prove it.

■ **Algorithm 3** `ColourCoarse_New`

This algorithm applies either `CoarseSmallCore` or `CoarseLargeCore` with each possible choice R of root classes, depending on the size of the set R , and returns the maximum number of edges estimated by any of these calls.

Oracle: Colourful independence oracle `cIND`(G) of an n -vertex k -hypergraph G .

Input: Integer n that is a power of two and satisfies $k \leq (\log n)/(\log \log n)^2$, and disjoint subsets $X_1, \dots, X_k \subseteq V(G)$.

Output: Non-negative integer m such that, with probability at least $2/3$, $m/b \leq e(G[X_1, \dots, X_k]) \leq mb$, where $\alpha'_k := \lceil \alpha_k \rceil - 1$ and

$$b := (2k)^{5k} (\log n)^{k-\alpha'_k-1} \log^k (8k \log^{40k(k-\alpha'_k)/(\alpha_k-\alpha'_k)}(n)).$$

```

1 begin
2   Set  $t := \lfloor n/(\log n)^{20k/(\alpha_k-\alpha'_k)} \rfloor$ .
3   Set  $\zeta := (t/n)^{2k(k-\alpha'_k)}/(8k)$ .
4   if  $n < 32$  or  $t < 12 \log k$  then
5     Count edges of  $G[X_1, \dots, X_k]$  by brute force, return the answer, and halt.
6   forall  $R \subseteq [k]$  do
7     if  $|R| \leq \alpha'_k$  then
8       Set  $Z_R = \text{CoarseSmallCore}(\text{cIND}(G), X_1, \dots, X_k, R, t, 1/2^{k+5})$ .
9     if  $|R| \geq \alpha'_k + 1$  then
10      Set  $Z_R = \text{CoarseLargeCore}(\text{cIND}(G), X_1, \dots, X_k, R, \zeta, 1/2^{k+5})$ .
11  return  $\max\{Z_R : |R| \subseteq [k]\}$ .
```

► **Lemma 50.** Let $\text{cost} = \{\text{cost}_k : k \geq 2\}$ be a regularly-varying parameterised cost function with parameter k and index α_k . Let

$$\begin{aligned} \alpha'_k &:= \lceil \alpha_k \rceil - 1, & T &:= k^{9k+2} \log^{2(k-\alpha'_k)+9} n, \\ b &:= (2k)^{5k} (\log n)^{k-\alpha'_k-1} \log^k (8k \log^{40k(k-\alpha'_k)/(\alpha_k-\alpha'_k)}(n)). \end{aligned}$$

There is a randomised `cIND`-oracle algorithm `ColourCoarse_New`(`cIND`(G), X_1, \dots, X_k) with worst-case running time $\mathcal{O}(T(\text{cost}_k(n) + n))$, worst-case oracle cost $\mathcal{O}(T \text{cost}_k(n))$, and the following behaviour: Given an n -vertex k -hypergraph G with n a power of two, $k \leq (\log n)/(\log \log n)^2$, and disjoint sets $X_1, \dots, X_k \subseteq V(G)$, the algorithm outputs an integer m such that, with probability at least $2/3$, we have $m/b \leq e(G[X_1, \dots, X_k]) \leq mb$.

Proof. We first prove that `CoarseLargeCore` has the correct behaviour, before we turn to time and cost.

Behaviour.

We begin by noting that the conditions of [Lemma 54](#) (respectively [Lemma 55](#)) are satisfied each time we call the relevant subroutine. For `CoarseLargeCore`, it is immediate that when reaching line 10 we have $n \geq 32$. We trivially have $1/2^{k+5} < 1/32$, and since $\zeta = (t/n)^{2k(k-\alpha'_k)}/(8k) \leq 1/\log^{20} n$ and $n > 4$, we also have $\zeta < 1/32$. For `CoarseSmallCore`, we trivially have $n \geq t$ and that t is an integer, and Lines 4-5 guarantee that $t \geq 12 \log k$. Finally, we have $\alpha'_k \leq k-1$, so `CoarseSmallCore` is always called with $R \subset [k]$.

Observe that the number of calls made to `CoarseSmallCore` and `CoarseLargeCore` combined is 2^k . [Lemmas 54](#) and [55](#) imply that the results of each such call lie within certain bounds with failure probability at most $1/2^{k+5}$; by a union bound, it follows that with probability at least $31/32$, no call to `CoarseSmallCore` or `CoarseLargeCore` fails in this way. In this case, we say that `ColourCoarse_New` *succeeds*. It therefore suffices to prove that, whenever `ColourCoarse_New` succeeds, its output m satisfies $m/b \leq e(G[X_1, \dots, X_k]) \leq mb$.

First, note if `ColourCoarse_New` succeeds, then by [Lemma 54\(i\)](#) and [Lemma 55\(i\)](#), every invocation of either `CoarseSmallCore` or `CoarseLargeCore` returns a value that is at most $b \cdot e(G[X_1, \dots, X_k])$, so we have $m \leq b \cdot e(G[X_1, \dots, X_k])$ as required.

To see that `ColourCoarse_New` returns a value that is not too small, recall from [Lemma 56](#) that there exists $I \subseteq [k]$ such that $G[X_1, \dots, X_k]$ contains an (I, ζ) -core. Suppose first that $|I| \leq \alpha'_k$. In this case we obtain Z_I by invoking `CoarseSmallCore` with $R = I$. Observe that, by definition,

$$(8k\zeta)^{1/(2(k-|I|))}n = (t/n)^{2k(k-\alpha'_k)/(2(k-|I|))}n \leq (t/n)^k n \leq t,$$

so it follows from [Lemma 54\(ii\)](#) that $Z_I \geq e(G[X_1, \dots, X_k])/b$. Since our output m is the maximum over all values Z_R , it follows that $m \geq e(G[X_1, \dots, X_k])/b$, as required. Now suppose that $|I| \geq \alpha'_k + 1$. In this case, we obtain Z_I by invoking `CoarseLargeCore` with $R = I$, and it follows from [Lemma 55\(ii\)](#) that $Z_R \geq e(G[X_1, \dots, X_k])/b$. As before, it is then immediate that $m \geq e(G[X_1, \dots, X_k])/b$, as required.

Running time and oracle cost.

We now prove the claimed bounds on the time and cost of `ColourCoarse_New`. Since $k \leq (\log n)/(\log \log n)^2$, we have $\log^k(n) \leq 2^{(\log n)/(\log \log n)} = n^{o(1)}$ as $n \rightarrow \infty$, and so

$$t = n^{1-o(1)}. \tag{4.1.3}$$

Let $\eta = (\alpha_k - \alpha'_k)/2$. Since cost_k is regularly-varying with index α_k and a slowly-varying component which does not depend on k , by [Lemma 8\(ii\)](#) there exists x_0 such that

$$\text{for all } k \geq 2, \text{ all } x \geq x_0 \text{ and all } A_x \geq 1, \text{cost}_k(A_x x)/\text{cost}_k(x) \geq A_x^{\alpha_k - \eta}. \tag{4.1.4}$$

Throughout the proof, we assume without loss of generality that n is sufficiently large that $t \geq x_0$.

Suppose we solve the problem by brute force in line 5, with time and oracle cost $\mathcal{O}(n^k)$. Since $t \leq 12 \log k \leq \log n$, and since $t = n^{1-o(1)}$ by (4.1.3), we have $\log n \leq n^{1-o(1)}$ and hence $n = \mathcal{O}(1)$. It follows that $n^k = 2^{\mathcal{O}(k)} = \mathcal{O}(k^k)$. For the rest of the proof, suppose we do not solve the problem by brute force in line 5.

First, we bound the oracle cost of invocations of `CoarseSmallCore`. Let

$$T_1 = \max \{ \log(2^{k+5}) k^{6k+1} \log^{4k+8}(n) (n/t)^i : 0 \leq i \leq \alpha'_k \}; \quad (4.1.5)$$

then by [Lemma 54](#), each invocation of `CoarseSmallCore` has oracle cost $\mathcal{O}(T_1 \text{cost}_k(2kt))$. Observe that the maximum in (4.1.5) is attained at $i = \alpha'_k$, so

$$T_1 \leq k^{6k+2} \log^{4k+8}(n) (n/t)^{\alpha'_k} \leq (T/k^{3k}) \log^{4k-1}(n) (n/t)^{\alpha'_k} \quad (4.1.6)$$

Since `CoarseSmallCore` is invoked at most 2^k times, by (4.1.6) and (4.1.4) applied with $x = 2kt$ and $A_x = n/(2kt)$, the total oracle cost of all such invocations is at most

$$\begin{aligned} \mathcal{O}(T_1 \text{cost}_k(2kt)) &= \mathcal{O}\left(2^k \frac{T}{k^{3k}} \log^{4k}(n) \left(\frac{n}{t}\right)^{\alpha'_k} \left(\frac{2kt}{n}\right)^{\alpha_k - \eta} \text{cost}_k(n)\right) \\ &= \mathcal{O}\left(T \log^{4k}(n) \left(\frac{t}{n}\right)^{\alpha_k - \alpha'_k - \eta} \text{cost}_k(n)\right). \end{aligned}$$

Substituting in the values of t and η yields an oracle cost of at most

$$\mathcal{O}(T \log^{4k}(n) \log^{-10k}(n) \text{cost}_k(n)) = \mathcal{O}(T \text{cost}_k(n)),$$

as required.

We now bound the running time of invocations of `CoarseSmallCore`. By [Lemma 54](#), each invocation of `CoarseSmallCore` has running time $\mathcal{O}(n + T_1 kt \log n)$. When $\alpha_k > 1$, we have $T_1 kt \log n = \mathcal{O}(T_1 \text{cost}_k(2kt))$, and so the running time is $\mathcal{O}(T(\text{cost}_k(n) + n))$ as above. Suppose instead $\alpha_k \leq 1$, so that $\alpha'_k = 0$. Since `CoarseSmallCore` is invoked at most 2^k times, by (4.1.6) it follows that the total running time is at most

$$\begin{aligned} \mathcal{O}\left(2^k n + (T/k^k) \log^{4k}(n) t\right) &= \mathcal{O}\left(n \left(2^k + (T/k^k) \log^{4k}(n) (t/n)\right)\right) \\ &= \mathcal{O}\left(n \left(T + T \log^{4k-20k/\alpha_k}(n)\right)\right). \end{aligned}$$

Since $\alpha_k \leq 1$, this is $\mathcal{O}(nT)$. Thus in both cases, the running time is $\mathcal{O}(T(\text{cost}_k(n) + n))$ as required.

We now bound the oracle cost of invocations of `CoarseLargeCore`. Let

$$T_2 = \max \{ 2^{7k} \log(2^{k+5}) \log^{2i}(1/\zeta) \log^{2(k-i)+1}(n) : \alpha'_k + 1 \leq i \leq k \}; \quad (4.1.7)$$

then by [Lemma 55](#), each invocation of `CoarseLargeCore` has oracle cost $\mathcal{O}(T_2 \text{cost}_k(n))$. Observe that, by the definitions of ζ and t , and since $t \geq 1$ by (4.1.3),

$$\log(1/\zeta) = \log(8k(n/t)^{2k(k-\alpha'_k)}) = \mathcal{O}(k^2 \log(n/t)) = \mathcal{O}(k^3 \log \log n). \quad (4.1.8)$$

If $k \geq (\log \log n)/(\log \log \log n)^2$, then

$$k^{3k} = 2^{3k \log k} \geq 2^{3k(1-o(1)) \log \log \log n} = \Omega((\log \log n)^k);$$

if instead $k \leq (\log \log n)/(\log \log \log n)^2$, then

$$(\log \log n)^k \leq 2^{(\log \log n)/(\log \log \log n)} = (\log n)^{o(1)}.$$

In either case, by (4.1.8) we have $\log^{2k}(1/\zeta) = \mathcal{O}(k^{6k} \log n)$. It follows from (4.1.7) that

$$T_2 = \mathcal{O}\left(2^{7k} k^{6k+1} \log^{2(k-\alpha'_k)}(n)\right) = \mathcal{O}(T/(2^k \log n)). \quad (4.1.9)$$

Thus the total oracle cost of all 2^k invocations of `CoarseLargeCore` is

$$\mathcal{O}(2^k T_2 \text{cost}_k(n)) = \mathcal{O}(T \text{cost}_k(n))$$

as required.

Similarly, by [Lemma 55](#) and [\(4.1.9\)](#), the running time of all 2^k invocations of `LargeCore` Coarse is $\mathcal{O}(2^k T_2 n \log n) = \mathcal{O}(Tn)$; thus the total time is $\mathcal{O}(T(\text{cost}_k(n) + n))$ as required.

Finally, we observe that lines [2](#), [3](#) and [11](#) take $\mathcal{O}(k2^k)$ time (including arithmetic operations on $(k \log n)$ -bit numbers) and make no oracle calls. The result therefore follows. \blacktriangleleft

4.1.2 Counting edges with a small core

As described in [Section 4.1.1](#), our second approximate counting algorithm (which will be efficient when the input graph has an (I, ζ) -core with $|I|$ small) will make use of the main counting algorithm from [\[15\]](#) as a subroutine; we can paraphrase this result as follows.

► **Theorem 57** (Dell, Lapinskas, and Meeks [\[15, Theorem 1.1 paraphrased\]](#)). *Let $\text{cost} = \{\text{cost}_k : k \geq 2\}$ be a regularly-varying parameterised cost function with parameter k . There is a randomised `cIND`-oracle algorithm `Count_DLM22`(`cIND`(G), ε , δ) with worst-case running time $\mathcal{O}(\log(1/\delta)\varepsilon^{-2}k^{6k}(\log n)^{4k+8}n)$, worst-case oracle cost $\mathcal{O}(\log(1/\delta)\varepsilon^{-2}k^{6k}(\log n)^{4k+7}\text{cost}_k(n))$, and the following behaviour: Given an n -vertex k -hypergraph G and rationals $\varepsilon, \delta \in (0, 1)$, the algorithm outputs a rational number that, with probability at least $1 - \delta$, is an ε -approximation to $e(G)$.*

We can now state the heart of our algorithm, `CoarseSmallCoreHelper`. Given correctness and time and cost bounds on `CoarseSmallCoreHelper` (which we will prove as [Lemma 60](#)), [Lemma 54](#) will follow immediately by applying [Lemma 19](#) to reduce the failure probability of `CoarseSmallCoreHelper` from $1/3$ to δ with $\mathcal{O}(\log(1/\delta))$ overhead.

Observe that line [7](#) guarantees that $|V(H_\pi)| \leq 2kt$, so the calls to `Count_DLM22` will run quickly – this is where the fast running time will come from. Moreover, the graphs H_π of line [10](#) are all edge-disjoint, and that each edge of $G[X_1, \dots, X_k]$ survives in some H_π with probability $(t/n)^{k-|I|}$; thus we have $\mathbb{E}(Z) = e(G[X_1, \dots, X_k])$. Establishing concentration here will be central to the correctness proof. To this end, we first recall the following standard martingale concentration bound due to McDiarmid [\[26\]](#); we then apply it to prove concentration for specific graphs H_π in [Lemma 59](#).

► **Lemma 58.** *Let f be a real function of independent real random variables Z_1, \dots, Z_m , and let $\mu = \mathbb{E}(f(Z_1, \dots, Z_m))$. Let $c_1, \dots, c_m \geq 0$ be such that, for all $i \in [m]$ and all pairs $(\mathbf{x}, \mathbf{x}')$ differing only in the i 'th coordinate, we have $|f(\mathbf{x}) - f(\mathbf{x}')| \leq c_i$. Then for all $y > 0$,*

$$\mathbb{P}(|f(Z_1, \dots, Z_m) - \mu| \geq y) \leq 2e^{-2y^2 / \sum_{i=1}^m c_i^2}.$$

► **Lemma 59.** *Let H be an n -vertex k -partite k -hypergraph with vertex classes X_1, \dots, X_k . Let $0 < \zeta, p < 1$, let $I \subseteq [k]$, and suppose H has an (I, ζ) -core. For all $i \in I$, let $X'_i = X_i$; for all $i \in [k] \setminus I$, let X'_i be a random subset of X_i in which each element is included independently with probability p . Then, setting $H' = H[X'_1, \dots, X'_k]$, we have*

$$\mathbb{P}(e(H') < p^{k-|I|}e(H)/(2k)^{k+1}) \leq 2 \exp(-p^{2(k-|I|)}/(2k\zeta)).$$

Proof. Let $r := |I|$. Without loss of generality, suppose $I = [r]$ (otherwise we can reorder X_1, \dots, X_k). By hypothesis, H has an (I, ζ) -core; denote this by (Y_1, \dots, Y_k) . For all $i \in [k]$,

■ **Algorithm 4** `CoarseSmallCoreHelper`

This algorithm computes an estimate of the number of edges in G which is unlikely to be much too large and, if G has an (I, ζ) -core satisfying certain properties, is also unlikely to be much too small. Its running time increases with the size of the set I .

Oracle: Colourful independence oracle $\text{cIND}(G)$ of an n -vertex k -hypergraph G .

Input: Positive integer n that is a power of two, disjoint subsets

$X_1, \dots, X_k \subseteq V(G)$, set $I \subseteq [k]$, and integer t satisfying $n \geq t \geq 12 \log k$.

Output: Non-negative integer Z that satisfies the following properties with probability at least $2/3$, where $b := (2k)^{k+2}$. Firstly, $e(G[X_1, \dots, X_k]) \leq Zb$. Secondly, if $G[X_1, \dots, X_k]$ has an (I, ζ) -core for some ζ satisfying $t \geq (8k\zeta)^{1/(2(k-|I|))}n$, then $e(G[X_1, \dots, X_k]) \geq Z/b$.

```

1 begin
2   forall  $i \in I$  do
3     Compute an arbitrary partition  $X_{i,1}, \dots, X_{i,x_i}$  of  $X_i$  into  $x_i \leq \lceil n/t \rceil$  sets each
       of cardinality at most  $t$ .
4   forall  $i \in [k] \setminus I$  do
5     Compute a random subset  $X_{i,1} \subseteq X_i$  by retaining each element independently
       with probability  $t/n$ .
6     if  $|X_{i,1}| > 2t$  then
7       Return an arbitrary value and halt.
8   Compute the set  $\Pi$  of all functions  $\pi$  with domain  $[k]$  such that  $\pi(i) \in [x_i]$  for
       each  $i \in I$  and  $\pi(i) = 1$  for all  $i \notin I$ .
9   forall  $\pi \in \Pi$  do
10    Set  $H_\pi := G[X_{1,\pi(1)}, \dots, X_{k,\pi(k)}]$ .
11  return  $Z := \frac{n^{k-|I|}}{t^{k-|I|}} \sum_{\pi \in \Pi} \text{Count\_DLM22}(\text{cIND}(H_\pi), 1/2, 1/(12n^k))$ .
```

let $Y'_i = Y_i \cap X'_i$; thus, Y'_1, \dots, Y'_r are equal to Y_1, \dots, Y_r , and Y'_{r+1}, \dots, Y'_k are formed by randomly deleting vertices from Y_{r+1}, \dots, Y_k . Let $J := H[Y_1, \dots, Y_k]$ and $J' := H[Y'_1, \dots, Y'_k]$.

We next show that it suffices to prove concentration of $e(J')$. Observe by linearity of expectation applied to indicator random variables for each edge in $E(J)$ that $\mathbb{E}(e(J')) = p^{k-r}e(J)$. Moreover, observe that $e(H') \geq e(J')$, and recall from [Definition 53\(i\)](#) that since Y_1, \dots, Y_k is a core we have $e(J) \geq e(H)/(2k)^k$; thus whenever $e(J') \geq p^{k-r}e(J)/2$, we also have $e(H') \geq p^{k-r}e(H)/(2k)^{k+1}$. It follows that

$$\mathbb{P}(e(H') < p^{k-r}e(H)/(2k)^{k+1}) \leq \mathbb{P}(e(J') < p^{k-r}e(J)/2). \quad (4.1.10)$$

We now bound the right-hand side of (4.1.10) above by applying [Lemma 58](#) to $e(J')$. Observe that $e(J')$ is a function of the independent indicator variables for the events $\{v \in Y'_j\}$ for $j \geq r+1$, and that modifying any of those indicator variables — that is, adding or removing a vertex v from some Y'_j — affects $e(J')$ by at most $d_J(v)$. Thus by [Lemma 58](#),

$$\mathbb{P}(e(J') < p^{k-r}e(J)/2) \leq 2 \exp \left(-p^{2(k-r)}e(J)^2 / \left(2 \sum_{i=r+1}^k \sum_{v \in Y_i} d_J(v)^2 \right) \right). \quad (4.1.11)$$

Now, since (Y_1, \dots, Y_k) is an (I, ζ) -core for H , by [Definition 53\(iii\)](#) we have $d_J(v) \leq \zeta e(J)$ for all $v \in Y_{r+1} \cup \dots \cup Y_k$. Moreover, since H is k -partite we have $\sum_{v \in Y_i} d_J(v) = e(J)$ for all $i \in [k]$. Thus

$$\sum_{i=r+1}^k \sum_{v \in Y_i} d_J(v)^2 \leq \zeta e(J) \sum_{i=r+1}^k \sum_{v \in Y_i} d_J(v) \leq k\zeta e(J)^2.$$

By (4.1.10) and (4.1.11), it follows that

$$\mathbb{P}(e(H') < p^{2(k-|I|)}e(H)/(2k)^{k+1}) \leq 2 \exp(-p^{2(k-r)}/(2k\zeta)),$$

as required. ◀

We now prove correctness of [CoarseSmallCoreHelper](#).

► **Lemma 60.** [CoarseSmallCoreHelper](#) behaves as stated. Moreover, let

$$T := k^{6k+1} \log^{4k+8}(n) n^{|I|} / t^{|I|}.$$

Then on an n -vertex k -hypergraph G , [CoarseSmallCoreHelper](#)($\text{cIND}(G), X_1, \dots, X_k, I, t$) has oracle cost $\mathcal{O}(T \text{cost}_k(2kt))$ and running time $\mathcal{O}(Ttk \log n + n)$.

Proof. *Running time and oracle cost.* Since we halt early at line 7 whenever any set $X_{i,j}$ contains more than $2t$ vertices, [Count_DLM22](#) is only invoked on graphs with at most $2kt$ vertices. By [Theorem 57](#), it follows that each invocation of [Count_DLM22](#) runs in time $\mathcal{O}(k^{6k+1} \log^{4k+9}(n) \cdot 2kt)$ and has oracle cost $\mathcal{O}(k^{6k+1} \log^{4k+8}(n) \text{cost}_k(2kt))$. Since $|\Pi| = \mathcal{O}((n/t)^{|I|})$, it follows that line 11 runs in time $\mathcal{O}(Ttk \log n)$ and has oracle cost $\mathcal{O}(T \text{cost}_k(2kt))$. The rest of the algorithm runs in time $\mathcal{O}(nk)$ and does not call the oracle, so the desired bounds follow.

Correctness. Let $r := |I|$, and let $H := G[X_1, \dots, X_k]$. Without loss of generality, suppose that $I = [r]$ (otherwise we can reorder X_1, \dots, X_k). We will argue that, with high probability, none of the following bad events occur:

\mathcal{E}_1 : we halt at line 7 due to a set $X_{i,1}$ having size greater than $2t$;

- \mathcal{E}_2 : at least one invocation of `Count_DLM22` at line 11 returns a value that is not a $(1/2)$ -approximation to $e(H_\pi)$;
 \mathcal{E}_3 : we do not have $Z \leq b \cdot e(H)$;
 \mathcal{E}_4 : H has an (I, ζ) -core for some ζ satisfying $t \geq (8k\zeta)^{1/(2(k-r))}n$ and we do not have $Z \geq e(H)/b$.

If none of these events occur, then the algorithm behaves as stated. Hence

$$\mathbb{P}(\text{CoarseSmallCore fails}) \leq \mathbb{P}(\mathcal{E}_1) + \mathbb{P}(\mathcal{E}_2) + \mathbb{P}(\mathcal{E}_3 \mid \neg\mathcal{E}_1, \neg\mathcal{E}_2) + \mathbb{P}(\mathcal{E}_4 \mid \neg\mathcal{E}_1, \neg\mathcal{E}_2). \quad (4.1.12)$$

We first bound $\mathbb{P}(\mathcal{E}_1)$ above. For all $i \in I$, we have $|X_{i,j}| \leq t$ for all j by construction in line 3. For all $i \in [k] \setminus I$, we have $x_i = 1$, and $|X_{i,1}|$ follows a binomial distribution with mean $|X_i|t/n \leq t$. Since $t \geq 12 \log k$, it follows by a standard Chernoff bound (Lemma 13 with $\delta = 1$) that

$$\mathbb{P}(\mathcal{E}_1) \leq 2e^{-t/3} \leq 2/e^4 < 1/12. \quad (4.1.13)$$

We next observe that, by Theorem 57, each invocation of `Count_DLM22` fails with probability at most $1/(12n^k)$. There are $|\Pi| \leq \lceil n/t \rceil^{|I|} \leq n^k$ invocations in total, so by a union bound we have

$$\mathbb{P}(\mathcal{E}_2) \leq 1/12. \quad (4.1.14)$$

We next observe that, writing $H' = G[X_1, \dots, X_r, X_{r+1,1}, \dots, X_{k,1}]$, since H is k -partite, we have

$$e(H') = \sum_{\pi \in \Pi} e(H_\pi).$$

Hence, conditioned on $\neg\mathcal{E}_1$ and $\neg\mathcal{E}_2$, i.e. conditioned on `CoarseSmallCoreHelper` reaching line 11 and on the calls to `Count_DLM22` returning valid approximations, we have

$$(n/t)^{k-r} e(H')/2 \leq Z \leq 2(n/t)^{k-r} e(H'). \quad (4.1.15)$$

We now bound $\mathbb{P}(\mathcal{E}_3 \mid \neg\mathcal{E}_1, \neg\mathcal{E}_2)$ above using (4.1.15). Each edge of $G[X_1, \dots, X_k]$ survives in H' with probability $(t/n)^{k-r}$, so by linearity of expectation we have

$$\mu := \mathbb{E}(e(H') \mid \neg\mathcal{E}_1, \neg\mathcal{E}_2) = e(H)(t/n)^{k-r}.$$

It follows by the fact that $k \geq 2$, (4.1.15), and Markov's inequality that

$$\begin{aligned} \mathbb{P}(Z \geq e(H) \cdot b \mid \neg\mathcal{E}_1, \neg\mathcal{E}_2) &\leq \mathbb{P}(Z \geq 24\mu(n/t)^{k-r} \mid \neg\mathcal{E}_1, \neg\mathcal{E}_2) \\ &\leq \mathbb{P}(e(H') \geq 12\mu \mid \neg\mathcal{E}_1, \neg\mathcal{E}_2) \leq 1/12. \end{aligned} \quad (4.1.16)$$

Finally, we bound $\mathbb{P}(\mathcal{E}_4 \mid \neg\mathcal{E}_1, \neg\mathcal{E}_2)$ above. Suppose $G[X_1, \dots, X_k]$ has an (I, ζ) -core for some ζ satisfying $t \geq (8k\zeta)^{1/(2(k-|I|))}n$ (since otherwise this probability is zero). Using (4.1.15) and Lemma 59, taking $p = t/n$, we have

$$\begin{aligned} \mathbb{P}(\mathcal{E}_4 \mid \neg\mathcal{E}_1, \neg\mathcal{E}_2) &= \mathbb{P}(Z < e(H)/(2k)^{k+2} \mid \neg\mathcal{E}_1, \neg\mathcal{E}_2) \\ &\leq \mathbb{P}((n/t)^{k-r} e(H')/2 < e(H)/(2k)^{k+2} \mid \neg\mathcal{E}_1, \neg\mathcal{E}_2) \\ &\leq \mathbb{P}(e(H') < (t/n)^{k-r} e(H)/(2k)^{k+1} \mid \neg\mathcal{E}_1, \neg\mathcal{E}_2) \\ &\leq 2 \exp(-(t/n)^{2(k-r)}/(2k\zeta)). \end{aligned}$$

Since $t \geq (8k\zeta)^{1/(2(k-r))}n$ by hypothesis, it follows that

$$\mathbb{P}(\mathcal{E}_4 \mid \neg\mathcal{E}_1, \neg\mathcal{E}_2) \leq 2e^{-4} < 1/12. \quad (4.1.17)$$

The result now follows from (4.1.12), (4.1.13), (4.1.14), (4.1.16) and (4.1.17). \blacktriangleleft

Lemma 54 now follows immediately from Lemma 60 together with Lemma 19, which is used to reduce the failure probability from $1/3$ to $1 - \delta$ for arbitrary rational $\delta \in (0, 1)$.

4.1.3 Counting edges with a large core

In this section, we adapt the algorithm `ColourCoarse_DLM22` of [15] into `CoarseLargeCore`, which runs faster but still gives a good approximation when supplied with a set I and a rational number ζ such that the input graph $H = G[X_1, \dots, X_k]$ has an (I, ζ) -core. The crucial ingredient in [15] is a subroutine `VerifyGuess` which takes as input a guess M of the number of edges in H , and distinguishes — with reasonable probability — two cases: the number of edges is at least M , or the number of edges is *much* less than M . It is then relatively easy to use this subroutine to implement the algorithm of [15] by using binary search to find the least guess M which `VerifyGuess` accepts. Our adaptation follows exactly the same structure; we first present the analogue of `VerifyGuess`.

■ **Algorithm 5** `VerifyGuess_New`

This algorithm, with reasonable probability, distinguishes between two cases: the number of edges in the input hypergraph is much less than the guess M , or the input hypergraph contains an (I, ζ) -core and its number of edges is at least M .

Oracle: Colourful independence oracle $\text{cIND}(G)$ of an n -vertex k -hypergraph G .

Input: Positive integers n and M that are powers of two and satisfy $n \geq 32$, disjoint subsets $X_1, \dots, X_k \subseteq V(G)$, set $I \subseteq [k]$, and rational number $\zeta \in (0, 1/32)$ with denominator $\mathcal{O}(n^k)$.

Output: Either **Yes** or **No**. Setting

$$p_{\text{out}} = 1/(2^{5k} \log^{|I|}(1/\zeta) \log^{k-|I|} n),$$

we require the following two properties. *Completeness* ensures that if $G[X_1, \dots, X_k]$ contains an (I, ζ) -core and $e(G[X_1, \dots, X_k]) \geq M$, then `VerifyGuess_New` outputs **Yes** with probability at least p_{out} . *Soundness* ensures that if $e(G[X_1, \dots, X_k]) < M \cdot p_{\text{out}} / ((8k)^k \log^{|I|}(1/\zeta) \log^{k-|I|} n)$, then `VerifyGuess_New` outputs **No** with probability at least $1 - p_{\text{out}}/2$.

```

1 begin
2   For each  $i \in [k]$  and each  $0 \leq j \leq 2 \log n - 1$ , construct a subset  $Z_{i,j}$  of  $X_i$  by
     including each vertex independently with probability  $1/2^j$ .
3   Construct the finite set  $A$  of all tuples  $(a_1, \dots, a_k)$  of non-negative integers
     satisfying:  $a_i \leq 2 \log n$  for all  $i \in [k]$ ;  $a_i \leq 2 \log(1/\zeta) + 1$  for all  $i \in I$ ; and
      $a_1 + \dots + a_k \geq \log M - k \log(2k)$ .
4   forall  $(a_1, \dots, a_k) \in A$  do
5     if  $\text{cIND}(G)_{Z_{1,a_1}, \dots, Z_{k,a_k}} = 0$  then
6       return Yes.
7   return No.
```

The main difference between `VerifyGuess_New` and the algorithm `VerifyGuess` of [15] is the choice of the set A , which is much smaller; this is possible because of the (I, ζ) -core, and leads to an improved running time and oracle cost. The main difference in the proofs is that we must show that the presence of the core implies that completeness still holds even with this smaller set A , i.e. that if $E(G[X_1, \dots, X_k]) \geq M$ then with reasonable probability

there still exists $(a_1, \dots, a_k) \in A$ such that $e(G[Z_{1,a_1}, \dots, Z_{k,a_k}]) > 0$ and the algorithm outputs **Yes** in line 6. The following lemma is a generalisation of part of the proof of [15, Lemma 4.1], and we will use it to find this tuple (a_1, \dots, a_k) .

► **Lemma 61.** *Let J be a k -partite k -hypergraph with vertex classes C_1, \dots, C_k , and let $i \in [k]$. Let $\Lambda = \max\{5, \log |C_i|\}$. Then there exists a integer $0 \leq a \leq 2\Lambda - 1$ and a set $S \subseteq C_i$ such that:*

- (i) *for all $v \in S$, $d_J(v) \geq e(J)/2^a$; and*
- (ii) *$2^{-a}|S| \geq 1/(16\Lambda)$.*

Proof. Without loss of generality, suppose $i = 1$ (by reordering C_1, \dots, C_k if necessary). We first throw away every vertex in C_1 with degree significantly lower than average; let $C_1^- := \{v \in C_1 : d_J(v) > e(J)/(2|C_1|)\}$, and let $J^- := J[C_1^-, C_2, \dots, C_k]$. Observe that

$$e(J^-) = e(J) - \sum_{v \in C_1 \setminus C_1^-} d_J(v) \geq e(J) - |C_1| \cdot e(J)/(2|C_1|) = e(J)/2, \quad (4.1.18)$$

and that $d_{J^-}(v) = d_J(v)$ for all $v \in C_1^-$.

We now partition vertices of C_1^- according to their degree. For all integers $d \geq 1$, let

$$C_1^d := \{v \in C_1^- : 2^{d-1} \leq d_{J^-}(v) < 2^d\}.$$

Thus C_1^d is the set of vertices in C_1^- with degree roughly 2^d in J^- (or equivalently in J). By the definition of C_1^- , for all $v \in C_1^-$ we have $e(J)/(2|C_1|) < d_{J^-}(v) \leq e(J)$; hence $C_1^d = \emptyset$ for all values of d not satisfying $2^d > e(J)/(2|C_1|)$ and $2^{d-1} \leq e(J)$. Since each edge in J^- is incident to a vertex in exactly one set C_1^d , by the pigeonhole principle, we deduce that there exists

$$D \in \left[1 + \left\lfloor \log \frac{e(J)}{2|C_1|} \right\rfloor, 1 + \lfloor \log e(J) \rfloor\right] \quad (4.1.19)$$

such that

$$\begin{aligned} e(J^-[C_1^D, C_2, \dots, C_k]) &\geq \frac{e(J^-)}{[\log e(J)] - [\log(e(J)/(2|C_1|))] + 1} \\ &\geq \frac{e(J^-)}{\log(e(J)) - \log(e(J)/(2|C_1|)) + 2} = \frac{e(J^-)}{3 + \log |C_1|}. \end{aligned}$$

It follows by (4.1.18) that

$$e(J^-[C_1^D, C_2, \dots, C_k]) \geq \frac{e(J)}{6 + 2 \log |C_1|}. \quad (4.1.20)$$

We now take $S := C_1^D$ and $a := \lceil \log e(J) \rceil - D + 1$. It remains to prove that S and a satisfy the conditions required by the lemma statement.

First, observe from (4.1.19) that

$$0 \leq a \leq \lceil \log e(J) \rceil - \left\lfloor \log \frac{e(J)}{2|C_1|} \right\rfloor \leq \log e(J) - \log \frac{e(J)}{2|C_1|} + 2 < 2\Lambda - 1,$$

as required in the lemma statement. We next prove (i). Since every vertex $v \in C_1^D$ has degree at least 2^{D-1} in J^- , from the definition of a we have

$$2^a d_J(v) \geq 2^{a+D-1} = 2^{\lceil \log e(J) \rceil} \geq e(J),$$

as required by (i). Finally, we prove (ii). Since every vertex in C_1^D has degree at most 2^D in J^- , by (4.1.20) we have

$$2^D |C_1^D| \geq e(J^-[C_1^D, C_2, \dots, C_k]) \geq \frac{e(J)}{6 + 2 \log |C_1|},$$

and hence

$$|C_1^D| \geq \frac{e(J)}{2^{D+1}(3 + \log |C_1|)} \geq \frac{e(J)}{2^{D+2}\Lambda}.$$

By the definition of a we have

$$2^{-a} \geq 2^{-\lceil \log e(J) \rceil + D - 1} \geq \frac{2^{D-2}}{e(J)},$$

so it follows that $2^{-a} |C_1^D| \geq 1/(16\Lambda)$ as required. \blacktriangleleft

The rest of the analysis of `VerifyGuess_New` is almost exactly the same as in [15], with the relevant part of the proof replaced by Lemma 61 and with slightly different algebra. We provide full details for the benefit of the reader.

► **Lemma 62.** *`VerifyGuess_New` behaves as stated, runs in time*

$$\mathcal{O}(2^{2k}(\log n)^{k-|I|} \log^{|I|}(1/\zeta) + kn \log n),$$

and has oracle cost

$$\mathcal{O}(2^{2k}(\log n)^{k-|I|} \log^{|I|}(1/\zeta) \cdot \text{cost}_k(n)).$$

Proof. Let $G, M, X_1, \dots, X_k, I, \zeta$ be the input for `VerifyGuess_New`, and write $H = G[X_1, \dots, X_k]$.

Running time and oracle cost. We first observe that

$$|A| \leq 2^k(\log(1/\zeta) + 1)^{|I|}(\log n)^{k-|I|}.$$

Since $\log(1/\zeta) \geq 5$, we have $\log(1/\zeta) + 1 \leq (6/5)\log(1/\zeta)$ and hence

$$|A| \leq (12/5)^k \log^{|I|}(1/\zeta)(\log n)^{k-|I|} = \mathcal{O}(2^{2k}(\log n)^{k-|I|} \log^{|I|}(1/\zeta)). \quad (4.1.21)$$

Lines 2 and 3 make no oracle calls; line 2 takes $\mathcal{O}(kn \log n)$ time to construct the sets $Z_{i,j}$, and line 3 takes $\mathcal{O}(|A|)$ time to construct A . The loop at line 4 invokes the oracle $|A|$ times and runs in $\mathcal{O}(|A|)$ time. Line 7 makes no oracle calls and runs in constant time. The claimed bounds on running time and oracle cost follow on bounding the cost of every oracle call by $\text{cost}_k(n)$ (using Lemma 12), and it remains to prove that the soundness and completeness properties hold.

Soundness. This follows by an identical argument to that used in the proof of [15, Lemma 4.1]; for clarity we reproduce this reasoning with the values of p_{out} and $|A|$ changed for the situation at hand. For notational convenience, we denote the gap in the soundness case by γ , that is, we set $\gamma := p_{\text{out}} / ((8k)^k \log^{|I|}(1/\zeta) \log^{k-|I|} n)$. Suppose that $e(H) < \gamma M$.

Recall that the algorithm returns **Yes** if and only if $e(G[Z_{1,a_1}, \dots, Z_{k,a_k}]) > 0$ holds for some $(a_1, \dots, a_k) \in A$. Hence by a union bound over all $e \in E(H)$ and all $(a_1, \dots, a_k) \in A$, we have

$$\mathbb{P}(\text{Returns No}) \geq 1 - \sum_{(a_1, \dots, a_k) \in A} e(H) \prod_{j=1}^k 2^{-a_j}. \quad (4.1.22)$$

We now bound this product above. By the definition of A , we have $2^{-a_1} \dots 2^{-a_k} \leq (2k)^k/M$. By hypothesis, we have $e(H) < \gamma M$ and hence $(2k)^k/M \leq \gamma(2k)^k/e(H)$. It follows from (4.1.22) that

$$\mathbb{P}(\text{Returns No}) \geq 1 - |A|\gamma(2k)^k. \quad (4.1.23)$$

By (4.1.21) and the fact that $k \geq 2$ we have

$$\gamma \leq \frac{p_{\text{out}}}{(40/12)^k k^k |A|} < \frac{p_{\text{out}}}{2^{k+1} k^k |A|}.$$

It therefore follows from (4.1.23) that we return **No** with probability at least $1 - p_{\text{out}}/2$. This establishes the soundness of the algorithm, so it remains to prove completeness.

Completeness. Suppose now that (Y_1, \dots, Y_k) is an (I, ζ) -core for H and that $e(H) \geq M$. We must prove that **VerifyGuess_New** outputs **Yes** with probability at least p_{out} . Let $H_0 := H[Y_1, \dots, Y_k]$, and note that by Definition 53(i), $e(H_0) \geq e(H)/(2k)^k \geq M/(2k)^k$. It suffices to show that with probability at least p_{out} , there is at least one setting of the vector $(a_1, \dots, a_k) \in A$ such that $H_0[Z_{1,a_1}, \dots, Z_{k,a_k}]$ contains at least one edge.

We will define this setting iteratively. First, with reasonable probability, we will find an integer a_1 and a vertex $v_1 \in Z_{1,a_1}$ such that $H_1 := H_0[\{v_1\}, Y_2, \dots, Y_k]$ contains roughly $2^{-a_1}e(H_0)$ edges; our choice of a_1 will come from an application of Lemma 61. In the process, we expose $Z_{1,j}$ for all j . We then, again with reasonable probability, find an integer a_2 and a vertex $v_2 \in Z_{2,a_2}$ such that $H_2 := H_0[\{v_1\}, \{v_2\}, Y_3, \dots, Y_k]$ contains roughly $2^{-a_1-a_2}e(H')$ edges. Continuing in this vein, we eventually find $(a_1, \dots, a_k) \in A$ and vertices $v_i \in Z_{i,a_i}$ such that $\{v_1, \dots, v_k\}$ is an edge in $H_0[Z_{1,a_1}, \dots, Z_{k,a_k}]$, proving the result.

We formalise this idea by defining a collection of events. For all $i \in [k]$, let \mathcal{E}_i be the event that there exist integers $a_1, \dots, a_i \geq 0$ and $v_1, \dots, v_i \in V(H)$ such that:

- (a) for all $j \in [i]$, $v_j \in Z_{j,a_j}$;
- (b) for all $j \in [i] \setminus I$, $a_j \leq 2 \log(n) - 1$;
- (c) for all $j \in [i] \cap I$, $a_j \leq 2 \log(1/\zeta) + 1$; and
- (d) setting $H_i := H_0[\{v_1\}, \dots, \{v_i\}, Y_{i+1}, \dots, Y_k]$, we have $e(H_i) \geq e(H_0)/\prod_{j=1}^i 2^{a_j}$.

We make the following **Claim**: for all $i \in [k]$,

$$\mathbb{P}(\mathcal{E}_i \mid \mathcal{E}_1, \dots, \mathcal{E}_{i-1}) \geq \begin{cases} 1/(32 \log(1/\zeta)) & \text{if } i \in I, \\ 1/(32 \log n) & \text{otherwise.} \end{cases}$$

Note that for $i = 1$, the range $\mathcal{E}_1, \dots, \mathcal{E}_{i-1}$ is empty.

Proof of Lemma 62 from Claim: Suppose \mathcal{E}_k occurs, and let a_1, \dots, a_k and v_1, \dots, v_k be as in the definition of \mathcal{E}_k . By (d), we know that $\{v_1, \dots, v_k\}$ is an edge in H ; it follows by (a) that it is also an edge in $G[Z_{1,a_1}, \dots, Z_{k,a_k}]$. Also by (d), since the number of edges containing $\{v_1, \dots, v_k\}$ cannot be more than one, we have

$$\prod_{j=1}^k 2^{a_j} \geq e(H_0). \quad (4.1.24)$$

Since $e(H) \geq M$ by hypothesis, and (Y_1, \dots, Y_k) is an (I, ζ) -core, by Definition 53(i) we have $e(H_0) \geq M/(2k)^k$; it follows from (4.1.24) that $a_1 + \dots + a_k \geq \log M - k \log(2k)$. It follows from (b) and (c) that $(a_1, \dots, a_k) \in A$, so whenever \mathcal{E}_k occurs, **VerifyGuess_New** returns **Yes** on reaching (a_1, \dots, a_k) in line 4. By the Claim, we have

$$\mathbb{P}(\mathcal{E}_k) = \prod_{j=1}^k \mathbb{P}(\mathcal{E}_j \mid \mathcal{E}_1, \dots, \mathcal{E}_{j-1}) \geq 1/(32^k \log^{|I|}(1/\zeta) \log^{k-|I|} n) = p_{\text{out}},$$

so completeness follows. The lemma statement therefore follows as well.

Proof of Claim: We proceed by induction. Suppose we have defined v_1, \dots, v_{i-1} and a_1, \dots, a_{i-1} as a deterministic function of $\{Z_{i',j} : i' \leq i-1\}$ conditioned on $\mathcal{E}_1, \dots, \mathcal{E}_{i-1}$; this is vacuously true for $i = 1$, and true by the induction hypothesis for $2 \leq i \leq k$. We then expose the values of $\{Z_{i',j} : i' \leq i-1\}$ conditioned on $\mathcal{E}_1, \dots, \mathcal{E}_{i-1}$, and hence the values of v_1, \dots, v_{i-1} and a_1, \dots, a_{i-1} ; we abuse notation slightly by abbreviating the corresponding suite of events to \mathcal{F} and identifying a_1, \dots, a_{i-1} , v_1, \dots, v_{i-1} , and H_{i-1} with their values conditioned on \mathcal{F} . We seek to prove that for all choices of \mathcal{F} ,

$$\mathbb{P}(\mathcal{E}_i \mid \mathcal{F}) \geq \begin{cases} 1/(32 \log(1/\zeta)) & \text{if } i \in I, \\ 1/(32 \log n) & \text{otherwise.} \end{cases}$$

Observe that \mathcal{F} is independent of all sets $Z_{i,j}$.

We apply [Lemma 61](#) with $J = H_{i-1}$, so that $C_a = \{v_a\}$ for all $a \leq i-1$ and $C_a = Y_a$ for all $a \geq i$. We take a_i to be the resulting integer a , and $S_i \subseteq C_i$ to be the resulting set S . We will take \mathcal{E}_i to be the event that $S_i \cap Z_{i,a_i} \neq \emptyset$, and if \mathcal{E}_i occurs then we will choose v_i arbitrarily from $S_i \cap Z_{i,a_i}$. By construction, v_1, \dots, v_i satisfy property (a).

We first note that by [Lemma 61](#), $a_i \leq \max\{9, 2 \log(|Y_i|) - 1\}$. If $i \notin I$, then it follows that $a_i \leq \max\{9, 2 \log n - 1\}$; since $n \geq 32$, it follows that $a_i \leq 2 \log n - 1$. If $i \in I$, then since (Y_1, \dots, Y_k) is an (I, ζ) -core of H_0 , by [Definition 53\(ii\)](#), we have $|Y_i| \leq 2/\zeta$ and hence $a_i \leq \max\{9, 2 \log(1/\zeta) + 1\}$; since $\zeta \leq 1/32$, it follows that $a_i \leq 2 \log(1/\zeta) + 1$. Either way, a_1, \dots, a_i satisfy properties (b) and (c).

We next observe that if \mathcal{E}_i occurs, so that there exists some $v_i \in S_i \cap Z_{i,a_i}$, then by [Lemma 61\(i\)](#) and property (d) of H_{i-1} , we have

$$e(H_i) = d_{H_{i-1}}(v_i) \geq e(H_{i-1})/2^{a_i} \geq e(H_0)/\prod_{j=1}^i 2^{a_j}.$$

Thus property (d) is also satisfied for H_i .

Finally, we observe that since all sets $Z_{i,j}$ are independent of \mathcal{F} , we have

$$\mathbb{P}(\mathcal{E}_i \mid \mathcal{F}) = \mathbb{P}(Z_{i,a_i} \cap S_i \neq \emptyset) = 1 - (1 - 2^{-a_i})^{|S_i|} \geq 1 - e^{-2^{-a_i}|S_i|}.$$

By [Lemma 61\(ii\)](#), it follows that

$$\mathbb{P}(\mathcal{E}_i \mid \mathcal{F}) \geq 1 - e^{-1/(16 \max\{5, \log |Y_i|\})} \geq \frac{1}{32 \max\{5, \log |Y_i|\}}.$$

As before, since (Y_1, \dots, Y_k) is an (I, ζ) -core, $n \geq 32$ and $\zeta \leq 1/32$, the required lower bound follows whether $i \in I$ or not. \blacktriangleleft

In [\[15\]](#), `VerifyGuess` is used as a subroutine by an algorithm `ColourCoarse_DLM22`: this algorithm makes repeated calls to `VerifyGuess` for each $M \in \{1, 2, 4, 8, \dots, n^k\}$, and outputs an estimate of the number of edges in G that, with probability at least $2/3$, is a b -approximation. We mimic this behaviour with the following algorithm.

► **Lemma 63.** `CoarseLargeCoreHelper` behaves as stated. Moreover, let

$$T = 2^{7k} \log^{2|I|}(1/\zeta)(\log n)^{2(k-|I|)+1}n.$$

Then on an n -vertex k -hypergraph G , `CoarseLargeCoreHelper`(`cIND`(G), $X_1, \dots, X_t, I, \zeta$) runs in time $\mathcal{O}(Tn \log n)$, and has oracle cost $\mathcal{O}(T \text{cost}_k(n))$.

■ **Algorithm 6** `CoarseLargeCoreHelper`

This algorithm makes repeated calls to `VerifyGuess_New` with different guesses M , to obtain a coarse approximation to the number of edges in the input hypergraph, assuming that it contains an (I, ζ) -core.

Oracle: Colourful independence oracle $\text{cIND}(G)$ of an n -vertex k -hypergraph G .

Input: Integer $n \geq 32$ that is a power of two, disjoint subsets $X_1, \dots, X_k \subseteq V(G)$, set $I \subseteq [k]$, and rational number $\zeta \in (0, 1/32)$ with denominator $\mathcal{O}(n^k)$.

Output: Non-negative integer m such that, setting $b := (2k)^{5k} \log^{k-|I|} n \log^{|I|}(1/\zeta)$, m satisfies both of the following properties with probability at least $2/3$. Firstly, $e(G[X_1, \dots, X_k]) \leq mb$. Secondly, if $G[X_1, \dots, X_k]$ has an (I, ζ) -core then $e(G[X_1, \dots, X_k]) \geq m/b$.

```

1 begin
2   Set  $p_{\text{out}} := 1/(2^{5k} \log^{|I|}(1/\zeta) \log^{k-|I|} n)$ .
3   Calculate  $N := \lceil 24 \ln(12k \log n)/p_{\text{out}} \rceil$ .
4   forall  $M \in \{1, 2, 4, 8, \dots, n^k\}$  do
5     Call VerifyGuess_New( $\text{cIND}(G), M, X_1, \dots, X_k, I, \zeta$ ) a total of  $N$  times, and
       let  $S_M \in \{0, \dots, N\}$  be the number of calls that returned Yes. (Naturally,
       we use independent randomness for each call.)
6   if  $\text{cIND}(G)_{X_1, \dots, X_k} = 1$  then
7     Set  $m = 0$ .
8   else
9     if there exists  $M$  such that  $S_M \geq 3p_{\text{out}}N/4$  then
10      Let  $m$  be the greatest such  $M$ .
11    else
12      Set  $m = n^k$ .
13  return  $2m/b$ .
```

Proof. The proof is similar to that of [15, Lemma 4.2], but we include full details here for the sake of completeness.

Running time. For brevity, let $X := \log^{|I|}(1/\zeta) \log^{k-|I|} n$. `CoarseLargeCoreHelper` simply executes `VerifyGuess_New` N times, where $N = \mathcal{O}(2^{5k} X \log n)$, and performs $o(N)$ arithmetic operations to calculate N . By Lemma 62, each execution takes time $\mathcal{O}(2^{2k} X + kn \log n) = \mathcal{O}(2^{2k} X \log(n) \cdot n)$ and has oracle cost $\mathcal{O}(2^{2k} X \cdot \text{cost}_k(n))$. The overall running time of `CoarseLargeCoreHelper` is therefore $\mathcal{O}(2^{7k} X^2 \log^2(n) \cdot n)$, and the overall oracle cost is $\mathcal{O}(2^{7k} X^2 \log(n) \cdot \text{cost}_k(n))$, as claimed.

Correctness. Let $H := G[X_1, \dots, X_k]$. We will demonstrate that:

- (a) $\mathbb{P}(2m/b > b \cdot e(H)) \leq 1/6$; and
- (b) if H contains an (I, ζ) -core, then $\mathbb{P}(2m/b < e(H)/b) \leq 1/6$.

Given (a) and (b), correctness follows immediately by a union bound.

We first prove (a). Fix $M \in \{1, 2, 4, 8, \dots, n^k\}$ with $e(H) < 2M/b^2$. Observe that for such an M , `VerifyGuess_New` outputs `Yes` with probability at most $p_{\text{out}}/2$; thus the random variable S_M is a binomial variable with mean at most $Np_{\text{out}}/2$. A standard Chernoff bound (Lemma 13 with $\delta = 1/2$) then implies

$$\mathbb{P}(S_M \geq 3Np_{\text{out}}/4) \leq 2e^{-Np_{\text{out}}/24} \leq 1/(6k \log n).$$

Thus on taking a union bound over all such M , with probability at least $5/6$, we have $e(H) \geq 2m/b^2$ and hence our output, $2m/b$, is at most $b \cdot e(H)$ as required.

Next, suppose that H contains an (I, ζ) -core; we must prove (b). Let M^- be the maximum value in $\{1, 2, 4, \dots, n^k\}$ with $e(H) \geq M^-$, so that $M^- \leq e(H) < 2M^-$. Observe that for M^- , `VerifyGuess_New` outputs `Yes` with probability at least p_{out} ; hence $\mathbb{E}(S_{M^-}) \geq Np_{\text{out}}$, and so a standard Chernoff bound (Lemma 13 with $\delta = 1/4$) implies

$$\mathbb{P}(S_{M^-} < 3Np_{\text{out}}/4) \leq 2e^{-Np_{\text{out}}/48} \leq 1/6.$$

Thus with probability at least $5/6$, we have $m \geq M^-$ and hence $e(H) \leq 2m$. It follows that our output, $2m/b$, is at least $e(H)/b$. \blacktriangleleft

Lemma 55 now follows immediately from Lemma 63 together with Lemma 19, which is used to reduce the failure probability from $1/3$ to $1 - \delta$ for arbitrary rational $\delta \in (0, 1)$.

4.2 Lower bounds on oracle algorithms for edge detection

In order to prove the lower bound part of Theorem 2 in the regimes where k or α are large, we use a simple lower bound on cIND-oracle algorithms for the problem of deciding whether the given k -hypergraph has at least one edge or whether it is empty. In this section, we prove that simple lower bound as Proposition 65 and state the form we will need as Corollary 66.

For the following lemma, recall that a deterministic cIND-oracle algorithm makes a sequence S_1, \dots, S_N of queries to the oracle, where each query $S_i = (S_{i,1}, \dots, S_{i,k})$ is a tuple of disjoint vertex subsets, and recall that $S_i^{(k)}$ is the set of all possible edges in a k -partite k -hypergraph spanned by $S_{i,1}, \dots, S_{i,k}$. In the lemma, we show that N deterministic queries of overall bounded cost can detect a random edge e only with small probability.

► **Lemma 64.** *Let $\text{cost}_k(n) = n^{\alpha_k}$ be a cost function, where $\alpha_k \in [0, k]$. Let G be an n -vertex k -hypergraph, let S_1, \dots, S_N be an arbitrary sequence of queries to cIND(G) with total cost at most C , and let $e \in V(G)^{(k)}$ be sampled uniformly random. Then*

$$\mathbb{P}_e \left(e \in \bigcup_{i=1}^N S_i^{(k)} \right) \leq C/n^{\alpha_k}.$$

Proof. For all $i \in [N]$, let $s_i := \sum_{j=1}^k |S_{i,j}| \in [0, n]$. Then the total cost incurred by the N queries satisfies $\sum_{i=1}^N \text{cost}(S_i) = \sum_{i=1}^N s_i^{\alpha_k} \leq C$. By the AM-GM inequality, we have

$$|S_i^{(k)}| = \prod_{j=1}^k |S_{i,j}| \leq s_i^k / k^k.$$

By a union bound, it follows that

$$\mathbb{P}\left(e \in \bigcup_{i=1}^N S_i^{(k)}\right) \leq \binom{n}{k}^{-1} \sum_{i=1}^N |S_i^{(k)}| \leq \frac{k^k}{n^k} \sum_{i=1}^N \frac{s_i^k}{k^k} = \frac{1}{n^k} \sum_{i=1}^N s_i^k.$$

By assumption, we have $\sum_{i=1}^N s_i^{\alpha_k} \leq C$ and $s_i \in [0, n]$ for all i . We now apply Karamata's inequality in the form of [Corollary 22](#), taking $\alpha = \alpha_k$, $W = C$, $c = n$, $t = N$ and $r = k$. This yields:

$$\mathbb{P}\left(e \in \bigcup_{i=1}^N S_i^{(k)}\right) \leq \frac{C n^{k-\alpha_k}}{n^k} = \frac{C}{n^{\alpha_k}}. \quad \blacktriangleleft$$

Next, we use [Lemma 64](#) to show that any a randomised **cIND**-oracle algorithm must incur relatively high cost in order to distinguish the empty k -hypergraph from a k -hypergraph that is not empty.

► **Proposition 65.** *Let $\text{cost}_k(n) = n^{\alpha_k}$, where $\alpha_k \in [0, k]$, and let $p \in (0, 1]$. Let \mathcal{A} be a randomised **cIND**-oracle algorithm with worst-case oracle cost at most C such that, for all k -hypergraphs G , with probability at least p , $\mathcal{A}(\text{cIND}(G))$ returns 1 if and only if $e(G) > 0$. Then $C \geq pn^{\alpha_k}$.*

Proof. Let G_1 be the k -hypergraph on $[n]$ with no edges, and let G_2 be the k -hypergraph on $[n]$ with exactly one edge, chosen uniformly at random. Let A be a deterministic **cIND**-oracle algorithm with worst-case oracle cost at most C , and suppose $A(\text{cIND}(G_1)) \neq A(\text{cIND}(G_2))$ with probability at least p . Let $S_1(A), \dots, S_{N_A}(A)$ be the (deterministic) sequence of oracle queries issued by $A(\text{cIND}(G_1))$. On input G_2 , the deterministic algorithm $A(\text{cIND}(G_2))$ will make exactly the same queries until a query $S_i(A)$ potentially contains the planted random edge e . Thus by [Lemma 64](#), we have

$$p \leq \mathbb{P}_{G_2}\left(A(\text{cIND}(G_1)) \neq A(\text{cIND}(G_2))\right) \leq \mathbb{P}_e\left(e \in \bigcup_{i=1}^{N_A} S_i(A)^{(k)}\right) \leq C/n^{\alpha_k},$$

and hence $C \geq pn^{\alpha_k}$. Let F be the family of all such algorithms A .

We have $\mathbb{P}_{\mathcal{A}, G_2}(\mathcal{A}(\text{cIND}(G_1)) \neq \mathcal{A}(\text{cIND}(G_2))) \geq p$, so we must have $\mathcal{A} \in F$ with non-zero probability. Thus the worst-case oracle cost of \mathcal{A} is at least pn^{α_k} , as required. \blacktriangleleft

In the following corollary, we lift [Proposition 65](#) to worst-case *expected* oracle cost.

► **Corollary 66.** *Let $\text{cost}_k(n) = n^{\alpha_k}$, where $\alpha_k \in [0, k]$, and let $p \in (0, 1]$. Let \mathcal{A} be a randomised **cIND**-oracle algorithm with worst-case expected oracle cost at most C such that, for all k -hypergraphs G , with probability at least $9/10$, $\mathcal{A}(\text{cIND}(G))$ returns 1 if and only if $e(G) > 0$. Then $C \geq n^{\alpha_k}/100$.*

Proof. Suppose for a contradiction that such an algorithm \mathcal{A} existed with $C \leq n^{\alpha_k}/100$; then consider the algorithm \mathcal{A}' , which simulates \mathcal{A} , keeping track of the oracle costs. It aborts the

simulation of \mathcal{A} just before the total oracle cost would exceed $n^{\alpha_k}/10$. If \mathcal{A} has terminated normally, \mathcal{A}' copies the output, and otherwise \mathcal{A}' outputs an arbitrary value. By construction, \mathcal{A}' has worst-case oracle cost at most $n^{\alpha_k}/10$. By Markov's inequality, the probability that \mathcal{A} incurs cost at least $n^{\alpha_k}/10$ is at most $1/10$, and so \mathcal{A}' has success probability $p \geq 4/5$. By [Proposition 65](#), such an algorithm \mathcal{A}' cannot exist, a contradiction. \blacktriangleleft

4.3 Lower bounds on oracle algorithms for edge estimation

In this section, we unconditionally prove that the worst-case oracle cost achieved by `Count` is essentially optimal. As with [Theorem 40](#), we will construct two correlated random n -vertex k -hypergraphs G_1 and G_2 that (with high probability) have significantly different numbers of edges, but such that any deterministic `cIND`-oracle algorithm that can distinguish between G_1 and G_2 must incur a large worst-case oracle cost. This approach will yield the following result.

► **Theorem 67.** *Let $t, k \geq 1$, let $\alpha \in [0, k-3]$, and let $\text{cost}(x) = x^\alpha$. Let $t_0 := 2^{400k^6}$, and suppose that $t \geq t_0$. There exist two correlated distributions \mathcal{G}_1 and \mathcal{G}_2 on k -partite k -hypergraphs whose vertex classes V_1, \dots, V_k each have size t with the following properties:*

- (i) *We have $\mathbb{P}_{(G_1, G_2) \sim (\mathcal{G}_1, \mathcal{G}_2)}[e(G_2) \geq 4e(G_1)] \geq 19/20$.*
- (ii) *Let*

$$C := \frac{t^\alpha}{2^{5k+7} k^{7k}} \cdot \left(\frac{\log t}{\log \log t} \right)^{k - \lfloor \alpha \rfloor - 3}.$$

Suppose A is a deterministic `cIND`-oracle algorithm with

$$\mathbb{P}_{(G_1, G_2) \sim (\mathcal{G}_1, \mathcal{G}_2)} \left(A(\text{cIND}(G_1)) \neq A(\text{cIND}(G_2)) \right) \geq 2/3, \quad (4.3.1)$$

which only uses `cIND`-queries $S = (S_1, \dots, S_k)$ with $S_i \subseteq V_i$ for all $i \in [k]$. Then the expected oracle cost of A (with respect to cost) under random inputs $G_1 \sim \mathcal{G}_1$ satisfies $\mathbb{E}_{G_1 \sim \mathcal{G}_1}[\text{cost}(A, G_1)] \geq C/2$.

Before we prove [Theorem 67](#) in [Sections 4.3.1–4.3.3](#), let us apply the minimax principle ([Theorem 17](#)) to it, in order to derive our main lower bound for `cIND`-oracle algorithms.

► **Theorem 68.** *Let n, k be positive integers, and let $\text{cost}_k(n) = n^{\alpha_k}$ be a cost function, where $\alpha_k \in [0, k]$. Let \mathcal{A} be a randomised `cIND`-oracle algorithm such that, for all n -vertex k -hypergraphs G , $\mathcal{A}(\text{cIND}(G))$ is a $(1/2)$ -approximation to $e(G)$ with probability at least $9/10$. Then \mathcal{A} has worst-case expected oracle cost at least $\Omega(L(n, k))$, where*

$$L(n, k) := \frac{n^{\alpha_k}}{k^{10k}} \cdot \left(\frac{\log n}{\log \log n} \right)^{k - \lfloor \alpha_k \rfloor - 3}.$$

Proof. Let $\mathcal{C}(n, k)$ be the worst-case expected oracle cost of \mathcal{A} on n -vertex k -hypergraphs. We must show that $\mathcal{C}(n, k) \geq \Omega(L(n, k))$ holds. Note that [Corollary 66](#) already gives a simple bound of $\mathcal{C}(n, k) \geq n^\alpha/100$, where we write $\alpha = \alpha_k$. We bound $\mathcal{C}(n, k)$ by a case distinction depending on the values of n , k , and α .

Case 1: $\alpha \geq k-3$. In this case, we have $(\log n)^{k - \lfloor \alpha \rfloor - 3} \leq 1$, so $L(n, k) \leq n^\alpha$, thus the bound from [Corollary 66](#) already suffices.

Case 2: $4k \geq (\log n)^{1/7}$. In this case, we have $\log^k n \leq 4^{7k} k^{6k} \leq \mathcal{O}(k^{10k})$, so now we have $L(n, k) \leq \mathcal{O}(n^\alpha)$ and the bound from [Corollary 66](#) again suffices.

Case 3: $\alpha \leq k - 3$ and $4k \leq (\log n)^{1/7}$. In this case, observe that $n \geq 2^{4^7 k^7}$ holds. Let $t = \lfloor n/k \rfloor$. By $k \geq 2$, we have $t \geq 2^{400k^6} = t_0$. Let $G_1, G_2 \sim \mathcal{G}_1, \mathcal{G}_2$ be k -partite k -hypergraphs with t vertices in each vertex class as in [Theorem 67](#). By adding isolated vertices if necessary, we can use \mathcal{A} to approximately count edges in G_1 and G_2 in expected cost at most $\mathcal{C}(n, k)$.

Recall from [Section 2.1.2](#) that we regard \mathcal{A} as a discrete probability distribution over a set $\text{supp}(\mathcal{A})$ of deterministic algorithms based on random choices of \mathcal{A} , and let $A \sim \mathcal{A}$. By hypothesis, with probability at least $(9/10)^2 \geq 4/5$, A will correctly output a $(1/2)$ -approximation for both graphs G_1 and G_2 . Moreover, by [Theorem 67\(i\)](#), with probability at least $19/20$, we have $e(G_2) \geq 4e(G_1)$. If both of these events occur, then we cannot have $A(\text{cIND}(G_1)) = A(\text{cIND}(G_2))$; thus by a union bound, we have

$$\mathbb{P}_{\substack{A \sim \mathcal{A} \\ (G_1, G_2) \sim (\mathcal{G}_1, \mathcal{G}_2)}} \left(A(\text{cIND}(G_1)) \neq A(\text{cIND}(G_2)) \right) \geq 3/4. \quad (4.3.2)$$

Let F be the family of cIND -oracle algorithms $A \in \text{supp}(\mathcal{A})$ that satisfy [\(4.3.1\)](#). By [\(4.3.2\)](#) and conditioning on the event $A \in F$, we have

$$\begin{aligned} \frac{3}{4} &\leq \mathbb{P}_{\substack{A \sim \mathcal{A} \\ (G_1, G_2) \sim (\mathcal{G}_1, \mathcal{G}_2)}} \left(A(\text{cIND}(G_1)) \neq A(\text{cIND}(G_2)) \right) \\ &\leq 1 \cdot \mathbb{P}_{A \sim \mathcal{A}}(A \in F) + \frac{2}{3} \cdot \mathbb{P}_{A \sim \mathcal{A}}(A \notin F) = \frac{1}{3} \cdot \mathbb{P}_{A \sim \mathcal{A}}(A \in F) + \frac{2}{3}, \end{aligned}$$

and so $\mathbb{P}_{A \sim \mathcal{A}}(A \in F) \geq 1/4$. Thus by minimax (that is, [Theorem 17](#) with $p = 1/4$ and $\mathcal{D} = \mathcal{G}_1$), we can lower-bound the worst-case expected oracle cost of \mathcal{A} by the expected oracle cost of any deterministic $A \in F$ on a random input $G_1 \sim \mathcal{G}_1$:

$$\mathcal{C}(n, k) \geq \frac{1}{4} \inf_{A \in F} \mathbb{E}_{G_1 \sim \mathcal{G}_1} [\text{cost}(A, G_1)].$$

Observe that any such A can be turned into a cIND -algorithm which uses only queries of the form (S_1, \dots, S_k) with $S_i \subseteq V_i$ by splitting each query into at most k^k sub-queries of equal or lesser size; thus by [Theorem 67\(ii\)](#), we get

$$\mathcal{C}(n, k) \geq \frac{t^\alpha}{2^{5k+10} k^{8k}} \cdot \left(\frac{\log t}{\log \log t} \right)^{k - \lfloor \alpha \rfloor - 3}.$$

Since $2 \leq k \leq (\log n)^{1/7}$, we have $n/(2k) \leq t \leq n/k$, and so

$$\mathcal{C}(n, k) \geq \frac{n^\alpha}{2^{5k+10} k^{9k}} \left(\frac{\log t}{\log \log t} \right)^{k - \lfloor \alpha \rfloor - 3} = \frac{k^k}{2^{5k+10}} L(n, k).$$

We have $2^{5k+10} = O(k^k)$, so we have $\mathcal{C}(n, k) = \Omega(L(n, k))$ in all cases and the result follows. \blacktriangleleft

4.3.1 Defining the input graphs

Our first step in the proof of [Theorem 67](#) shall be to define the random graphs G_1 and G_2 and prove that $e(G_2) \geq 4e(G_1)$ holds with probability at least $19/20$. As in the uncoloured lower bound (see [Section 3.3](#)), we will form G_2 by adding a random set of “difficult-to-detect” edges to G_1 , writing $G_1 = H_1$ and $G_2 = H_1 \cup H_2$ for two independent random graphs H_1 and H_2 . Also as in [Section 3.3](#), we will take H_1 to be an Erdős-Rényi graph. Our choice of H_2

will be heavily motivated by the following important bottleneck in our colourful approximate counting algorithm `Count`.

Recall from [Section 4.1](#) that the main subroutine of `Count` is the coarse approximate counting algorithm `ColourCoarse_New` for k -partite k -hypergraphs. The key bottleneck in `ColourCoarse_New` arises when the input graph G has an (I, ζ) -core for some $|I| = \lfloor \alpha_k \rfloor + 1$; for technical convenience, we will instead take $|I| = \lfloor \alpha_k \rfloor + 2$. Writing V_1, \dots, V_k for the vertex classes of G , suppose for simplicity that $I = \{k - \lfloor \alpha_k \rfloor - 1, \dots, k\}$, and that for all $i \in I$ there exists $r_i \in V_i$ such that every edge in G contains r_i . In this case, the $\log^{\Theta(k - \alpha_k)} n$ oracle cost arises in `CoarseLargeCore` (in [Section 4.1.3](#)) because we must “guess” a probability vector (q_1, \dots, q_k) such that $q_1 \cdots q_k \approx 1/e(G)$, and such that deleting vertices from each V_i with probability $1 - q_i$ is still likely to leave some edges in G . We know we should choose $q_i \approx 1$ for $i \geq k - \lfloor \alpha_k \rfloor - 1$ in order to avoid deleting the high-degree “root vertices” in the colour classes $V_{k - \lfloor \alpha_k \rfloor - 1}, \dots, V_k$, but we don’t know the degree distribution of vertices in $V_1, \dots, V_{k - \lfloor \alpha_k \rfloor - 2}$. Since `CoarseLargeCore` only needs to return a coarse approximation, roughly speaking the algorithm tries each tuple of q_i ’s in $\{1, 1/2, 1/4, \dots, 1/n\}^{k - \lfloor \alpha_k \rfloor - 2}$ for a total of $\log^{\Theta(k - \alpha_k)} n$ possible probability vectors.

Motivated by this bottleneck, we will define H_2 to be a randomly-chosen **complete** k -partite graph. We define its parts in the “non-rooted classes” $V_1, \dots, V_{k - \lfloor \alpha_k \rfloor - 2}$ by binomially removing vertices according to a random probability vector $\vec{Q} := (Q_1, \dots, Q_{k - \lfloor \alpha_k \rfloor - 2})$, and we define its parts in the “rooted classes” $V_{k - \lfloor \alpha_k \rfloor - 1}, \dots, V_k$ to be uniformly random vertices $\mathcal{R}_{k - \lfloor \alpha_k \rfloor - 1}, \dots, \mathcal{R}_k$. This construction has the property that a random query of the form used in `CoarseLargeCore` is likely to distinguish H_1 from $H_1 \cup H_2$ only if it deletes vertices in the non-rooted classes with probabilities close to the “correct” probability vector \vec{Q} . More formally, we take the following notation as standard throughout the remainder of [Section 4.3](#).

► **Definition 69.** *Let t and k be integers with $k \geq 2$ and $t \geq t_0$, and let $\alpha \in [0, k - 3]$. Everything in this definition will formally depend on the values of t, k and α ; these values will always be clear from context, so we omit this dependence from the notation. Let*

$$\begin{aligned} p &:= 1/t^{(k + \lfloor \alpha \rfloor + 2)/2}, & x &:= pt^{\lfloor \alpha \rfloor + 2}, \\ \beta &:= \lfloor (\log t) / (20k^4 \log \log t) \rfloor, & B &:= \lfloor \log_{x^{1/\beta}}((24 \log t)/t) \rfloor. \end{aligned} \quad (4.3.3)$$

For all $i \in [k]$, we write $V_i = \{(i, j) : j \in [t]\}$. We define $H_1 := H_1(t, k, \alpha)$ to be a k -partite Erdős-Rényi k -hypergraph with vertex classes V_1, \dots, V_k , where each edge with one vertex in each V_i is present independently with probability p .

We say $V_1, \dots, V_{k - \lfloor \alpha \rfloor - 2}$ are non-rooted classes, and $V_{k - \lfloor \alpha \rfloor - 1}, \dots, V_k$ are rooted classes. For each rooted class V_i , let \mathcal{X}_i be a singleton subset of V_i containing a single uniformly random vertex $\mathcal{R}_i \in V_i$; we call \mathcal{R}_i the root of V_i . Let $\vec{Q} := (Q_1, \dots, Q_{k - \lfloor \alpha \rfloor - 2})$ be chosen uniformly at random from the set

$$\left\{ (q_1, \dots, q_{k - \lfloor \alpha \rfloor - 2}) : q_i \in \{(2^5 x)^{j/\beta} : j \in [0, B] \cap \mathbb{N}\} \text{ and } \prod_{i=1}^{k - \lfloor \alpha \rfloor - 2} q_i = 2^{5(k - \lfloor \alpha \rfloor - 2)} x \right\}.$$

For each non-rooted class V_i , let $\mathcal{X}_i \subseteq V_i$ be sampled at random by including each vertex of V_i independently with probability Q_i . We then define $H_2 := H_2(t, k, \alpha)$ to be the complete k -partite k -uniform hypergraph with vertex classes $\mathcal{X}_1, \dots, \mathcal{X}_k$. Finally, we define $G_1 \sim \mathcal{G}_1$ and $G_2 \sim \mathcal{G}_2$ by

$$G_1 := G_1(t, k, \alpha) = H_1, \quad G_2 := G_2(t, k, \alpha) = H_1 \cup H_2.$$

We first observe that these graphs G_1 and G_2 have tk vertices. By $\alpha \leq k-3$ we always have $x = t^{(\lfloor \alpha \rfloor + 2 - k)/2} \leq t^{-1/2} < 1$. Moreover, by $t \geq t_0$, the fact that $t \mapsto \beta(t, k)$ is non-decreasing, and $k \geq 2$, we always have

$$\beta \geq \lfloor 400k^6 / (20k^4 \log(400k^6)) \rfloor = \lfloor 20k^2 / \log(400k^6) \rfloor \geq 2k.$$

Finally, by $x = t^{(\lfloor \alpha \rfloor + 2 - k)/2}$, we have

$$\begin{aligned} B &\geq \frac{\beta \log((24 \log t)/t)}{\log x} - 1 = \frac{2\beta \log(t/(24 \log t))}{(k - \lfloor \alpha \rfloor - 2) \log t} - 1 \\ &\geq \frac{3\beta \log t}{2(k - \lfloor \alpha \rfloor - 2) \log t} - 1 \geq \frac{\beta}{k - \lfloor \alpha \rfloor - 2}. \end{aligned}$$

The second inequality follows from $\log(24 \log t) \leq \frac{1}{4} \log t$ for $t \geq t_0$ and $k \geq 2$, and the third inequality follows from $\beta/(k - \lfloor \alpha \rfloor - 2) \geq 2k/(k - \lfloor \alpha \rfloor - 2) \geq 2$. Since $B \geq \beta/(k - \lfloor \alpha \rfloor - 2)$ holds, there is a choice of $k - \lfloor \alpha \rfloor - 2$ integer values of $j \in [0, B]$ which sum to β ; hence there is at least one valid choice $\vec{q} \in \text{supp}(\vec{\mathcal{Q}})$. We also make some remarks to motivate the definitions of (4.3.3).

► **Remark 70.** In Definition 69:

(i) The value of x is chosen, so that the following holds for all $\vec{q} \in \text{supp}(\vec{\mathcal{Q}})$:

$$\begin{aligned} \mathbb{E}_{H_2}(e(H_2) \mid \vec{\mathcal{Q}} = \vec{q}) &= \prod_{i=1}^{k - \lfloor \alpha \rfloor - 2} \mathbb{E}_{H_2}(|\mathcal{X}_i| \mid \vec{\mathcal{Q}} = \vec{q}) = \prod_{i=1}^{k - \lfloor \alpha \rfloor - 2} (q_i t) \\ &= \left(2^{5(k - \lfloor \alpha \rfloor - 2)} x\right) \cdot t^{k - \lfloor \alpha \rfloor - 2} = 2^{5(k - \lfloor \alpha \rfloor - 2)} p t^k \\ &= 2^{5(k - \lfloor \alpha \rfloor - 2)} \mathbb{E}_{H_1}(e(H_1)). \end{aligned}$$

(ii) The value of B is chosen to be the largest integer with $x^{B/\beta} \geq (24 \log t)/t$; this ensures that, for all possible values $\vec{q} \in \text{supp}(\vec{\mathcal{Q}})$ and all $i \in [k - \lfloor \alpha \rfloor - 2]$, we have

$$\mathbb{E}_{H_2}(|\mathcal{X}_i| \mid \vec{\mathcal{Q}} = \vec{q}) = q_i t \geq 24 \log t,$$

which will allow us to show concentration of $|\mathcal{X}_i|$ conditioned on $\vec{\mathcal{Q}}$.

(iii) The values of p and β are chosen to ensure that there are $(\log t)^{\Omega(k - \alpha)}$ possible values of $\vec{\mathcal{Q}}$; we will discuss this in more detail when it becomes relevant.

The following lemma establishes Theorem 67(i) for \mathcal{G}_1 and \mathcal{G}_2 .

► **Lemma 71.** *We have*

$$\mathbb{P}_{(G_1, G_2) \sim (\mathcal{G}_1, \mathcal{G}_2)} \left(e(G_2) \geq 4e(G_1) \right) \geq 19/20.$$

Proof. It suffices to demonstrate that, with probability at least 19/20, the following events occur simultaneously:

- \mathcal{E}_1 , the event that $e(H_1) \leq 2pt^k$ holds; and
- \mathcal{E}_2 , the event that $e(H_2) \geq 8pt^k$ holds.

Indeed, if these two events occur, then $e(G_2) \geq e(H_2) \geq 4e(H_1) = 4e(G_1)$ as required.

We first show that \mathcal{E}_1 is likely. We have $\mathbb{E}(e(H_1)) = pt^k$, so by the Chernoff bound of Lemma 13 applied with $\delta = 1$ we have

$$\mathbb{P}(\mathcal{E}_1) = 1 - \mathbb{P}(e(H_1) > 2pt^k) \geq 1 - 2e^{-pt^k/3}.$$

Since $\alpha \leq k - 3$, we have $p \geq 1/t^{k-1/2}$ and so since $t \geq 400$ we have

$$\mathbb{P}(\mathcal{E}_1) \geq 1 - 2e^{-t^{1/2}/3} \geq 1 - 2e^{-20/3} > 99/100. \quad (4.3.4)$$

We now show that \mathcal{E}_2 is likely. Let $\vec{q} = (q_1, \dots, q_{k-\lfloor \alpha \rfloor - 2}) \in \text{supp}(\vec{\mathcal{Q}})$ be any possible value of $\vec{\mathcal{Q}}$. Then for all $j \in [k - \lfloor \alpha \rfloor - 2]$, we have $\mathbb{E}(|\mathcal{X}_j| \mid \vec{\mathcal{Q}} = \vec{q}) = tq_j$, so by [Lemma 13](#) applied with $\delta = 1/2$ we have

$$\mathbb{P}\left(|\mathcal{X}_j| < tq_j/2 \mid \vec{\mathcal{Q}} = \vec{q}\right) \leq 2e^{-tq_j/12}.$$

By [Remark 70ii](#) about our choice of B , we have $tq_j \geq tx^{B/\beta} \geq 24 \log t$; thus

$$\mathbb{P}\left(|\mathcal{X}_j| < tq_j/2 \mid \vec{\mathcal{Q}} = \vec{q}\right) \leq 2e^{-2 \log t} = 2/t^2.$$

By a union bound over j , it follows that

$$\mathbb{P}\left(\text{for all } j \in [k - \lfloor \alpha \rfloor - 2], \text{ we have } |\mathcal{X}_j| \geq tq_j/2 \mid \vec{\mathcal{Q}} = \vec{q}\right) \geq 1 - 2k/t^2.$$

If this event occurs, then since $\alpha \leq k - 3$, we have

$$e(H_2) = \prod_{i=1}^{k-\lfloor \alpha \rfloor - 2} |\mathcal{X}_i| \geq (t/2)^{k-\lfloor \alpha \rfloor - 2} \cdot \prod_{i=1}^{k-\lfloor \alpha \rfloor - 2} q_i = t^{k-\lfloor \alpha \rfloor - 2} \cdot \frac{2^{5(k-\lfloor \alpha \rfloor - 2)}}{2^{k-\lfloor \alpha \rfloor - 2}} \cdot x \geq 8pt^k,$$

and so \mathcal{E}_2 also occurs. Since $t \geq t_0$, it follows that

$$\mathbb{P}(\mathcal{E}_2 \mid \vec{\mathcal{Q}} = \vec{q}) \geq 1 - 2k/t^2 \geq 99/100.$$

Since \vec{q} was arbitrary, by a union bound with (4.3.4) we arrive at $\mathbb{P}(\mathcal{E}_1 \cap \mathcal{E}_2) \geq 19/20$, and the result follows as described above. \blacktriangleleft

4.3.2 A framework to distinguish \mathcal{G}_1 from \mathcal{G}_2

Throughout the rest of this section, let t, k, α, t_0, A and C be as in the statement of [Theorem 67\(ii\)](#), and let $\mathcal{G}_1, \mathcal{G}_2, G_1, G_2, H_1, H_2, p, x, \beta, B, V_1, \dots, V_k, \mathcal{X}_1, \dots, \mathcal{X}_k, \mathcal{R}_{k-\lfloor \alpha \rfloor - 1}, \dots, \mathcal{R}_k$ and $\vec{\mathcal{Q}}$ be as in [Definition 69](#). As in [Remark 44](#) in the uncoloured case, we first observe that we may assume without loss of generality that the number of queries made by A on G_1 is deterministic.

► **Remark 72.** Without loss of generality, we may assume that there is an $N = N_{n,k} \in \mathbb{N}$ such that $A(\text{cIND}(G))$ makes exactly the same number N of queries for each n -vertex k -hypergraph G . Moreover, we may assume that for some $N'(G) \leq N$, the first $N'(G)$ queries have non-zero cost and the last $N - N'(G)$ queries have zero cost.

Proof. As in the proof of [Remark 44](#), the result follows by first skipping zero-cost queries and then “padding” the end of A ’s execution with zero-cost queries to the empty set. \blacktriangleleft

We now reintroduce some notation from [Section 3.3](#) to describe the sequence of oracle queries executed by A on a given input graph; this is a precise analogue of [Definition 45](#) in the colourful setting.

► **Definition 73.** Let $n := kt$ and recall that V_1, \dots, V_k are disjoint sets of size t each. Let G be an arbitrary n -vertex k -hypergraph on the vertex classes V_1, \dots, V_k . Let $S_1(G), \dots, S_N(G)$ be the sequence of oracle queries that A makes when given input $\text{cIND}(G)$. For all $i \in [N]$, write $S_i(G) =: (S_{i,1}(G), \dots, S_{i,k}(G))$.

In the following, we outline the key idea of the proof of [Theorem 67\(ii\)](#). We first recall from the assumption on A in [Theorem 67\(ii\)](#) that $S_{i,j}(G) \subseteq V_i$ holds for all $i \in [N]$ and $j \in [k]$. As in the uncoloured case, in order for a query S to distinguish between G_1 and G_2 , it must contain at least one edge of H_2 but no edges of H_1 ; this means it must contain all roots $\mathcal{R}_j \in \mathcal{X}_j$ of each rooted class V_j of H_2 and at least one vertex from \mathcal{X}_j for each non-rooted class V_j of H_2 . Roughly speaking, we will show that, with high probability, in order for a query $S = (S_1, \dots, S_k)$ to be useful, it must satisfy the following criteria:

- (C1) There cannot be too many “unexposed” possible edges of H_1 in $S^{(k)}$, since otherwise $H_1[S]$ is likely to contain an edge and the query S will not distinguish G_1 from G_2 . (Recall from [Section 2.1.1](#) that we write $S^{(k)} = \{\{s_1, \dots, s_k\} : s_j \in S_j \text{ for all } j \in [k]\}$ and $H_1[S] = H_1[S_1, \dots, S_k]$.)
- (C2) For each rooted class V_j , the set $S_j \subseteq V_j$ must be very large, since otherwise the query S is likely to miss the root \mathcal{R}_j of H_2 , in which case $H_2[S]$ will contain no edges and the query S will not distinguish G_1 from G_2 . In particular, we will see that this criterion requires $\text{cost}(S)$ to be at least roughly n^α . (This is similar to [Corollary 66](#).)
- (C3) For each non-rooted class V_j , the set $S_j \subseteq V_j$ must contain at least roughly $1/\mathcal{Q}_j$ vertices, since otherwise S_j will contain no vertices of \mathcal{X}_j , in which case $H_2[S]$ will contain no edges and the query S will again not distinguish G_1 from G_2 .

By combining these three properties, we will be able to show that, with high probability, any query distinguishing G_1 and G_2 must be “accurately profiled” in the sense that, for all non-rooted classes V_j , we have $|S_j| \approx 1/\mathcal{Q}_j$. Intuitively, this shows that A has to guess the value of $\vec{\mathcal{Q}}$; there are $\log^{\Theta(k-\lfloor\alpha\rfloor)} t$ possible values of $\vec{\mathcal{Q}}$, and property (C2) says that each useful query has cost at least roughly n^α , so this leads naturally to the overall lower bound on the oracle cost that we are claiming. Thus in a sense, the proof is based around showing that the bottleneck of `CoarseLargeCore` described in [Section 4.3.1](#) (in which the algorithm essentially does guess an analogue of $\vec{\mathcal{Q}}$) is necessary.

Turning these ideas into a rigorous argument is difficult, particularly since we work in an adaptive setting where queries may depend on answers received in past queries, and so we must be very careful with conditioning. We will first define formal terminology and events corresponding to (C1)–(C3), as well as the idea of “accuracy”. We will then prove (in [Lemma 81](#)) that if these events all occur and all queries are inaccurate, then A fails to distinguish G_1 from G_2 . Finally, in [Section 4.3.3](#) we will show that these events are all likely to occur, and apply a union bound to prove the result.

We first formalise the idea of “accuracy”.

- **Definition 74.** Let $S_i = (S_{i,1}, \dots, S_{i,k})$ be a query.
 - We say S_i is *accurately rooted* if $\mathcal{R}_j \in S_{i,j}$ holds for all j with $k - \lfloor\alpha\rfloor - 1 \leq j \leq k$, and otherwise we say S_i is *inaccurately rooted*.
 - We say S_i is *accurately profiled* if, for all j with $1 \leq j \leq k - \lfloor\alpha\rfloor - 2$, we have $|S_{i,j}| \in (x^{1/2\beta}/\mathcal{Q}_j, x^{-1/2\beta}/\mathcal{Q}_j)$, and otherwise we say S_i is *inaccurately profiled*.
 - We say S_i is *accurate* if it is both *accurately rooted* and *accurately profiled*, and otherwise we say S_i is *inaccurate*.
 - We define $\mathcal{E}_{\text{inacc}}$ to be the event that all queries $S_1(G_1), \dots, S_N(G_1)$ are inaccurate.

We next formalise the notion of an “unexposed” edge from (C1), in a similar fashion to the proof of [Lemma 47](#) in the uncoloured setting.

- **Definition 75.** For all $i \in [N]$, we define the set F_i of *unexposed edges* via

$$F_i := S_i(G_1)^{(k)} \setminus \bigcup_{\substack{\ell \leq i-1 \\ e(G_1[S_\ell])=0}} S_\ell(G_1)^{(k)}.$$

Note for intuition that, before the i^{th} query, the algorithm has already performed the queries $S_\ell(G_1)$ for $\ell \leq i-1$. Each time it received $e(G_1[S_\ell(G_1)]) = 0$ as a response from the query $\text{cIND}(G_1)_{S_\ell(G_1)}$, the algorithm can know for certain that none of the edges in $S_\ell(G_1)^{(k)}$ exist. Thus, F_i is the set of all possible, unexposed edges of G_1 that the i^{th} query can hope to newly detect.

We next define three events which, if they occur, formalise (C1)–(C3) respectively.

► **Definition 76** (Formalisation of (C1)). Let $\mathcal{E}_{\text{edge}}$ be the event that, for all $i \in [N]$ with $|F_i| \geq (\log^{k^3} t)/(2p)$, we have $e(G_1[S_i(G_1)]) > 0$.

► **Definition 77** (Formalisation of (C2)). We say a query $S = (S_1, \dots, S_k)$ is safely rooted if $\min\{|S_j| : k - \lfloor \alpha \rfloor - 1 \leq j \leq k\} > t/\log^{2k} t$.

Let $\mathcal{E}_{\text{root}}$ be the event that, for all $i \in [N]$, the query $S_i(G_1)$ is safely rooted or inaccurately rooted.

► **Definition 78** (Formalisation of (C3)). Let $\xi := x^{-1/(2k\beta)}$. Let $\mathcal{E}_{\text{nonroot}}$ be the event that, for all $i \in [N]$ such that $S_i(G_1)$ is safely rooted and such that there exists $j \in [k - \lfloor \alpha \rfloor - 2]$ with $|S_{i,j}(G_1)| \leq 1/(\xi Q_j)$, we have $e(H_2[S_i(G_1)]) = 0$.

Note that $\xi > 1$ holds, since $x < 1$ and $\beta \geq 1$. Finally, recall from the assumption on A in Theorem 67(ii) that $A(\text{cIND}(G_1))$ has expected oracle cost at most $C/2$. We now define an event for the cost of $A(\text{cIND}(G_1))$ not being too much larger than this expectation.

► **Definition 79.** Let $\mathcal{E}_{\text{cost}}$ be the event that $\text{cost}(A, G_1) \leq C$ holds.

Our next goal is to prove Lemma 81, which says that, if $\mathcal{E}_{\text{inacc}}$, $\mathcal{E}_{\text{edge}}$, $\mathcal{E}_{\text{root}}$, $\mathcal{E}_{\text{nonroot}}$, and $\mathcal{E}_{\text{cost}}$ all occur, then A fails to distinguish G_1 from G_2 , as required by Theorem 67(ii). To this end we first prove an ancillary lemma, which says that any sequence of individually-cheap queries cannot “cover” many possible tuples of root locations in $V_{k-\lfloor \alpha \rfloor-1}, \dots, V_k$.

► **Lemma 80.** Let $\gamma \in (0, 1]$, and let S_1, \dots, S_z be a sequence of queries of total cost at most C . Further suppose $|S_{i,k-\lfloor \alpha \rfloor-1}| + \dots + |S_{i,k}| \leq \gamma kt$ for all $i \in [z]$. Then

$$\sum_{i \in [z]} |S_{i,k-\lfloor \alpha \rfloor-1} \times \dots \times S_{i,k}| \leq C(\gamma kt)^{\lfloor \alpha \rfloor+2-\alpha}.$$

Proof. Note first that, by the AM-GM inequality, for all $i \in [z]$ we have

$$\prod_{j=k-\lfloor \alpha \rfloor-1}^k |S_{i,j}| \leq \left(\frac{\sum_{j=k-\lfloor \alpha \rfloor-1}^k |S_{i,j}|}{\lfloor \alpha \rfloor + 2} \right)^{\lfloor \alpha \rfloor+2} \leq \left(\sum_{j=k-\lfloor \alpha \rfloor-1}^k |S_{i,j}| \right)^{\lfloor \alpha \rfloor+2}.$$

Setting $s_i := |S_{i,k-\lfloor \alpha \rfloor-1}| + \dots + |S_{i,k}|$, this gives

$$\sum_{i \in [z]} |S_{i,k-\lfloor \alpha \rfloor-1} \times \dots \times S_{i,k}| \leq \sum_{i \in [z]} s_i^{\lfloor \alpha \rfloor+2}. \quad (4.3.5)$$

By assumption, we have $\sum_{i \in [z]} s_i^\alpha \leq \sum_{i \in [z]} \text{cost}(S_i) \leq C$ and $s_i \in [0, \gamma kt]$ for all $i \in [z]$. Moreover, $\lfloor \alpha \rfloor + 2 > \alpha$. We now apply Karamata’s inequality in the form of Corollary 22, taking $W = C$, $c = \gamma kt$, $t = z$ and $r = \lfloor \alpha \rfloor + 2$. This yields:

$$\sum_{i \in [z]} s_i^{\lfloor \alpha \rfloor+2} \leq C(\gamma kt)^{\lfloor \alpha \rfloor+2-\alpha},$$

and so the result follows from (4.3.5). ◀

We now prove the main lemma of this subsection: If the five key events defined above occur simultaneously, then the algorithm A is unable to distinguish the two possible input graphs G_1 and G_2 from each other.

► **Lemma 81.** *If $\mathcal{E}_{\text{inacc}}$, $\mathcal{E}_{\text{edge}}$, $\mathcal{E}_{\text{root}}$, $\mathcal{E}_{\text{nonroot}}$, and $\mathcal{E}_{\text{cost}}$ all occur, then we have $A(\text{cIND}(G_1)) = A(\text{cIND}(G_2))$.*

Proof. Throughout, we assume that $\mathcal{E}_{\text{inacc}}$, $\mathcal{E}_{\text{edge}}$, $\mathcal{E}_{\text{root}}$, $\mathcal{E}_{\text{nonroot}}$ and $\mathcal{E}_{\text{cost}}$ occur and work deterministically. For all $i \in [N]$, let S_i be the value of $S_i(G_1)$.

We will prove by induction that, for all $i \in [N]$, we have $\text{cIND}(G_1)_{S_i} = \text{cIND}(G_2)_{S_i}$; since A is deterministic, this immediately implies $A(\text{cIND}(G_1)) = A(\text{cIND}(G_2))$. Fix $i \in [N]$, and suppose that $\text{cIND}(G_1)_{S_\ell} = \text{cIND}(G_2)_{S_\ell}$ holds for all $\ell \in [i-1]$.

We first define some useful notation. Let

$$\begin{aligned}\mathcal{Y}_i &= \{S_\ell : \ell \in [i-1] \text{ and } e(G_1[S_\ell]) > 0\}, \\ \mathcal{N}_i &= \{S_\ell : \ell \in [i-1] \text{ and } e(G_2[S_\ell]) = 0\}.\end{aligned}$$

Recall that G_1 is a subgraph of G_2 ; thus \mathcal{Y}_i is the set of all queries in $\{S_1, \dots, S_{i-1}\}$ which return that an edge exists, regardless of whether the input graph is G_1 or G_2 ; and \mathcal{N}_i is the set of all queries which return that no edge exists, regardless of whether the input graph is G_1 or G_2 . By our induction hypothesis, we have $[i-1] = \mathcal{Y}_i \cup \mathcal{N}_i$.

Since $\mathcal{E}_{\text{inacc}}$ occurs by assumption, S_i is inaccurate. We now consider three cases depending on how S_i fails to be accurate: by definition, either it is inaccurately rooted (Case 1), or it is inaccurately profiled and there is at least one vertex class whose intersection with the query is small (Case 2), or it is inaccurately profiled and the intersection of every vertex class with the query is reasonably large (Case 3).

Case 1: S_i is inaccurately rooted. In this case, by definition of H_2 , we have $e(H_2[S_i]) = 0$. Since $G_2 = G_1 \cup H_2$, it follows that $e(G_1[S_i]) = e(G_2[S_i])$, and we have $\text{cIND}(G_1)_{S_i} = \text{cIND}(G_2)_{S_i}$ as required.

Case 2: S_i is accurately rooted and there exists a non-rooted class V_j such that $|S_{i,j}| \leq 1/(\xi Q_j)$. In this case, since $\mathcal{E}_{\text{nonroot}}$ occurs, we again have $e(H_2[S_i]) = 0$ and thus $\text{cIND}(G_2)_{S_i} = \text{cIND}(G_1)_{S_i}$ as required.

Case 3: S_i is accurately rooted and, for all non-rooted classes V_j , we have $|S_{i,j}| \geq 1/(\xi Q_j)$. This is the difficult case of the proof. Our aim is to prove $F_i \geq (\log^{k^3} t)/(2p)$, which by $\mathcal{E}_{\text{edge}}$ implies $e(G_1[S_i]) > 0$; since G_1 is a subgraph of G_2 , this immediately implies $\text{cIND}(G_1)_{S_i} = \text{cIND}(G_2)_{S_i} = 0$.

We divide the queries in \mathcal{N}_i into two sets according to their size; let

$$\begin{aligned}\mathcal{N}_i^+ &= \left\{S_m \in \mathcal{N}_i : \max \{|S_{m,j}| : j \in [k]\} \geq t/(\log t)^{4k^2/(\lfloor \alpha \rfloor + 2 - \alpha)}\right\}, \\ \mathcal{N}_i^- &= \mathcal{N}_i \setminus \mathcal{N}_i^+.\end{aligned}$$

Observe that by the definition of F_i , we have

$$F_i = S_i^{(k)} \setminus \bigcup_{S_m \in \mathcal{N}_i} S_m^{(k)} = S_i^{(k)} \setminus \left(S_i^{(k)} \cap \bigcup_{S_m \in \mathcal{N}_i} \left(S_m^{(k)} \setminus \bigcup_{S_r \in \mathcal{N}_m} S_r^{(k)} \right) \right),$$

and thus by $\mathcal{N}_i = \mathcal{N}_i^- \cup \mathcal{N}_i^+$, we have

$$\begin{aligned} |F_i| &\geq |S_i^{(k)}| - \left| S_i^{(k)} \cap \bigcup_{S_m \in \mathcal{N}_i^-} S_m^{(k)} \right| - \left| \bigcup_{S_m \in \mathcal{N}_i^+} \left(S_m^{(k)} \setminus \bigcup_{S_r \in \mathcal{N}_m} S_r^{(k)} \right) \right| \\ &\geq |S_i^{(k)}| - \left| S_i^{(k)} \cap \bigcup_{S_m \in \mathcal{N}_i^-} S_m^{(k)} \right| - \sum_{S_m \in \mathcal{N}_i^+} \left| S_m^{(k)} \setminus \bigcup_{S_r \in \mathcal{N}_m} S_r^{(k)} \right|. \end{aligned} \quad (4.3.6)$$

We will show that the second and the third term each are at most $\frac{1}{4}|S_i^{(k)}|$. We will bound each term of (4.3.6) in turn.

First term of (4.3.6): We bound $|S_i^{(k)}|$ below. Since $\mathcal{E}_{\text{inacc}}$ occurs, S_i is an inaccurate query; as we are assuming in Case 3 that S_i is accurately rooted, it must be inaccurately profiled, so that there exists $\ell \in [k - \lfloor \alpha \rfloor - 2]$ with $|S_{i,\ell}| \notin (x^{1/(2\beta)}/\mathcal{Q}_\ell, x^{-1/(2\beta)}/\mathcal{Q}_\ell)$. Moreover, in Case 3 we have $|S_{i,\ell}| \geq 1/(\xi \mathcal{Q}_\ell)$. Note that $1/\xi = x^{1/(2k\beta)} > x^{1/(2\beta)}$ holds by $x < 1$, which implies $|S_{i,\ell}| > x^{1/(2\beta)}/\mathcal{Q}_\ell$; due to the inaccuracy of $|S_{i,\ell}|$, we must have $|S_{i,\ell}| \geq x^{-1/(2\beta)}/\mathcal{Q}_\ell$. Moreover, since $\mathcal{E}_{\text{root}}$ occurs and we are assuming that S_i is accurately rooted, S_i must also be safely rooted; thus $|S_{i,j}| \geq t/\log^{2k} t$ for all $j \geq k - \lfloor \alpha \rfloor - 1$. Putting these three bounds together, it follows that

$$\begin{aligned} |S_i^{(k)}| &= \prod_{j=1}^k |S_{i,j}| \geq |S_{i,\ell}| \cdot \prod_{\substack{j \leq k - \lfloor \alpha \rfloor - 2 \\ j \neq \ell}} |S_{i,j}| \cdot \prod_{j=k - \lfloor \alpha \rfloor - 1}^k |S_{i,j}| \\ &\geq \frac{x^{-1/(2\beta)}}{\mathcal{Q}_\ell} \cdot \prod_{\substack{j \leq k - \lfloor \alpha \rfloor - 2 \\ j \neq \ell}} \frac{1}{\xi \mathcal{Q}_j} \cdot \prod_{j=k - \lfloor \alpha \rfloor - 1}^k \frac{t}{\log^{2k} t} \\ &= \frac{x^{-1/(2\beta)}}{\xi^{k - \lfloor \alpha \rfloor - 3}} \cdot \frac{t^{\lfloor \alpha \rfloor + 2}}{\log^{2k(\lfloor \alpha \rfloor + 2)} n} \cdot \prod_{j=1}^{k - \lfloor \alpha \rfloor - 2} \frac{1}{\mathcal{Q}_j}. \end{aligned}$$

Thus by $\prod_j \mathcal{Q}_j = 2^{5(k - \lfloor \alpha \rfloor - 2)} x \leq 2^{5k} p t^{\lfloor \alpha \rfloor + 2}$ and $\xi = x^{-1/(2k\beta)} > 1$ and $\lfloor \alpha \rfloor + 2 \leq k$, we have

$$|S_i^{(k)}| \geq \frac{x^{-1/(2\beta)}}{\xi^{k-2}} \cdot \frac{t^{\lfloor \alpha \rfloor + 2}}{\log^{2k^2} t} \cdot \frac{1}{2^{5k} p \cdot t^{\lfloor \alpha \rfloor + 2}} = \frac{x^{-1/(k\beta)}}{2^{5k} p \log^{2k^2} t}. \quad (4.3.7)$$

Recall that $x = p t^{\lfloor \alpha \rfloor + 2} = t^{(\lfloor \alpha \rfloor + 2 - k)/2}$. Since $\alpha \leq k - 3$ we have $k - \lfloor \alpha \rfloor - 2 \geq 1$, so $x^{-1} \geq t^{1/2}$. By the definition of β in Definition 69, it follows that

$$x^{-1/(k\beta)} \geq t^{1/(2k\beta)} \geq t^{(20k^4 \log \log t)/(2k \log t)} = (\log t)^{10k^3}.$$

Since $t \geq t_0$ we have $(\log t)^{k^3} \geq 2^{5k}$, and so by (4.3.7) we have

$$|S_i^{(k)}| \geq \frac{(\log t)^{7k^3}}{p}. \quad (4.3.8)$$

Second term of (4.3.6): We show that the second term is at most $\frac{1}{4}|S_i^{(k)}|$, by showing that almost all possible edges of G_1 covered by S_i intersect $V_{k - \lfloor \alpha \rfloor - 1}, \dots, V_k$ at vertices not covered by any query in \mathcal{N}_i^- .

Indeed, we will apply Lemma 80 with $\gamma = 1/(\log t)^{4k^2/(\lfloor \alpha \rfloor + 2 - \alpha)}$ to the queries in \mathcal{N}_i^- . Note that the conditions of the lemma are met, because, by $\mathcal{E}_{\text{cost}}$, the total cost of all queries

is at most C and, by definition of \mathcal{N}_i^- , the total size $\sum_{j=1}^k |S_{m,j}|$ of queries $S_m \in \mathcal{N}_i^-$ is at most γkt . Thus we get

$$\begin{aligned} \left| \bigcup_{S_m \in \mathcal{N}_i^-} (S_{m,k-\lfloor \alpha \rfloor - 1} \times \cdots \times S_{m,k}) \right| &\leq \sum_{S_m \in \mathcal{N}_i^-} |S_{m,k-\lfloor \alpha \rfloor - 1} \times \cdots \times S_{m,k}| \\ &\leq C \left(\frac{kt}{(\log t)^{4k^2/(\lfloor \alpha \rfloor + 2 - \alpha)}} \right)^{\lfloor \alpha \rfloor + 2 - \alpha}. \end{aligned}$$

By the definition of C in [Theorem 67](#), we get

$$\left| \bigcup_{S_m \in \mathcal{N}_i^-} (S_{m,k-\lfloor \alpha \rfloor - 1} \times \cdots \times S_{m,k}) \right| \leq C \cdot \frac{t^{\lfloor \alpha \rfloor + 2 - \alpha} k^2}{\log^{4k^2} t} \leq \frac{t^{\lfloor \alpha \rfloor + 2} \log^k t}{\log^{4k^2} t} \leq \frac{t^{\lfloor \alpha \rfloor + 2}}{\log^{3k^2} t}. \quad (4.3.9)$$

Next, we observe that since $\mathcal{E}_{\text{root}}$ occurs and S_i is accurately rooted by hypothesis of Case 3, we have $|S_{i,j}| \geq t/\log^{2k} t$ for all $j \geq k - \lfloor \alpha \rfloor - 1$; thus by (4.3.9) we have

$$\begin{aligned} |S_{i,k-\lfloor \alpha \rfloor - 1} \times \cdots \times S_{i,k}| &= \prod_{j=k-\lfloor \alpha \rfloor - 1}^k |S_{i,j}| \geq \frac{t^{\lfloor \alpha \rfloor + 2}}{\log^{2k^2} t} \\ &\geq (\log^{k^2} t) \cdot \left| \bigcup_{S_m \in \mathcal{N}_i^-} (S_{m,k-\lfloor \alpha \rfloor - 1} \times \cdots \times S_{m,k}) \right|. \end{aligned}$$

This implies that almost all k -tuples of $S_i^{(k)}$ intersect $V_{k-\lfloor \alpha \rfloor - 1}, \dots, V_k$ at vertices not covered by any query in \mathcal{N}_i^- ; we therefore have

$$\left| S_i^{(k)} \cap \bigcup_{S_m \in \mathcal{N}_i^-} S_m^{(k)} \right| \leq \frac{1}{4} |S_i^{(k)}|. \quad (4.3.10)$$

Third term of (4.3.6): We show that the third term is at most $\frac{1}{4} |S_i^{(k)}|$. Observe that by the definition of F_m , we conveniently have

$$\sum_{S_m \in \mathcal{N}_i^+} \left| S_m^{(k)} \setminus \bigcup_{S_r \in \mathcal{N}_m} S_r^{(k)} \right| = \sum_{S_m \in \mathcal{N}_i^+} |F_m|. \quad (4.3.11)$$

We proceed by proving an upper bound on $|\mathcal{N}_i^+|$ as well as on each term $|F_m|$. By the definition of \mathcal{N}_i^+ , for each query $S_m \in \mathcal{N}_i^+$, we have

$$\text{cost}(S_m) \geq (\max_j |S_{m,j}|)^\alpha \geq (t/(\log t)^{4k^2/(\lfloor \alpha \rfloor + 2 - \alpha)})^\alpha \geq (t/\log^{4k^2} t)^\alpha.$$

Since $\mathcal{E}_{\text{cost}}$ occurs, the total cost of all queries is at most C ; by the definition of C , it follows that

$$|\mathcal{N}_i^+| \leq \frac{C}{(t/\log^{4k^2} t)^\alpha} \leq \frac{t^\alpha \log^k t}{(t/\log^{4k^2} t)^\alpha} \leq \log^{5k^3} t.$$

Moreover, since $\mathcal{E}_{\text{edge}}$ occurs, we have $|F_m| \leq (\log^{k^3} t)/(2p)$ for all queries $S_m \in \mathcal{N}_i$. It therefore follows from (4.3.11) that

$$\sum_{S_m \in \mathcal{N}_i^+} \left| S_m^{(k)} \setminus \bigcup_{S_r \in \mathcal{N}_m} S_r^{(k)} \right| \leq (\log t)^{5k^3} \cdot \frac{\log^{k^3} t}{2p} \leq \frac{\log^{6k^3} t}{p}.$$

By (4.3.8), it follows that

$$\sum_{S_m \in \mathcal{N}_i^+} \left| S_m^{(k)} \setminus \bigcup_{S_r \in \mathcal{N}_m} S_r^{(k)} \right| \leq \frac{1}{\log^{k^3} n} |S_i^{(k)}| \leq \frac{1}{4} |S_i^{(k)}|. \quad (4.3.12)$$

Conclusion of the proof of Lemma 81. We are now essentially done. Combining (4.3.6), (4.3.10), and (4.3.12) yields $|F_i| \geq |S_i^{(k)}|/2$; applying (4.3.8) then yields

$$|F_i| \geq \frac{1}{2} |S_i^{(k)}| \geq \frac{\log^{7k^3} t}{2p}.$$

Since $\mathcal{E}_{\text{edge}}$ occurs, we have $e(H_1[S_i]) > 0$, and so $\text{cIND}(G_1)_{S_i} = \text{cIND}(G_2)_{S_i} = 0$ as required. \blacktriangleleft

4.3.3 Bounding probabilities

We are now going to **separately** bound the probability that each event $\mathcal{E}_{\text{edge}}$, $\mathcal{E}_{\text{root}}$, $\mathcal{E}_{\text{nonroot}}$ and $\mathcal{E}_{\text{inacc}}$ fails to occur in terms of the cost C . The event $\mathcal{E}_{\text{cost}}$ will be easy to handle using Markov's inequality. The final result then follows using Lemma 81 and the assumption that A *does* distinguish \mathcal{G}_1 from \mathcal{G}_2 with probability at least $2/3$; together, this shows that the cost must be high.

For every event except $\mathcal{E}_{\text{edge}}$, we will be able to work with H_1 (and hence S_1, S_2, \dots) exposed, using the fact that H_1 and H_2 are independent.

► **Definition 82.** *For the rest of the section and as in the proof of Lemma 81, we write $S_i := S_i(G_1)$ for all $i \in [N]$.*

► **Lemma 83.** *We have*

$$\mathbb{P}(\overline{\mathcal{E}_{\text{cost}}} \vee \mathcal{E}_{\text{edge}}) \geq 1 - Ce^{-(\log^{k^3} t)/2}.$$

Proof. For all i , let $\mathcal{E}_{\text{edge}, i}$ be the event that either $|F_i| \leq (\log^{k^3} t)/(2p)$ or $e(G_1[S_i]) > 0$. Let $\mathcal{E}_{\text{edge}, \leq i} = \mathcal{E}_{\text{edge}, 1} \wedge \dots \wedge \mathcal{E}_{\text{edge}, i}$. Recall from Remark 72 that all queries have non-zero cost — and hence cost at least 1 — except for a final segment of “padding” at the end of the algorithm. Thus if $\mathcal{E}_{\text{edge}, \leq \lfloor C \rfloor}$ occurs, then either $\mathcal{E}_{\text{edge}}$ occurs or the total query cost of $A(\text{cIND}(G_1))$ is greater than C ; we therefore have

$$\begin{aligned} \mathbb{P}(\overline{\mathcal{E}_{\text{cost}}} \vee \mathcal{E}_{\text{edge}}) &\geq \mathbb{P}(\mathcal{E}_{\text{edge}, \leq \lfloor C \rfloor}) = 1 - \sum_{i=1}^{\lfloor C \rfloor} \mathbb{P}\left(\overline{\mathcal{E}_{\text{edge}, i}} \wedge \bigwedge_{j=1}^{i-1} \mathcal{E}_{\text{edge}, j}\right) \\ &\geq 1 - \sum_{i=1}^{\lfloor C \rfloor} \mathbb{P}\left(\overline{\mathcal{E}_{\text{edge}, i}} \mid \bigwedge_{j=1}^{i-1} \mathcal{E}_{\text{edge}, j}\right). \end{aligned} \quad (4.3.13)$$

We will bound this sum term-by-term by exposing the results of past queries. Let $i \leq \lfloor C \rfloor$. Let $T_{< i}(G_1) = (S_1, \dots, S_i, \text{cIND}(G_1)_{S_1}, \dots, \text{cIND}(G_1)_{S_{i-1}})$ be the information to which A has access after its $(i-1)$ query to $\text{cIND}(G_1)$; thus $T_{< i}(G_1)$ is a deterministic function of G_1 . Let $t_{< i}(G_1) = (s_1, \dots, s_i, b_1, \dots, b_{i-1})$ be any possible value for $T_{< i}(G_1)$ consistent with the conditioning of (4.3.13), and let f_i be the value of F_i implied by conditioning on $T_{< i}(G_1) = t_{< i}(G_1)$. Let $\mathcal{Y}_i = \{s_j : b_j = 0\}$ and $\mathcal{N}_i = \{s_j : b_j = 1\}$; thus conditioned on

$T_{<i}(G_1) = t_{<i}(G_1)$, we have $s_j \in \mathcal{Y}_i$ if $e(G_1[s_j]) > 0$ and $s_j \in \mathcal{N}_i$ if $e(G_1[s_j]) = 0$. Then we have

$$\begin{aligned} & \mathbb{P}(\overline{\mathcal{E}_{\text{edge}, i}} \mid T_{<i}(G_1) = t_{<i}(G_1)) \\ &= \mathbb{P}(\overline{\mathcal{E}_{\text{edge}, i}} \mid e(G_1[s_j]) > 0 \text{ for all } s_j \in \mathcal{Y}_i \text{ and } e(G_1[s_j]) = 0 \text{ for all } s_j \in \mathcal{N}_i). \end{aligned}$$

If $|f_i| \leq (\log^{k^3} t)/(2p)$ then this probability is zero, so suppose $|f_i| > (\log^{k^3} t)/(2p)$. In this case $\overline{\mathcal{E}_{\text{edge}, i}}$ occurs if and only if $e(G_1[s_i]) = 0$. This event is a monotonically decreasing function of the indicator variables of G_1 's (independently-present) edges, and for all j , the events $e(G_1[s_j]) > 0$ are monotonically increasing functions of these variables. Thus by the FKG inequality (Lemma 15), we obtain

$$\begin{aligned} \mathbb{P}(\overline{\mathcal{E}_{\text{edge}, i}} \mid T_{<i}(G_1) = t_{<i}(G_1)) &\leq \mathbb{P}(e(G_1[s_i]) = 0 \mid e(G_1[s_j]) = 0 \text{ for all } s_j \in \mathcal{N}_i) \\ &= (1-p)^{\left| s_i^{(k)} \setminus \bigcup_{s_j \in \mathcal{N}_i} s_j^{(k)} \right|}. \end{aligned}$$

By the definition of f_i , the term in the exponent here is simply $|f_i|$; we therefore have

$$\mathbb{P}(\overline{\mathcal{E}_{\text{edge}, i}} \mid T_{<i}(G_1) = t_{<i}(G_1)) \leq e^{-p|f_i|} \leq e^{-(\log^{k^3} t)/2}.$$

By (4.3.13), it follows that

$$\mathbb{P}(\overline{\mathcal{E}_{\text{cost}}} \vee \mathcal{E}_{\text{edge}}) \geq 1 - \lfloor C \rfloor e^{-(\log^{k^3} t)/2} \geq 1 - C e^{-(\log^{k^3} t)/2},$$

as required. \blacktriangleleft

We will next show that $\mathcal{E}_{\text{root}}$ is likely to occur, in Lemma 86. In order to do so, we will apply Karamata's inequality to show in Lemma 85 that an arbitrary sequence of unsafely-rooted queries cannot cover too many possible tuples of roots. In order to prove Lemma 85, we first bound the effectiveness an unsafely-rooted query in terms of its cost when $\alpha \geq 1$ and the query is large.

► **Lemma 84.** *Suppose $\alpha \geq 1$. If S_i is not safely rooted and there exists $j \geq k - \lfloor \alpha \rfloor - 1$ with $|S_{i,j}| \geq t/\log^k t$, then*

$$|S_{i,k-\lfloor \alpha \rfloor-1} \times \cdots \times S_{i,k}| \leq \frac{\text{cost}(S_i)^{(\lfloor \alpha \rfloor+2)/\alpha}}{\log^k t}.$$

Proof. Without loss of generality, suppose $|S_{i,k-\lfloor \alpha \rfloor-1}| \geq |S_{i,k-\lfloor \alpha \rfloor}| \geq \cdots \geq |S_{i,k}|$. If $|S_{i,k}| = 0$, the claim is trivially true, so suppose $|S_{i,k}| \geq 1$. For brevity, let $\sigma = |S_{i,k-\lfloor \alpha \rfloor-1}|^\alpha + \cdots + |S_{i,k}|^\alpha$. We now set out parameters for an application of Karamata's inequality. To this end, we define τ_j and σ_j for $j \geq k - \lfloor \alpha \rfloor - 1$ via

$$\begin{aligned} \tau_j &= \frac{1}{\lfloor \alpha \rfloor + 1} \left(\sigma - \frac{t^\alpha}{\log^{2k\alpha} t} \right) \text{ for all } j \leq k-1, & \tau_k &= \frac{t^\alpha}{\log^{2k\alpha} t}, \\ \sigma_j &= |S_{i,j}|^\alpha \text{ for all } j \geq k - \lfloor \alpha \rfloor - 1. \end{aligned}$$

Observe that $\sum_j \tau_j = \sigma = \sum_j \sigma_j$, that $\sigma_{k-\lfloor \alpha \rfloor-1} \geq \cdots \geq \sigma_k$, and that $\tau_{k-\lfloor \alpha \rfloor-1} = \cdots = \tau_{k-1} > \tau_k$ since $\sigma \geq |S_{i,k-\lfloor \alpha \rfloor-1}|^\alpha \geq (t/\log^k t)^\alpha$ by hypothesis.

It remains to show that $(\sigma_{k-\lfloor \alpha \rfloor-1}, \dots, \sigma_k)$ majorises $(\tau_{k-\lfloor \alpha \rfloor-1}, \dots, \tau_k)$. Since S_i is not safely rooted, we have $|S_{i,k}| \leq t/\log^{2k} t$. Hence $\sigma_k \leq \tau_k$, and so $\sigma_{k-\lfloor \alpha \rfloor-1} + \cdots + \sigma_{k-1} \geq$

$\tau_{k-\lfloor\alpha\rfloor-1} + \dots + \tau_{k-1}$. Since the σ_j 's are decreasing and $\tau_{k-\lfloor\alpha\rfloor-1} = \dots = \tau_{k-1}$, it follows that for all $x \in \{0, \dots, \lfloor\alpha\rfloor\}$,

$$\sum_{j=k-\lfloor\alpha\rfloor-1}^{k-\lfloor\alpha\rfloor-1+x} \sigma_j \geq \frac{x+1}{\lfloor\alpha\rfloor+1} \sum_{j=k-\lfloor\alpha\rfloor-1}^{k-1} \sigma_j \geq \frac{x+1}{\lfloor\alpha\rfloor+1} \sum_{j=k-\lfloor\alpha\rfloor-1}^{k-1} \tau_j = \sum_{j=k-\lfloor\alpha\rfloor-1}^{k-\lfloor\alpha\rfloor-1+x} \tau_j.$$

(Here the first inequality follows from the fact that the average of the $x+1$ largest values in a set is no smaller than the average of the entire set.) Thus $(\sigma_{k-\lfloor\alpha\rfloor-1}, \dots, \sigma_k)$ majorises $(\tau_{k-\lfloor\alpha\rfloor-1}, \dots, \tau_k)$.

Finally, for all $y \geq 1$, let $\phi(y) = -\log(y^{1/\alpha})$, and observe that ϕ is a convex function. It now follows by Karamata's inequality (Lemma 21) that

$$\sum_{j=k-\lfloor\alpha\rfloor-1}^k \phi(\tau_j) \leq \sum_{j=k-\lfloor\alpha\rfloor-1}^k \phi(\sigma_j).$$

Substituting in the definition of ϕ and negating both sides yields

$$\log \left(\prod_{j=k-\lfloor\alpha\rfloor-1}^k \sigma_j^{1/\alpha} \right) \leq \log \left(\prod_{j=k-\lfloor\alpha\rfloor-1}^k \tau_j^{1/\alpha} \right).$$

Exponentiating both sides and substituting in the definitions of the σ_j 's and τ_j 's then yields

$$\prod_{j=k-\lfloor\alpha\rfloor-1}^k |S_{i,j}| \leq \frac{t}{\log^{2k} t} \cdot \left(\sigma - \frac{t^\alpha}{\log^{2k\alpha} t} \right)^{(\lfloor\alpha\rfloor+1)/\alpha} \leq \frac{t\sigma^{(\lfloor\alpha\rfloor+1)/\alpha}}{\log^{2k} t}$$

Since $\sigma \geq t^\alpha / \log^{k\alpha} t$ by hypothesis and $\sigma \leq \text{cost}(S_i)$ since $\alpha \geq 1$, it follows that

$$\prod_{j=k-\lfloor\alpha\rfloor-1}^k |S_{i,j}| \leq \frac{\sigma^{(\lfloor\alpha\rfloor+2)/\alpha}}{\log^k t} \leq \frac{\text{cost}(S_i)^{(\lfloor\alpha\rfloor+2)/\alpha}}{\log^k t},$$

and the result follows immediately. \blacktriangleleft

► **Lemma 85.** *Writing X for the set of all unsafely-rooted queries in $\{S_1, \dots, S_N\}$, we have*

$$\sum_{S_i \in X} \prod_{j=k-\lfloor\alpha\rfloor-1}^k |S_{i,j}| \leq \frac{k^k t^{\lfloor\alpha\rfloor+2-\alpha}}{\log^k t} \sum_{i \in X} \text{cost}(S_i).$$

Proof. We first split the terms of the sum according to the size of the largest part of the corresponding query among rooted vertex classes. Let

$$X^+ = \left\{ S_i \in X : \max\{|S_{i,j}| : j \geq k - \lfloor\alpha\rfloor - 1\} > t / \log^k t \right\}, \quad X^- = X \setminus X^+,$$

so that

$$\sum_{S_i \in X} \prod_{j=k-\lfloor\alpha\rfloor-1}^k |S_{i,j}| = \sum_{S_i \in X^-} \prod_{j=k-\lfloor\alpha\rfloor-1}^k |S_{i,j}| + \sum_{S_i \in X^+} \prod_{j=k-\lfloor\alpha\rfloor-1}^k |S_{i,j}|. \quad (4.3.14)$$

We first bound the X^- term. By Lemma 80 applied with $\gamma = 1/\log^k t$ (which is less than 1 since $t \geq t_0$), we have

$$\sum_{S_i \in X^-} \prod_{j=k-\lfloor\alpha\rfloor-1}^k |S_{i,j}| \leq \left(\frac{kt}{\log^k t} \right)^{\lfloor\alpha\rfloor+2-\alpha} \sum_{S_i \in X^-} \text{cost}(S_i) \leq \frac{k^k t^{\lfloor\alpha\rfloor+2-\alpha}}{\log^k t} \sum_{S_i \in X^-} \text{cost}(S_i).$$

(4.3.15)

We next bound the X^+ term, splitting into two cases depending on the value of α .

Case 1: $\alpha \leq 1$. For all $S_i \in X^+$, we have $\text{cost}(S_i) \geq (\max_i |S_i|)^\alpha \geq (t/\log^k t)^\alpha$; since S_i is safely rooted, it follows that

$$\prod_{j=k-\lfloor\alpha\rfloor-1}^k |S_{i,j}| \leq \frac{t}{\log^{2k} t} \cdot t^{\lfloor\alpha\rfloor+1} = \frac{t^{\lfloor\alpha\rfloor+2}}{\log^{2k} t} \leq \frac{t^{\lfloor\alpha\rfloor+2-\alpha}}{\log^{2k-\alpha k} t} \text{cost}(S_i) \leq \frac{t^{\lfloor\alpha\rfloor+2-\alpha}}{\log^k t} \text{cost}(S_i).$$

The last inequality holds since $\alpha \leq 1$. Summing over all $S_i \in X^+$, we obtain

$$\sum_{S_i \in X^+} \prod_{j=k-\lfloor\alpha\rfloor-1}^k |S_{i,j}| \leq \frac{t^{\lfloor\alpha\rfloor+2-\alpha}}{\log^k t} \sum_{S_i \in X^+} \text{cost}(S_i). \quad (4.3.16)$$

Case 2: $\alpha > 1$. In this case, by definition of X^+ , we can apply [Lemma 84](#) to each term in the sum; this yields

$$\sum_{S_i \in X^+} \prod_{j=k-\lfloor\alpha\rfloor-1}^k |S_{i,j}| \leq \frac{1}{\log^k t} \sum_{S_i \in X^+} \text{cost}(S_i)^{(\lfloor\alpha\rfloor+2)/\alpha} \leq \frac{1}{\log^k t} \sum_{S_i \in X^+} \text{cost}(S_i)^{\lfloor\alpha\rfloor+2}.$$

We now apply Karamata's inequality in the form of [Corollary 22](#), taking $s_i = |S_{i,1}| + \dots + |S_{i,k}|$, $W = \sum_{S_i \in X^+} \text{cost}(S_i)$, $c = kt$, and $r = \lfloor\alpha\rfloor + 2$. This yields:

$$\sum_{S_i \in X^+} \prod_{j=k-\lfloor\alpha\rfloor-1}^k |S_{i,j}| \leq \frac{(kt)^{\lfloor\alpha\rfloor+2-\alpha}}{\log^k t} \sum_{S_i \in X^+} \text{cost}(S_i) \leq \frac{k^k t^{\lfloor\alpha\rfloor+2-\alpha}}{\log^k t} \sum_{S_i \in X^+} \text{cost}(S_i). \quad (4.3.17)$$

The result therefore follows from (4.3.14) and (4.3.15) combined with (4.3.16) and (4.3.17). \blacktriangleleft

► **Lemma 86.** *We have*

$$\mathbb{P}(\overline{\mathcal{E}_{\text{cost}}} \vee \mathcal{E}_{\text{root}}) \geq 1 - \frac{Ck^k}{t^\alpha \log^k t}.$$

Proof. Let s_1, \dots, s_N be any possible sequence of values for the query sets S_1, \dots, S_N , and let \mathcal{T} be the event that $(S_1, \dots, S_N) = (s_1, \dots, s_N)$ holds. We will prove $\mathbb{P}(\mathcal{E}_{\text{cost}} \wedge \overline{\mathcal{E}_{\text{root}}} \mid \mathcal{T}) \leq Ck^k/(t^\alpha \log^k t)$, from which the result follows immediately.

Note that $\mathcal{E}_{\text{cost}}$ is a function of S_1, \dots, S_N ; if $\overline{\mathcal{E}_{\text{cost}}}$ occurs under \mathcal{T} then we are done, so suppose not. Let $X \subseteq \{s_1, \dots, s_N\}$ be the set of all queries that are not safely rooted. Recall that S_1, \dots, S_N are functions of G_1 , and that the roots $\mathcal{R}_{k-\lfloor\alpha\rfloor-1}, \dots, \mathcal{R}_k$ are independent of G_1 ; thus

$$\begin{aligned} \mathbb{P}(\mathcal{E}_{\text{cost}} \wedge \overline{\mathcal{E}_{\text{root}}} \mid \mathcal{T}) &= \mathbb{P}(\text{Some } s_i \in \{s_1, \dots, s_N\} \text{ is neither safely nor inaccurately rooted}) \\ &\leq \sum_{s_i \in X} \mathbb{P}(s_i \text{ is accurately rooted}) = \sum_{s_i \in X} \prod_{j=k-\lfloor\alpha\rfloor-1}^k \mathbb{P}(\mathcal{R}_j \in s_{i,j}) \\ &= \frac{1}{t^{\lfloor\alpha\rfloor+2}} \sum_{s_i \in X} \prod_{j=k-\lfloor\alpha\rfloor-1}^k |s_{i,j}|. \end{aligned}$$

Since $\mathcal{E}_{\text{cost}}$ occurs, the total cost of s_1, \dots, s_N is at most C ; by applying Lemma 85, we arrive at

$$\mathbb{P}(\mathcal{E}_{\text{cost}} \wedge \overline{\mathcal{E}_{\text{root}}} \mid \mathcal{T}) \leq \frac{1}{t^{\lfloor \alpha \rfloor + 2}} \sum_{s_i \in X} \prod_{j=k-\lfloor \alpha \rfloor-1}^k |s_{i,j}| \leq Ck^k / (t^\alpha \log^k t). \quad \blacktriangleleft$$

► **Lemma 87.** *We have*

$$\mathbb{P}(\overline{\mathcal{E}_{\text{cost}}} \vee \mathcal{E}_{\text{nonroot}}) \geq 1 - \frac{C}{t^\alpha \log^{2k} t}.$$

Proof. Let s_1, \dots, s_N be any possible sequence of values for S_1, \dots, S_N , let \vec{q} be any possible value for \vec{Q} , and let \mathcal{T} be the event that $(S_1, \dots, S_N) = (s_1, \dots, s_N)$ and $\vec{Q} = \vec{q}$. We will prove

$$\mathbb{P}(\mathcal{E}_{\text{cost}} \wedge \overline{\mathcal{E}_{\text{nonroot}}} \mid \mathcal{T}) \leq C / (t^\alpha \log^{2k} t).$$

From this, the result follows immediately.

If $\overline{\mathcal{E}_{\text{cost}}}$ occurs under \mathcal{T} then we are done, so suppose not; then $\sum_i \text{cost}(s_i) \leq C$. Let X be the set of all queries in s_1, \dots, s_N which are safely rooted and for which there exists $j \leq k - \lfloor \alpha \rfloor - 2$ with $|s_{i,j}| \leq 1/(\xi q_j)$. Recall that S_1, \dots, S_N are deterministic functions of G_1 , and that \vec{Q} is independent of G_1 ; thus

$$\begin{aligned} \mathbb{P}(\mathcal{E}_{\text{cost}} \wedge \overline{\mathcal{E}_{\text{nonroot}}} \mid \mathcal{T}) &= \mathbb{P}(\text{Some } s_i \in X \text{ has } e(H_2[s_i]) > 0 \mid \vec{Q} = \vec{q}) \\ &\leq \sum_{s_i \in X} \mathbb{P}(e(H_2[s_i]) > 0 \mid \vec{Q} = \vec{q}). \end{aligned}$$

We have $e(H_2[s_i]) = 0$ whenever $\mathcal{X}_j \cap s_{i,j} = \emptyset$ for any j , so it follows that

$$\mathbb{P}(\mathcal{E}_{\text{cost}} \wedge \overline{\mathcal{E}_{\text{nonroot}}} \mid \mathcal{T}) \leq \sum_{s_i \in X} \min_{j \leq k - \lfloor \alpha \rfloor - 2} \mathbb{P}(\mathcal{X}_j \cap s_{i,j} \neq \emptyset \mid \vec{Q} = \vec{q}).$$

By the definition of X , for all $s_i \in X$ there exists $j \leq k - \lfloor \alpha \rfloor - 2$ such that $|s_{i,j}| \leq 1/(\xi q_j)$, and each vertex in $s_{i,j}$ lies in \mathcal{X}_j with probability q_j . It follows by a union bound over all vertices in $s_{i,j}$ that

$$\mathbb{P}(\mathcal{E}_{\text{cost}} \wedge \overline{\mathcal{E}_{\text{nonroot}}} \mid \mathcal{T}) \leq \sum_{s_i \in X} \frac{1}{\xi q_j} \cdot q_j = \frac{|X|}{\xi}.$$

Now, since $\mathcal{E}_{\text{cost}}$ occurs under \mathcal{T} , the total cost of s_1, \dots, s_N is at most C . Since each query in X is safely rooted, it has cost at least $(t/\log^{2k} t)^\alpha$. It follows that the total number of queries in X is at most $C/(t/\log^{2k} t)^\alpha$, and so

$$\mathbb{P}(\mathcal{E}_{\text{cost}} \wedge \overline{\mathcal{E}_{\text{nonroot}}} \mid \mathcal{T}) \leq \frac{C \log^{2k\alpha} t}{\xi t^\alpha} \leq \frac{C \log^{2k^2} t}{\xi t^\alpha}. \quad (4.3.18)$$

By definition, and using the fact that $k - \lfloor \alpha \rfloor + 2 \geq 1$, we have

$$\xi = (pn^{\lfloor \alpha \rfloor + 2})^{-1/(2k\beta)} = t^{\frac{k - \lfloor \alpha \rfloor + 2}{4k\beta}} \geq t^{1/(4k\beta)} \geq t^{\frac{20k^4 \log \log t}{4k \log t}} \geq (\log t)^{5k^3},$$

and the result follows from (4.3.18). ◀

► **Lemma 88.** *We have*

$$\mathbb{P}(\overline{\mathcal{E}_{\text{cost}}} \vee \mathcal{E}_{\text{inacc}}) \geq 1 - \frac{2^{5k} k^{7k} C}{t^\alpha} \cdot \left(\frac{\log \log t}{\log t} \right)^{k - \lfloor \alpha \rfloor - 3}.$$

Proof. Let s_1, \dots, s_N be any possible sequence of values for S_1, \dots, S_N , and let \mathcal{T} be the event that $(S_1, \dots, S_N) = (s_1, \dots, s_N)$. We will prove that

$$\mathbb{P}(\overline{\mathcal{E}_{\text{cost}}} \vee \mathcal{E}_{\text{inacc}} \mid \mathcal{T}) \geq 1 - \frac{2^{5k} k^{3k} C}{t^\alpha} \cdot \left(\frac{\log \log t}{\log t} \right)^{k - \lfloor \alpha \rfloor - 3},$$

from which the result follows immediately.

If $\overline{\mathcal{E}_{\text{cost}}}$ occurs under \mathcal{T} then we are done, so suppose not; then $\sum_i \text{cost}(s_i) \leq C$. Recall that S_1, \dots, S_N are deterministic functions of G_1 , and that H_2 is independent of G_1 ; thus

$$\mathbb{P}(\overline{\mathcal{E}_{\text{cost}}} \vee \mathcal{E}_{\text{inacc}} \mid \mathcal{T}) = 1 - \mathbb{P}(\text{some } s_i \text{ is accurate}) \geq 1 - \sum_{i=1}^N \mathbb{P}(s_i \text{ is accurate}). \quad (4.3.19)$$

A given query s_i is accurate precisely when it is accurately rooted (which depends only on $\vec{\mathcal{R}} := (\mathcal{R}_{k-\lfloor \alpha \rfloor-1}, \dots, \mathcal{R}_k)$) and accurately profiled (which depends only on $\vec{\mathcal{Q}}$). These events are independent, so we have

$$\mathbb{P}(\text{some } s_i \text{ is accurate}) = \mathbb{P}(s_i \text{ is accurately rooted}) \cdot \mathbb{P}(s_i \text{ is accurately profiled}). \quad (4.3.20)$$

Observe that s_i is accurately profiled for at most one choice of \mathcal{Q} . Let m be the number of ways in which β can be decomposed into a sum of $k - \lfloor \alpha \rfloor - 2$ ordered integers in $\{0, \dots, B\}$. Each such sequence of integers corresponds to the numerators of the exponents of a possible value of \mathcal{Q} , and so there are m possible values of \mathcal{Q} in total. We therefore have

$$\begin{aligned} \mathbb{P}(s_i \text{ is accurately rooted}) &= |s_{i,k-\lfloor \alpha \rfloor-1} \times \dots \times s_{i,k}| / t^{\lfloor \alpha \rfloor + 2}, \\ \mathbb{P}(s_i \text{ is accurately profiled}) &= 1/m. \end{aligned}$$

It follows from (4.3.19) and (4.3.20) that

$$\mathbb{P}(\overline{\mathcal{E}_{\text{cost}}} \vee \mathcal{E}_{\text{inacc}} \mid \mathcal{T}) \geq 1 - \frac{1}{mt^{\lfloor \alpha \rfloor + 2}} \sum_{i=1}^N |s_{i,k-\lfloor \alpha \rfloor-1} \times \dots \times s_{i,k}|.$$

Recall that $\mathcal{E}_{\text{cost}}$ occurs and so s_1, \dots, s_N have total cost at most C ; applying Lemma 80 with $\gamma = 1$, we obtain

$$\mathbb{P}(\overline{\mathcal{E}_{\text{cost}}} \vee \mathcal{E}_{\text{inacc}} \mid \mathcal{T}) \geq 1 - \frac{Ck^k}{mt^\alpha}. \quad (4.3.21)$$

It remains to bound m below. We will first bound B below. Recall that $x = pt^{\lfloor \alpha \rfloor + 2} = t^{-(k-\lfloor \alpha \rfloor-2)/2}$; thus by the definition of B , we have

$$B = \left\lfloor \frac{\log((24 \log t)/t)}{\log(x^{1/\beta})} \right\rfloor = \left\lfloor -\frac{2\beta \log((24 \log t)/t)}{(k - \lfloor \alpha \rfloor - 2) \log t} \right\rfloor = \left\lfloor \frac{2\beta(\log t - \log(24 \log t))}{(k - \lfloor \alpha \rfloor - 2) \log t} \right\rfloor.$$

Since $t \geq t_0 \geq e^{640}$, it follows that

$$B \geq \frac{3}{2} \cdot \frac{\beta}{k - \lfloor \alpha \rfloor - 2}. \quad (4.3.22)$$

We now exploit this bound on B in order to bound m below. For every choice of integers $z_1, \dots, z_{k-\lfloor \alpha \rfloor - 3}$ in $[(1 - \frac{1}{2k})\beta/(k - \lfloor \alpha \rfloor - 2), (1 + \frac{1}{2k})\beta/(k - \lfloor \alpha \rfloor - 2)]$, we have

$$\begin{aligned} \sum_{\ell=1}^{k-\lfloor \alpha \rfloor - 3} z_\ell &\leq \frac{(k - \lfloor \alpha \rfloor - 3)\beta}{k - \lfloor \alpha \rfloor - 2} + \frac{(k - \lfloor \alpha \rfloor - 3)\beta}{2k(k - \lfloor \alpha \rfloor - 2)} \leq \frac{(k - \lfloor \alpha \rfloor - 3)\beta}{k - \lfloor \alpha \rfloor - 2} + \frac{\beta}{2(k - \lfloor \alpha \rfloor - 2)} \\ &= \beta - \frac{\beta}{2(k - \lfloor \alpha \rfloor - 2)}, \end{aligned}$$

and similarly

$$\sum_{\ell=1}^{k-\lfloor \alpha \rfloor - 3} z_\ell \geq \beta - \frac{3\beta}{2(k - \lfloor \alpha \rfloor - 2)}.$$

By (4.3.22), it follows that for all such choices of integers $z_1, \dots, z_{k-\lfloor \alpha \rfloor - 3}$ there is a unique integer $z_{k-\lfloor \alpha \rfloor - 2} \in [0, B]$ such that $\sum_i z_i = \beta$. Thus

$$m \geq \left| \mathbb{Z} \cap \left[\left(1 - \frac{1}{2k}\right) \frac{\beta}{k - \lfloor \alpha \rfloor - 1}, \left(1 + \frac{1}{2k}\right) \frac{\beta}{k - \lfloor \alpha \rfloor - 1} \right] \right|^{k-\lfloor \alpha \rfloor - 3}.$$

Since $t \geq t_0$, it follows that

$$m \geq \left(\frac{\beta}{2k(k - \lfloor \alpha \rfloor - 2)} \right)^{k-\lfloor \alpha \rfloor - 3}.$$

Again since $t \geq t_0$, we have $\beta \geq 1$, and so we can bound away the floor in its definition to obtain

$$m \geq \left(\frac{\log t}{40k^5(k - \lfloor \alpha \rfloor - 2) \log \log t} \right)^{k-\lfloor \alpha \rfloor - 3} \geq \frac{(\log t)^{k-\lfloor \alpha \rfloor - 3}}{2^{5k} k^{6k} (\log \log t)^{k-\lfloor \alpha \rfloor - 3}}.$$

The result therefore follows immediately from (4.3.21). \blacktriangleleft

With these probability bounds in place, Theorem 67 now follows easily.

► **Theorem 67.** *Let $t, k \geq 1$, let $\alpha \in [0, k - 3]$, and let $\text{cost}(x) = x^\alpha$. Let $t_0 := 2^{400k^6}$, and suppose that $t \geq t_0$. There exist two correlated distributions \mathcal{G}_1 and \mathcal{G}_2 on k -partite k -hypergraphs whose vertex classes V_1, \dots, V_k each have size t with the following properties:*

- (i) *We have $\mathbb{P}_{(G_1, G_2) \sim (\mathcal{G}_1, \mathcal{G}_2)}[e(G_2) \geq 4e(G_1)] \geq 19/20$.*
- (ii) *Let*

$$C := \frac{t^\alpha}{2^{5k+7} k^{7k}} \cdot \left(\frac{\log t}{\log \log t} \right)^{k-\lfloor \alpha \rfloor - 3}.$$

Suppose A is a deterministic cIND-oracle algorithm with

$$\mathbb{P}_{(G_1, G_2) \sim (\mathcal{G}_1, \mathcal{G}_2)} \left(A(\text{cIND}(G_1)) \neq A(\text{cIND}(G_2)) \right) \geq 2/3, \quad (4.3.1)$$

which only uses cIND-queries $S = (S_1, \dots, S_k)$ with $S_i \subseteq V_i$ for all $i \in [k]$. Then the expected oracle cost of A (with respect to cost) under random inputs $G_1 \sim \mathcal{G}_1$ satisfies $\mathbb{E}_{G_1 \sim \mathcal{G}_1}[\text{cost}(A, G_1)] \geq C/2$.

Proof. Part (i) of the theorem is immediate from Lemma 71. Suppose A is a deterministic cIND-oracle algorithm with

$$\mathbb{P}_{(G_1, G_2) \sim (\mathcal{G}_1, \mathcal{G}_2)} \left(A(\text{cIND}(G_1)) \neq A(\text{cIND}(G_2)) \right) \geq 2/3. \quad (4.3.23)$$

By Lemma 81, it follows that

$$\mathbb{P}(\mathcal{E}_{\text{cost}} \wedge \mathcal{E}_{\text{edge}} \wedge \mathcal{E}_{\text{root}} \wedge \mathcal{E}_{\text{nonroot}} \wedge \mathcal{E}_{\text{inacc}}) \leq 1/3. \quad (4.3.24)$$

Observe that since $k \geq 2$ and $t \geq t_0 \geq \exp(640)$, we have $\log^{k^3} t \geq 2k(\ln t + \ln \ln t)$, and hence $e^{-(\log^{k^3} t)/2} \leq 1/(t^\alpha \log^k t)$. By Lemmas 83, 86, 87 and 88 together with a union bound, it follows that

$$\begin{aligned} & \mathbb{P} \left(\overline{\mathcal{E}_{\text{cost}}} \vee (\mathcal{E}_{\text{edge}} \wedge \mathcal{E}_{\text{root}} \wedge \mathcal{E}_{\text{nonroot}} \wedge \mathcal{E}_{\text{inacc}}) \right) \\ & \geq 1 - \left(C e^{-(\log^{k^3} t)/2} + \frac{C k^k}{t^\alpha \log^k t} + \frac{C}{t^\alpha \log^{2k} t} + \frac{2^{5k} k^{7k} C}{t^\alpha} \cdot \left(\frac{\log \log t}{\log t} \right)^{k - \lfloor \alpha \rfloor - 3} \right) \\ & \geq 1 - \frac{C k^k}{t^\alpha} \left(\frac{3}{\log^k t} + 2^{5k} k^{6k} \left(\frac{\log \log t}{\log t} \right)^{k - \lfloor \alpha \rfloor - 3} \right) \\ & \geq 1 - \frac{4C}{t^\alpha} \cdot 2^{5k} k^{7k} \left(\frac{\log \log t}{\log t} \right)^{k - \lfloor \alpha \rfloor - 3}. \end{aligned}$$

By the definition of C , it follows that

$$\mathbb{P} \left(\overline{\mathcal{E}_{\text{cost}}} \vee (\mathcal{E}_{\text{edge}} \wedge \mathcal{E}_{\text{root}} \wedge \mathcal{E}_{\text{nonroot}} \wedge \mathcal{E}_{\text{inacc}}) \right) \geq 31/32,$$

and hence

$$\mathbb{P} \left(\mathcal{E}_{\text{cost}} \vee \overline{(\mathcal{E}_{\text{edge}} \wedge \mathcal{E}_{\text{root}} \wedge \mathcal{E}_{\text{nonroot}} \wedge \mathcal{E}_{\text{inacc}})} \right) \leq 1/32.$$

By (4.3.24), it follows that

$$\mathbb{P}(\mathcal{E}_{\text{cost}}) \leq 1/32 + 1/3 < 1/2. \quad (4.3.25)$$

By Markov's inequality, with probability at least $1/2$, $\text{cost}(A, G_1)$ is at most twice its expected value. If $\mathbb{E}(\text{cost}(A, G_1))$ were at least $C/2$ then this would contradict (4.3.25), so we must have $\mathbb{E}(\text{cost}(A, G_1)) \leq C/2$ as required. \blacktriangleleft

References

- 1 Raghavendra Addanki, Andrew McGregor, and Cameron Musco. Non-adaptive edge counting and sampling via bipartite independent set queries. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 2:1–2:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ESA.2022.2.
- 2 Noga Alon and Joel H. Spencer. *The Probabilistic Method, Third Edition*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 2008.
- 3 Simon Apers, Yuwan Efron, Paweł Gawrychowski, Troy Lee, Sagnif Mukhopadhyay, and Danupon Nanongkai. Cut query algorithms with star contraction. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, 2022, to appear. arXiv:2201.05674.
- 4 Paul Beame, Sariel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. Edge estimation with independent set oracles. *ACM Trans. Algorithms*, 16(4):52:1–52:27, 2020. doi:10.1145/3404867.

- 5 Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. Hyperedge estimation using polylogarithmic subset queries. *CoRR*, abs/1908.04196, 2019. URL: <https://arxiv.org/abs/1908.04196>, arXiv:1908.04196.
- 6 Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. Faster counting and sampling algorithms using colorful decision oracle. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPIcs*, pages 10:1–10:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.STACS.2022.10.
- 7 Arijit Bishnu, Arijit Ghosh, Sudeshna Kolay, Gopinath Mishra, and Saket Saurabh. Parameterized query complexity of hitting set using stability of sunflowers. In Wen-Lian Hsu, Der-Tsai Lee, and Chung-Shou Liao, editors, *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan*, volume 123 of *LIPIcs*, pages 25:1–25:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ISAAC.2018.25.
- 8 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *J. Comput. Syst. Sci.*, 87:119–139, 2017. doi:10.1016/j.jcss.2017.03.003.
- 9 Cornelius Brand, Holger Dell, and Thore Husfeldt. Extensor-coding. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 151–164. ACM, 2018. doi:10.1145/3188745.3188902.
- 10 Marco Bressan and Marc Roth. Exact and approximate pattern counting in degenerate graphs: New algorithms, hardness results, and complexity dichotomies. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 276–285, 2022. doi:10.1109/FOCS52979.2021.00036.
- 11 Karl Bringmann. *Sampling from discrete distributions and computing Fréchet distances*. PhD thesis, Saarland University, 2015. URL: <http://scidok.sulb.uni-saarland.de/volltexte/2015/5988/>.
- 12 Timothy Chan, Virginia Vassilevska Williams, and Yinzhan Xu. Fredman’s trick meets dominance product: Fine-grained complexity of unweighted APSP, 3SUM counting, and more. *CoRR*, abs/2303.14572, 2023. URL: <https://arxiv.org/abs/2303.14572>, arXiv:2303.14572.
- 13 Xi Chen, Amit Levi, and Erik Waingarten. Nearly optimal edge estimation with independent set queries. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2916–2935. SIAM, 2020. doi:10.1137/1.9781611975994.177.
- 14 Holger Dell and John Lapinskas. Fine-grained reductions from approximate counting to decision. *ACM Trans. Comput. Theory*, 13(2):8:1–8:24, 2021. doi:10.1145/3442352.
- 15 Holger Dell, John Lapinskas, and Kitty Meeks. Approximately counting and sampling small witnesses using a colorful decision oracle. *SIAM J. Comput.*, 51(4):849–899, 2022. doi:10.1137/19m130604x.
- 16 Martin Farach-Colton and Meng-Tsung Tsai. Exact sublinear binomial sampling. *Algorithmica*, 73(4):637–651, 2015. doi:10.1007/s00453-015-0077-8.
- 17 William Feller. *An introduction to probability theory and its applications. Vol. II*. Second edition. John Wiley & Sons Inc., New York, 1971.
- 18 George S. Fishman. Sampling from the binomial distribution on a computer. *Journal of the American Statistical Association*, 74(366):418–423, 1979. URL: <http://www.jstor.org/stable/2286346>, doi:10.2307/2286346.
- 19 Jacob Focke, Leslie Ann Goldberg, Marc Roth, and Stanislav Zivný. Approximately counting answers to conjunctive queries with disequalities and negations. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*,

- PODS '22, page 315–324, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3517804.3526231.
- 20 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM*, 63(4):29:1–29:60, 2016. doi:10.1145/2886094.
 - 21 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1), mar 2007. doi:10.1145/1206035.1206036.
 - 22 Svante Janson, Tomasz Luczak, and Andrzej Rucinski. *Random graphs*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 2000. doi:10.1002/9781118032718.
 - 23 Hung Le. Approximate distance oracles for planar graphs with subpolynomial error dependency. In *Proceedings of the 2023 annual ACM-SIAM symposium on discrete algorithms (SODA)*, pages 1877–1904, 2023.
 - 24 Daniel Lokshtanov, Andreas Björklund, Saket Saurabh, and Meirav Zehavi. Approximate counting of k -paths: Simpler, deterministic, and in polynomial space. *ACM Trans. Algorithms*, 17(3):26:1–26:44, 2021. doi:10.1145/3461477.
 - 25 Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Efficient computation of representative weight functions with applications to parameterized counting (extended version). In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 179–198. SIAM, 2021. doi:10.1137/1.9781611976465.13.
 - 26 Colin McDiarmid. On the method of bounded differences. In J. Editor Siemons, editor, *Surveys in Combinatorics, 1989: Invited Papers at the Twelfth British Combinatorial Conference*, page 148–188. Cambridge University Press, 1989. doi:10.1017/CBO9781107359949.008.
 - 27 Kitty Meeks. The challenges of unbounded treewidth in parameterised subgraph counting problems. *Discrete Applied Mathematics*, 198:170 – 194, 2016. doi:10.1016/j.dam.2015.06.019.
 - 28 Kitty Meeks. Randomised enumeration of small witnesses using a decision oracle. *Algorithmica*, 81(2):519–540, 2019. doi:10.1007/s00453-018-0404-y.
 - 29 Sagnik Mukhopadhyay and Danupon Nanongkai. Weighted min-cut: Sequential, cut-query, and streaming algorithms. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, page 496–509, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3357713.3384334.
 - 30 Moritz Müller. Randomized approximations of parameterized counting problems. In Hans L. Bodlaender and Michael A. Langston, editors, *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, volume 4169 of *Lecture Notes in Computer Science*, pages 50–59. Springer, 2006. doi:10.1007/11847250_5.
 - 31 Pan Peng and Jiapeng Zhang. Towards a query-optimal and time-efficient algorithm for clustering with a faulty oracle. In *Proceedings of Machine Learning Research (COLT)*, volume 134, pages 1–19, 2021.
 - 32 Josip Pečarić, Frank Proschan, and Y.L. Tong. *Convex functions, partial orderings, and statistical applications*. Academic Press Inc., San Diego, 1992.
 - 33 Cyrus Rashtchian, David P. Woodruff, and Hanlin Zhu. Vector-matrix-vector queries for solving linear algebra, statistics, and graph problems. In Jaroslav Byrka and Raghu Meka, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2020, August 17-19, 2020, Virtual Conference*, volume 176 of *LIPIcs*, pages 26:1–26:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.APPROX/RANDOM.2020.26.
 - 34 Jakub Tetek and Mikkel Thorup. Edge sampling and graph parameter estimation via vertex neighborhood accesses. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1116–1129. ACM, 2022. doi:10.1145/3519935.3520059.

- 35 Marc Thurley. An approximation algorithm for $\#k$ -SAT. In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*, volume 14 of *LIPIcs*, pages 78–87. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.STACS.2012.78.
- 36 Leslie G. Valiant and Vijay V. Vazirani. NP is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47:85–93, 1986. doi:10.1016/0304-3975(86)90135-0.
- 37 R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524.
- 38 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018. doi:10.1145/3186893.
- 39 Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 222–227. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.24.