The Limits of Black-Box Reductions for All-Pairs Detection

Nathan S. Sheffield, Virginia Vassilevska Williams, and Zoe Xi

Abstract

Given 3 sets of objects and some tripartite relation R, we consider the problem of determining for each pair of objects whether there exists a third object satisfying the relation with them. For many specific relations R, an appropriate form of matrix product allows us to solve this "all-pairs" detection problem just as efficiently as we could detect whether a *single* relation-satisfying triple exists. We study the extent to which all-pairs detection can be used to generically solve other problems (and vice-versa), defining a natural corresponding hypergraph query model and proving upper and lower bounds on the cost of listing and counting hyperedges.

Our results have implications in fine-grained complexity. Our positive results yield new reductions between several classes of triangle and matrix problems — for instance, we demonstrate that an $O(n^{2.53})$ algorithm for computing the equality product of two $n \times n$ matrices would imply an improvement on known algorithms for computing boolean $(\min, +)$-product. Our negative results can be thought of as barriers against natural fine-grained proof techniques: we show that no appropriately "black-box" reductions are capable of demonstrating a tight equivalence between boolean matrix multiplication and triangle detection, a subcubic equivalence between triangle counting and binary integer matrix multiplication, or a tight equivalence between boolean matrix multiplication and listing n^2 triangles, despite the fact that all of these equivalences are conjectured to hold.

1 Introduction

Many of the most fundamental problems in fine-grained complexity are variants of triangle detection in weighted tripartite graphs. For instance, in addition to the standard (unweighted) triangle detection problem, there has been much work studying *negative triangles* (i.e. triangles where the sum of the edge weights is negative), *exact triangles* (i.e. triangles where the sum of the edge weights is 0), *monochromatic triangles* (i.e. triangles where all edge weights are distinct), and *nondecreasing triangles* (i.e. triangles where the weight of the edge between parts I and J is less than or equal to the weight of the edge between parts J and K is less than or equal to the weight of the edge between parts K and I), among others. In general, for any relation $R \subseteq \mathbb{Z}^3$, one could define an "R-triangle" to be a triangle (i, j, k) whose edge weights satisfy $(w_{ij}, w_{jk}, w_{ik}) \in R$. Even for a given R, there are a whole host of problems that may end up being relevant in fine-grained complexity, beyond the question of just detecting whether an R-triangle exists. One may want to *count* the number of R-triangles, or *find* some number of R-triangles, or possibly detect *for every edge* whether that edge belongs to an R-triangle. This latter "all-edge detection" problem is of particular interest because it is closely related to matrix multiplication. (More precisely, for many natural relations R, the all-edge R-triangle detection problem can be solved with a single instance of matrix multiplication over a related algebraic structure. Classic examples include all-edge unweighted triangle, which is equivalent to boolean matrix multiplication, and all-edge negative triangle, which is equivalent to $(\min, +)$-product.)

One particularly celebrated result in this area is that, for any relation R, the problems of R-triangle detection and all-edge R-triangle detection are *subcubically equivalent* [WW10]. That is, Vassilevska Williams and Williams show that for any $0 \leq \varepsilon \leq 1$, given a time-$O(n^{3-\varepsilon})$ algorithm for R-triangle detection, one can solve all-edge R-triangle detection in time $O(n^{3-\varepsilon/3})$. This is quite a strong result: for a problem like all-edge negative triangle where we expect (assuming APSP requires cubic time) no subcubic algorithm to exist, this tells us that detecting a single negative triangle should also require cubic time. However, in cases where we *are* able to get some nontrivial algorithms, this result is somewhat less satisfying. For instance,

even if we had an optimal $O(n^2)$ time algorithm for (unweighted) triangle detection, Vassilevska Williams and Williams' reduction would yield only an $O(n^{2+2/3})$-time algorithm for boolean matrix multiplication, which is worse than the currently known matrix multiplication exponent. Many conjecture that boolean matrix multiplication and triangle detection should be *fully* equivalent, in the sense that a $T(n)$ runtime for one should imply a $O(T(O(n)))$ runtime for the other. In fact, we do not know of *any* relation R for which it is suspected that all-edge detection is more difficult than detection. Despite this, there has been no success in finding quantitative improvements over Vassilevska Williams and Williams' reduction. This leads to the following question:

Question 1: Is a tighter equivalence possible between triangle detection and all-edge triangle detection? In particular, is there a tighter reduction from all-edge triangle detection to triangle detection?

One viewpoint one might have with regards to this question is as follows: over the past 15 years, nobody has found any relation such that there is evidence that all-edge detection is harder than detection, but also nobody has managed to come up with an exact reduction between these problems. So perhaps it is the case that these problems *are* always equivalent, but this equivalence just *isn't* witnessed by any fine-grained reduction. As reductions are essentially the only way we know to establish equivalence in complexity between problems, this would be a strong barrier to our ability to prove this equivalence, even if it is true.

Of course, in order for this viewpoint to be coherent, one must place some stipulations on what it means for this fact to be "provable via fine-grained reduction". If these two problems have the same complexity, then there is always the trivial reduction between them where one solves all-edge detection directly. To make any sort of meaningful statement, we need to consider a restricted form of reduction — ideally one that captures the standard techniques used in known fine-grained reductions, while not admitting the trivial reductions of the form "solve the problem from scratch in the optimal runtime". The definition we propose is as follows:

Definition 1. For two runtimes T_A and T_B in terms of a vertex count n and edge count m, an R-triangle problem A (e.g. detection or all-edge detection) (T_B, T_A)-***black-box reduces*** to another R-triangle problem B if there exists an algorithm M running in $\widetilde{O}\left(T_A(n,m)\right)$ time and making queries to subgraphs $G_1, \ldots, G_r \subseteq G$ of the input, for $\sum_i T_B(|V(G_i)|, |E(G_i)|) \leq T_A(n,m)$, to an oracle for B, such that for any relation R, M successfully computes A when instantiated with a valid B oracle.

There are two key restrictions this definition imposes. One is that every query is made to a subgraph of the input graph. This is the case for essentially all known fine-grained reductions between triangle problems, and is an important restriction for the purposes of some of our nonexistence proofs, but is in some sense not the most crucial part of the definition — one could imagine that perhaps with alternative techniques one could rule out reductions even without that condition. The essential fact is that M is *not allowed to depend on* R. That is, we would like a single reduction that works when instantiated for *any* relation: even if problems A and B are not efficiently solvable for that relation, if instantiated with a black-box solving B, the reduction will successfully solve A. Given this definition, we can now reframe Question 1 as follows:

Question 1': does all-edge triangle detection $\left(n^{3-\varepsilon}, n^{3-\delta}\right)$-black box reduce to triangle detection for any $\varepsilon > 0$, $\delta > \varepsilon/3$?

As we show in Section 3, it turns out that this question has a relatively straightforward (unconditional) negative answer. In the remainder of this introduction, we introduce a model of hypergraph query complexity corresponding to all-edge detection in a black-box setting, and state our main results in that model. Then, in Section 2, we show how to use those results to derive both new barriers against black-box reductions, as well as some new positive results, for a number of open questions in fine-grained complexity.

1.1 All-s codegree and all-s containment queries

If we are dealing with an arbitrary unknown relation R, these R-triangle problems can be thought of simply as *hyperedge detection* (resp. listing, counting, etc.) problems on an unknown hypergraph. Observe that, if the edge weights of the graph are all distinct, by choice of the relation R one can make it so that any arbitrary collection of 3-tuples of vertices correspond to R-triangles. So, if one wishes to use a reduction to

some triangle problem B to solve a problem A in a black-box sense, one is effectively attempting to answer a question about an unknown 3-uniform hypergraph, where access to this hypergraph is provided in the form of queries to B.

This perspective has already proved quite fruitful in generating generally-applicable positive results in fine-grained complexity. It has been observed that the question of "what information about the set of witnesses for a relation can we determine via black-box use of a detection algorithm" can be thought of in terms of query complexity in the ***independent set oracle*** model, where access to an unknown hypergraph is provided via an oracle which takes in a set of vertices and determines whether they contain some hyperedge. One known result is that, for a k-uniform, k-partite hypergraph, it is possible to approximately count the number of hyperedges by making only $\mathsf{polylog}(n)$ many independent set queries — in particular, this gives a black-box reduction from approximate counting to detection for triangle problems, as well as higher-arity relations like k-SUM or k-CLIQUE [Bea+20; DL21; Bha+19; DLM22; CEV25].

In this paper, we consider a new type of hypergraph query where, instead of just learning whether some subset of the 3-uniform hypergraph contains at least one hyperedge, we learn *for every pair of vertices* whether that pair belongs to a hyperedge in the queried subset. This is the kind of information we would get from an all-edge detection algorithm — in particular, it is the kind of information we get "for free" when we use a matrix multiplication-based algorithm to solve a detection problem. Given that it arises with no additional cost for the best known detection algorithms for many problems, it is natural to wonder whether all-pairs detection can be (in a black-box sense) more useful than standard detection for solving other problems. We note that the connection to matrix multiplication makes this kind of question most interesting from a fine-grained complexity standpoint in the case of all-pairs detection in 3-uniform hypergraphs, but that in fact one could ask a more general version about all-s detection in r-uniform hypergraphs for any s and r. As many of our results apply in this more general setting, we state the oracle model as follows:

Definition 2. For a fixed integer s, an ***all-s containment oracle*** for an unknown hypergraph $H = (V, E)$ takes as input a pair of subsets $Q \subseteq V$ and $R \subseteq \mathcal{P}(Q)$ (where $\mathcal{P}$ denotes the power set), and returns the list of all s-element subsets of Q that are contained in at least one hyperedge in $R \cap E$.

We also define the following related model, a strengthened version of hyperedge *counting* queries (also known as ***additive queries*** in the literature):

Definition 3. For a fixed integer s, an ***all-s codegree oracle*** for an unknown hypergraph $H = (V, E)$ takes as input a pair of subsets $Q \subseteq V$ and $R \subseteq \mathcal{P}(Q)$, and returns the codegree of (i.e. number of hyperedges containing) each of the $\binom{|Q|}{s}$ distinct sets of s vertices in the hypergraph $(Q, R \cap E)$.

We will discuss primarily the weaker ***induced*** versions of these oracles, in which we always require $R = \mathcal{P}(Q)$ — that is, the input to a query is simply a collection of vertices and the output is the codegree of every s-tuple in the induced subhypergraph. However, it is relevant to some of the discussion of fine-grained complexity that most of our lower bounds hold for the stronger model where some hyperedges can be excluded from consideration (so $R \subsetneq \mathcal{P}(Q)$). We will generally work with r-partite, r-uniform hypergraphs. In that setting, we could also weaken the oracles by letting them only return information about the s-tuples of vertices spanning a specific set of parts (once again, this is a relevant notion for certain questions in fine-grained complexity — see Section 2).

Definition 4. For an r-partite hypergraph $H = (X_1 \sqcup \cdots \sqcup X_r, E)$, a ***specified-parts*** all-s codegree oracle takes as input a pair of subsets $Q \subseteq V$ and $R \subseteq \mathcal{P}(Q)$ and returns the codegree of every tuple $(v_1, \ldots, v_s) \in (Q \cap X_1) \times \cdots \times (Q \cap X_s)$ in the subhypergraph $(Q, R \cap E)$. (Correspondingly, a specified-parts all-s containment oracle returns whether each of these values is 0.)

As our motivation for studying algorithms in these query models is to instantiate the oracles with efficient algorithms and get an efficient algorithm out, our results will deal with oracles whose costs scale with the size of their input, as opposed to unit cost queries. When we say an oracle has "cost n^c" for some c, this means that an algorithm making queries $(Q_1, R_1), \ldots, (Q_q, R_q)$ is defined to have query complexity

$\sum_{1\le i\le q}|Q_i|^c$. Our cost functions will always be of the form n^c for some $s \le c \le r$, where r is the uniformity of the hypergraph, as an all-s algorithm with cost less than n^s is impossible due to the output size, and an all-s algorithm with cost $O(n^r)$ is trivial so long as we can efficiently decide whether a given r-tuple is an edge.

In addition to relating these query models to previously studied ones, the primary problem we will focus on solving with these oracles is the ***exact recovery*** problem: given query access to an unknown hypergraph, can we learn a complete explicit description of the hypergraph? For general r-uniform hypergraphs, it is reasonably straightforward to see that no nontrivial algorithm can solve this: a full description of the hypergraph is simply too much information to obtain with few queries. However, if we restrict the hypergraph to have a small number of hyperedges, this is an interesting problem that has been studied previously in a number of different query models.

1.2 Main results

Note that, for all of our results, we treat parameters s, c, and r as constants, so that our asymptotic notation is asymptotic in terms of the vertex and edge counts. We write tildes over O, Ω, o, and ω to denote hiding of polylog factors in addition to constants.

We begin in Section 3 by comparing our all-s codegree and all-s containment oracles with the previously studied independent set and additive (i.e. hyperedge counting) oracles. We show that (the not-necessarily-induced version of) independent set queries can simulate all-s containment queries at some quantitative loss, via a direct generalization of Vassilevska Williams and Williams's reduction from boolean matrix multiplication to triangle detection. However, we show that this quantitative loss is necessary and tight, even if we instead have access to additive queries. We also observe that an all-s containment oracle can do no better than the trivial approach for simulating additive queries (meaning also that it cannot simulate an all-s codegree oracle).

Theorem 1. Given any $s < c < r$, for unknown r-partite, r-uniform hypergraphs we have the following:

i) Given a cost-n^c oracle for detecting whether a subhypergraph contains an edge, all-s containment queries can be answered at cost $\widetilde{O}\left(n^{\left(s+\frac{c(r-s)}{r}\right)}\right)$.

ii) Given a cost-n^c oracle for counting the number of edges in a subhypergraph, $\widetilde{\Omega}\left(n^{\left(s+\frac{c(r-s)}{r}\right)}\right)$ cost is required to answer an induced all-s containment query.

iii) Given a cost-n^c oracle for counting the number of edges in a subhypergraph, $\widetilde{\Omega}(n^{\min(r,s+c)})$ cost is required to answer an induced all-s codegree query.

iv) Given a cost-n^c all-s containment oracle, $\widetilde{\Omega}(n^r)$ cost is required to count the total number of hyperedges.

We then begin considering the problem of exact recovery using these oracles. In Section 5, we give our main algorithmic result (with the special case of $r = 3$, $s = 2$ proved in Section 4):

Theorem 2. Given cost-n^c specified-parts induced all-s containment oracle access to an unknown r-partite, r-uniform hypergraph $H = (V, E)$, we can exactly recover H with high probability at cost $\widetilde{O}\left(|V|^{r(1-\gamma)}|E|^{\gamma}\right)$, where $\gamma = \frac{r-c}{2r-s-c}$.

Then, in Section 6, we show lower bounds:

Theorem 3. For any values of $s \le c < r$, and any sufficiently large values of $|V|$ and $|E| < \frac{1}{2}\binom{|V|/r}{r}$, any algorithm that exactly recovers unknown r-partite, r-uniform hypergraphs $H = (V, E)$ with high probability using cost-n^c specified-parts all-s codegree queries must incur cost $\widetilde{\Omega}\left(|V|^{r(1-\gamma)}|E|^{\gamma}\right)$ on some instances, where $\gamma = \max\left(\frac{r-c}{2r-s-c}, \frac{r-c}{r-1}\right)$.

Theorem 4. For any values of r, $s < r$, and any sufficiently large values of $|V|$ and $|E| < \frac{1}{2}\binom{|V|/r}{r}$, any algorithm that exactly recovers unknown r-partite, r-uniform hypergraphs $H = (V, E)$ with high probability using cost-n^c all-s codegree queries must incur cost $\widetilde{\Omega}\left(|V|^{r(1-\gamma)}|E|^{\gamma}\right)$ on some instances, where $\gamma = \max\left(\frac{\binom{r}{s}(r-c)}{(r-s)+\binom{r}{s}(r-c)}, 1 - \frac{c-1}{(r-s)+(s-1)\binom{r}{s}}\right)$.

Observe that the upper bound of Theorem 2 is tight with the lower bound of Theorem 4 in our motivating case of $r = 3$, $s = 2$. (In fact, it is tight whenever $c \geq r - s + 1$.) However, it remains far from the lower bound of Theorem 4 in the case where we get this containment information for every pair of parts. While we do not know how to match Theorem 4 even in the case of $r = 3$, $s = 2$, we do demonstrate that this all-parts information can grant additional power, showing the following improvement over Theorem 2:

Theorem 5. Given cost-n^c specified-parts induced all-pairs containment oracle access to an unknown 3-partite, 3-uniform hypergraph $H = (V, E)$, we can exactly recover H with high probability at cost $\widetilde{O}\left(|V|^{\frac{-1+\sqrt{9+4\varepsilon^2}}{2}-\varepsilon}|E| + |V|^{\frac{3+\sqrt{9+4\varepsilon^2}}{2}-\varepsilon}\right)$, where $\varepsilon = 3 - c$.

We note that, while our other results are fully algorithmic in addition to information theoretic (that is, the computational overhead of the algorithm is always upper bounded by the cost of its queries), this algorithm further requires listing many triangles in an explicitly-known graph, which requires no query cost but introduces an additional computational overhead not accounted for in this query cost. See Figure 1 for an illustration of these bounds in the case that $|E| = |V|^2$.

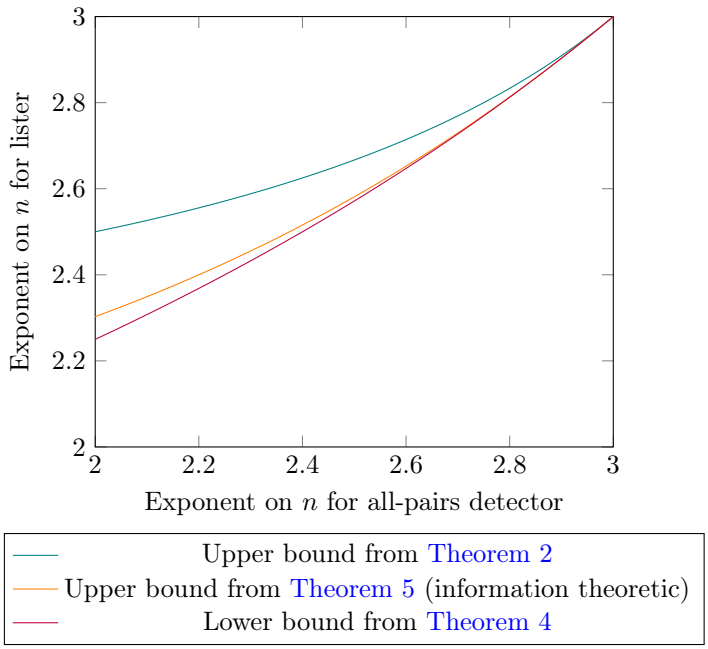

Figure 1. The cost of exactly recovering an n-vertex, n^2-edge, 3-uniform hypergraph, in terms of the cost of an all-pairs detetion oracle.

In Section 2, we discuss how these results relate to several open questions in fine-grained complexity, and speculate as to how to interpret the consequent barrier results.

1.3 Background and related work

There is a rich literature surrounding learning (hyper)graphs, or determining properties thereof, from restricted query models. Much of this focuses around independent set queries, which determine whether a given subset of vertices contains a hyperedge. There are lines of work on optimal parameters for learning graphs [Alo+04; AA05; AC05; CFS14; AN19] and hypergraphs [AC05; Dya+16; Aba18; ABM18; CFS18; BHW22; ART25] using these queries. Another particularly well-studied problem in the independent set query model is estimating the number of edges [Bea+20; CLW20; DL21] or hyperedges [Bha+19; DLM22; DLM24; CEV25]. The latter reference [CEV25] considers independent set queries with a non-constant cost. Note that in some of these works there is a separation between what can be done in bipartite graphs (or r-partite r-uniform hypergraphs) compared to general graphs.

Graph and hypergraph recovery problems have also been studied in a number of other global query models, including queries that count the number of edges [CK08; BM09; BM10], measure distances [MZ13], test connectivity [KLP24], or find maximal independent sets [KOT25].

2 Implications in fine-grained complexity

Recall our Question 1 from the introduction: can we get a better quantitative reduction than that of [WW10] from all-edge triangle detection to triangle detection? Our results on simulating all-pairs containment queries with independent set queries immediately rule out such a reduction in a "black-box" sense:

Corollary 1 (to Theorem 1)**.** All-edge triangle detection does not $(n^{3-\varepsilon}, n^{3-\delta})$-black box reduce to triangle detection (or even triangle counting) for any $\varepsilon > 0$, $\delta > \varepsilon/3$.

Deducing Corollary 1 from Theorem 1 is immediate: as noted, we can encode an arbitrary hypergraph in the relation R if all edge weights are distinct. Here, though, observe that it was important that our lower bound held against arbitrary all-pairs codegree queries, as opposed to just *induced* all-pairs codegree queries, as a black-box reduction is free to delete edges from the graph. Corollary 1 can be thought of as a barrier explaining why we've been unable to find improvements to [WW10]: while it may be the case that all-edge triangle detection and triangle detection have the same complexity, no better reduction can be constructed in a black-box manner. In the remainder of this section, we will discuss a few other stories with similar morals, which served as our motivation for studying these hypergraph query problems.

2.1 Reducing binary matrix multiplication to triangle counting

Similar to the question of quantitatively improving Vassilevska Williams and Williams's reduction from boolean matrix multiplication to triangle detection, an important open problem in fine grained complexity is whether fast algorithms for triangle *counting* generically imply fast algorithms for *integer* matrix multiplication of binary matrices.

Question 2: Is there a (combinatorial) subcubic equivalence between triangle counting and all-edges triangle counting, analogous to the one for triangle detection?

Again, in this case we know that both of these problems are solvable directly in $O(n^\omega) \ll O(n^3)$ time, so the crucial part of this question is whether there exists a "simple" such reduction, which would be interesting because we do not know "simple" subcubic algorithms for either problem. There has been much interest over the years in the answer to Question 2 — no reduction between these problems is known, but one might a priori imagine some approach in which one determines the counts of triangles in some specially (maybe adaptively) chosen subgraphs of the graph defined by the matrix and e.g. does some linear algebra to combine them into counts for every pair of vertices. Our results demonstrate that no reduction of that form is possible, at least without exploiting some graph-theoretic property that wouldn't hold for other relations R:

Corollary 2 (to Theorem 1)**.** All-edge triangle counting does not $(n^3, n^{3-\varepsilon})$-black box reduce to triangle counting for any $\varepsilon > 0$.

In our language, such a generic reduction would constitute an additive-query algorithm for all-pairs codegree in 3-uniform hypergraphs, which Theorem 1 shows is impossible. As with the previous example, our philosophy is that this as a barrier to fine-grained reductions in the same vein as e.g. relativization barriers in classical complexity theory. While a resolution of Question 2 may be possible, our "black-box" barrier identifies large classes of techniques which may have initially seemed promising but we now know are doomed to fail. We hope this can help hone future efforts to more productive paths, or at least offer some justification as to why past efforts have been fruitless.

2.2 Listing triangles via matrix multiplication

Another notable known relationship between triangle problems comes from a pair of reductions by Duraj, Kleiner, Polak, and Vassilevska Williams [Dur+20] between the all-edge triangle detection problem and the problem of listing m triangles (or all triangles if there are fewer than m) in an m-edge graph. Their reduction shows that these problems are equivalent up to polylog factors when one considers runtimes in terms of the number of edges. An examination of their reductions reveals that they hold in our "black-box" sense for any relation R:

Theorem 6 (Duraj, Kleiner, Polak, and Vassilevska Williams [Dur+20])**.** For any $1 \leq c \leq 1.5$, all-edge triangle detection (m^c, m^c)-black-box reduces to listing m triangles, and listing m triangles (m^c, m^c)-black-box reduces to all-edge triangle detection.

One might wonder whether a similar result can be shown for runtimes in terms of the number of vertices.

Question 3: Is the problem of listing n^2 triangles equivalent to all-edge detection for runtimes in terms of n?

In one direction, their argument applies directly:

Theorem 7 (Duraj, Kleiner, Polak, and Vassilevska Williams [Dur+20])**.** For any $2 \leq c \leq 3$, all-edge triangle detection (n^c, n^c)-black-box reduces to listing n^2 triangles.

However, in the other direction, such a translation is unclear. The reverse direction of this implication would be quite interesting, because for dense graphs often we *can* get fast algorithms for all-edge R-triangle detection using matrix products, so this might be an even more directly useful statement than Theorem 6. Our results give a quantitatively weaker version of a positive answer to Question 4:

Corollary 3 (to Theorem 2)**.** For any $2 \leq c \leq 3$, the problem of listing n^2 triangles $(n^c, n^{\left(\frac{9-2c}{4-c}\right)})$-black-box reduces to all-edge triangle detection.

Observe that, if our graph had at most n^2 triangles, we could list all of them using our exact recovery algorithm for 3-uniform hypergraphs. Corollary 3 then follows because we can first subsample the vertices of the graph so that approximately n^2 triangles survive, and then use that exact recovery algorithm. Our algorithm gives a subcubic reduction between these problems for any relation R, but fails to show a full equivalence as exists in the sparse case.

In fact, we demonstrate that a full equivalence between these problems, if true, will require new techniques:

Corollary 4 (to Theorem 4)**.** For any $2 \leq c \leq 3$ and any $\varepsilon > 0$, the problem of listing n^2 triangles does not $(n^c, n^{\left(\frac{9}{6-c}-\varepsilon\right)})$-black-box reduce to all-edge triangle detection.

2.3 Relating the complexity of "rotated" matrix products

Recall that many forms of matrix multiplication can be thought of as all-edge R-triangle detection problems for corresponding relations R. Although we think of these problems in tripartite graphs, most of the matrix products we care about give rise to relations R that are symmetric in the three parts I, J, and K. For instance, boolean matrix multiplication is equivalent to all-edge (unweighted) triangle detection, and APSP is equivalent to all-edge negative triangle detection, where $R = \{(w_{ij}, w_{jk}, w_{ki}) \mid w_{ij} + w_{jk} + w_{ki} < 0\}$. However, one can also consider asymmetric relations R, where the edge weights between e.g. parts I and J are treated differently than those between parts J and K. In these cases, the problem of detecting which edges *between I and J* belong to R-triangles may be a qualitatively quite different problem than detecting which edges *between J and K* belong to R triangles.

And there are in fact some examples of asymmetric R for which these problems are important. For instance, consider $R = \{w_{ij}, w_{jk}, w_{ki} \mid w_{jk} = w_{ki}\}$. In this case, the problem of all-edge R-triangle detection between parts I and J can be readily seen to be equivalent to computing the equality product of two $n \times n$ matrices (where the ***equality product*** of matrices A and B is the boolean matrix C whose (i, j)

entry is 1 if and only if there is some k such that $A[i,k] = B[k,j]$). On the other hand, it can be seen (by an appropriate binary search approach) that the problem of all-edge R-triangle detection between parts J and K is (up to log factors) at least as hard as the ***boolean-(min, +)-product*** — that is, the problem of computing $(\min,+)$-product between an $n \times n$ integer-valued matrix and an $n \times n$ $\{0,\infty\}$-valued matrix, which is important for some forms of the all-pairs shortest paths problem [Cha07]. Both equality product and boolean-$(\min,+)$-product are known to be solvable in time $\widetilde{O}(n^{(\omega+3)/2}) \leq O(n^{2.69})$, where $\omega < 2.371$ is the matrix multiplication constant [Alm+25], via essentially the same algorithmic techniques; however, neither problem is known to reduce to the other [Mat91; Cha07].

One might conjecture that, in fact, these "rotated" R-triangle detection problems are always equivalent:

Question 4: Does boolean-$(\min,+)$-product reduce to equality product? More generally, is all-IJ R-triangle detection equivalent to all-JK detection for every R?

Indeed, we know of no R for which we have reason to suspect different complexities. However no reduction of this form is known.

In Theorem 2 we prove (tight) upper bounds for recovery by *specified-parts* all-pairs containment oracles. This corresponds in this picture to a reduction from listing triangles to detection just between a single pair of parts:

Corollary 5 (to Theorem 2)**.** For any $2 \leq c \leq 3$, the problem of listing n^2 triangles $(n^c, n^{\left(\frac{9-2c}{4-c}\right)})$-black-box reduces to the problem of detecting whether each edge *between parts I and J* is involved in a triangle.

Along with Theorem 7, Corollary 5 implies the following:

Corollary 6 (to Theorem 2)**.** For any $2 \leq c \leq 3$, all-IJ detection $(n^c, n^{\left(\frac{9-2c}{4-c}\right)})$-black-box reduces to all-JK detection. So, an n^c-time algorithm for equality product yields an $n^{\left(\frac{9-2c}{4-c}\right)}$-time algorithm for boolean-$(\min,+)$-product.

Our Corollary 6 is the first nontrivial reduction between equality product and boolean-$(\min,+)$ product. It is not a full equivalence (in fact, our lower bound in Theorem 3 can be extended to rule out any black-box proof of such an equivalence), but it shows in particular that equality product in $O(n^{2.53})$ time would suffice to improve on the current best known algorithm for boolean-$(\min,+)$-product.

3 Relating counting and detection to their all-s versions

The all-s containment and all-s codegree queries we discuss in this paper can be viewed as direct strengthenings of two query models that have been extensively previously studied: *independent set queries*, which detect whether an induced subhypergraph contains at least one hyperedge, and *additive queries*, which count the number of edges in an induced subhypergraph. Our new models differ by additionally providing versions of this information for *every* s-tuple of vertices in the query. So, an all-s containment (resp. all-s codegree) oracle at cost n^c immediately yields an independent set (resp. additive) oracle at cost n^c. Similarly, additive (resp. all-s codegree) queries at cost n^c directly yield independent set (resp. all-s containment) queries at cost n^c, as we can just check which counts are 0. In this section, we investigate other relationships between these query models.

As a positive result, we observe that it *is* possible to nontrivially simulate all-s containment queries using an independent set oracle that's allowed to query *subhypergraphs* as opposed to just *induced subhypergraphs*. In other words, to decide if a given collection of r-tuples of vertices on a particular vertex set includes a hyperedge. This model has been called ***edge emptiness queries*** or ***OR queries*** in the literature [ACK21; BGM23]. We have the following (note that, as in the case of our all-s problems, we measure the cost of an oracle call in terms of the number of vertices it involves as opposed to the number of possible edges):

Lemma 1. Given edge emptiness queries at cost n^c to an unknown r-partite r-uniform hypergraph, we can compute the answer to an all-s containment query at cost $\widetilde{O}\left(n^{\left(s+\frac{c(r-s)}{r}\right)}\right)$.

Proof. We wish to solve the all-s containment problem on an n-vertex r-partite r-uniform hypergraph. It suffices to solve the specified-parts all-s containment problem (i.e. only report containment for vertices spanning the first s parts), as a generic algorithm for this could then be run $\binom{r}{s}$ times to determine values for every s-tuple of parts. We first arbitrarily partition each of the r parts into $n^{s/r}$ distinct chunks, each of size at most $n^{\left(\frac{r-s}{r}\right)}$. Then, we iterate through every r-tuple of those chunks, and do the following: Make an edge emptiness query to the subhypergraph induced by the union of those chunks. If the oracle reports that a hyperedge exists, then find such a hyperedge by binary search (by repeatedly throwing away half of the vertices in a given part and testing if a hyperedge still remains, we can find some hyperedge with $O(\log n)$ edge emptiness queries). Then, consider the s-tuple of vertices that the hyperedge involves among the first s parts. Record that this s-tuple *is* involved in some hyperedge, and subsequently exclude all sets containing those s vertices together from all queries made in the rest of the reduction (note that this is where we use that we are able to query any subhypergraph, not just induced subhypergraphs). Repeat this whole process again until the edge emptiness oracle returns that there are no more hyperedges between these chunks, at which point we progress to the next r-tuple of chunks.

To argue correctness, we note that for any s-tuple of vertices among the first s parts, if it is involved in a hyperedge, then some such hyperedge will be found: by the time we get to the first r-tuple of chunks involving such a hyperedge, the only hyperedges that we will be excluding from consideration in the induced subhypergraph will be those involving a different s-tuple of vertices among the first s parts. To argue cost, note that every time we attempt to find a hyperedge, we either successfully mark some previously-unmarked s-tuple as belonging to a hyperedge (which can happen at most n^s times), or we move on to the next r-tuple of chunks (which can happen at most $(n^{s/r})^r = n^s$ times). The cost of running that "attempt to find a hyperedge" procedure is at most $\log n$ many edge emptiness queries to a subhypergraph on at most $n^{\frac{r-s}{r}}$ vertices, and so has cost $O\left(\left(n^{\left(\frac{r-s}{r}\right)}\right)^c\right)$. Overall, this gives cost $\widetilde{O}\left(n^{\left(s+\frac{c(r-s)}{r}\right)}\right)$. ∎

In the case of $r = 3$, $s = 2$, note that this proof exactly corresponds to Vassilevska Williams and Williams's reduction from boolean matrix multiplication to triangle detection. The fact that their reduction is implementable in the hypergraph edge emptiness model suggests a limitation of their techniques for proving a stronger equivalence. Specifically, we can show information theoretically that the above is tight even given the *counting* version of edge emptiness queries. We start with the following observation:

Lemma 2. Given a cost-n^c all-s containment oracle, we can at cost $\widetilde{O}(n^c)$ exactly recover all hyperedges that involve some s-tuple with codegree at most $\mathsf{polylog}(n)$.

Proof. We first give an algorithm for recovering, for each s-tuple in a unique hyperedge, the hyperedge that the s-tuple is in, and then bootstrap this to recover all hyperedges involving an s-tuple with codegree at most $\mathsf{polylog}(n)$.

Write the hypergraph as $H = (V = X_1 \sqcup, \ldots, \sqcup X_r), E)$. We give an algorithm for recovering all such s-tuples in the specific tuple of parts $(X_1, \ldots, X_s)$ (then, to get an algorithm for recovering all such s-tuples, we can just run this algorithm on all possible s-tuples of parts). The idea is to do binary search on all the vertex parts to simultaneously find for every s-tuple x in exactly one hyperedge, the hyperedge that it is in. For each $i \in \{s+1, \ldots, r\}$, let $f_i : X_i \to [|X_i|]$ be an ordering on the vertices in X_i. Initialize a list L of size n^s where each entry is a tuple of $r - s$ empty strings. For every x in a unique hyperedge e, the ith string in $L[x]$ will contain $f_i(v_i)$ where v_i is the vertex in X_i in e.

For $j \in [\log n]$, we split each X_i into $X_{i,0}$ and $X_{i,1}$, where $X_{i,0}$ (resp. $X_{i,1}$) consists of all vertices $v_i \in X_i$ such that $f_i(v)[j]$ is 0 (resp. 1). For $k \in [2^{r-s}]$, we run $\mathcal{A}$ on $X_1 \sqcup \cdots \sqcup X_s \sqcup X_{s+1,k[1]} \sqcup \cdots \sqcup X_{r,k[r-s]}$ (where $k[\ell]$ is the ℓth bit of the $(r-s)$-length binary representation of k). For each $x \in V(s)$, if $\mathcal{A}$ outputs that x is in some hyperedge, then for each $\ell \in [r-s]$ we append $k[\ell]$ to $L[x][\ell]$. At the end, for every s-tuple x in a unique hyperedge, $L[x]$ contains the other vertices in the hyperedge with x. We return L. This algorithm runs in time $\widetilde{O}(n^c)$ since we make $\widetilde{O}(1)$ cost-n^c calls to the oracle.

Now we bootstrap this algorithm: to do this, the idea is to, in $K\mathsf{polylog}(n)$ iterations where K is some large enough constant, partition each of the vertex parts of H into equal-sized chunks of size $Cn/\mathsf{polylog}(n)$

for some large enough C, then run the algorithm previously described on all r-tuples of chunks, one from each part, and output all the hyperedges that the algorithm lists (in any iteration). Applying some concentration inequalities and choosing constants large enough, it can be readily seen that with high probability every s-tuple that has codegree at most $\mathsf{polylog}(n)$ is listed. The cost is $\widetilde{O}(n^c)$ since we make $\widetilde{O}(1)$ calls to the cost-n^c oracle.

■

We can now show the following:

Lemma 3. Fix some algorithm which makes queries to a cost-n^c oracle for counting the number of edges in subhypergraphs of an unknown r-partite r-uniform hypergraph. If this algorithm solves the all-s containment problem with high probability, then it requires cost $\widetilde{\Omega}\left(n^{\left(s+\frac{c(r-s)}{r}\right)}\right)$ for some hidden hypergraphs.

Proof. Suppose we had a more efficient reduction. By Lemma 2, this also gives us a $\widetilde{o}\left(n^{\left(s+\frac{c(r-s)}{r}\right)}\right)$ cost algorithm to exactly recover an r-uniform, r-partite hypergraph so long as each s-tuple belongs to at most $\mathsf{polylog}(n)$ distinct edges. We will show that this is impossible. By Yao's principle, it suffices to fix a distribution over hypergraphs such that with high probability each s-tuple belongs to at most $\mathsf{polylog}(n)$ hyperedges, and such that no deterministic algorithm with so small a query complexity can succeed with high probability over this distribution. The distribution we choose will simply place n vertices in each part and include every hyperedge with independent probability n^{s-r}. Observe that this distribution has entropy $n^r H(n^{s-r})$ — so, any algorithm solving the exact recovery problem with high probability must have an output distribution with mutual information at least $(1-o(1))n^r H(n^{s-r})$ with the hypergraph. By the data processing inequality, this implies that the distribution of all the results of its queries must have entropy at least $(1-o(1))n^r H(n^{s-r})$.

The goal now becomes bounding the total amount of information revealed by any given query. First, we pass to a relaxation of the oracle model: for any r-tuple of vertices, if at any point in the course of the algorithm's runtime the probability of that r-tuple being a hyperedge conditional on all previous queries exceeds $2n^{s-r}$, we will reveal for free whether it is involved in a hyperedge. We'll let random variable X_i denote the response to the ith query, and random variable Y_i denote the information revealed after the ith query from these threshold exceedences. Observe that this thresholding causes us to provide strictly more information to the algorithm, so if we can rule out algorithms in this query model we rule them out in the original model. Observe also that, since we expect at most $n^r/2$ threshold exceedences, the expected amount of information we reveal in response to them is insufficient to capture all of the entropy of the distribution:

$$\sum_i H(Y_i \mid X_1, \ldots, X_i, Y_1, \ldots, Y_{i-1}) \leq \frac{n^r}{2} \cdot H(2n^{s-r}) \leq n^r H(n^{s-r}) - \Omega(n^s).$$

By the chain rule for conditional entropy, this implies that we have

$$\sum_i H(X_i \mid X_1, \ldots, X_{i-1}, Y_1, \ldots, Y_{i-1}) \geq \Omega(n^s).$$

We now need to show that this is impossible for any query strategy that always has small query cost. Consider some run of the algorithm in which it made (possibly adaptively chosen) queries $(Q_1, R_1), \ldots, (Q_q, R_q)$ for $\sum_{i=1}^q |Q_i|^c < n^{\left(s+\frac{c(r-s)}{r}\right)} \cdot \log^{-2}(n)$, and got responses $x_1, \ldots, x_q$, with additional bonus information $y_1, \ldots, y_q$ revealed. We now bound $H(X_i \mid X_1 = x_1, \ldots, X_{i-1} = x_{i-1}, Y_1 = y_1, \ldots, Y_{i-1} = y_{i-1})$ in terms of $|Q_i|$. When $|Q_i| \geq n^{\left(\frac{r-s}{r}\right)}$, we use the trivial bound of $H(X_i) \leq O(\log n)$ (which holds because X_i is supported only on integers between 0 and n^{r-s}). When $|Q_i| < n^{\left(\frac{r-s}{r}\right)}$, we instead use subadditivity of entropy. Such a query can involve at most $|Q_i|^r$ distinct r-tuples of vertices that haven't yet been revealed to have be a hyperedge, and we know each of them has probability at most $2n^{s-r}$ conditional on our previous queries of being a hyperedge. So, the total conditional entropy of this query is at most $|Q_i|^r H(2n^{s-r}) < O(|Q_i|^r n^{s-r} \cdot \log(n))$.

Together, we can therefore bound (assuming small cost and appealing to convexity in the penultimate inequality):

$$\sum_i H(X_i \mid X_1 = x_1, \ldots, X_{i-1} = x_{i-1}, Y_1 = y_1, \ldots, Y_{i-1} = y_{i-1}) \leq$$

$$\sum_{i \atop |Q_i| > n^{\left(\frac{r-s}{r}\right)}} O(\log n) + \sum_{i \atop |Q_i| < n^{\left(\frac{r-s}{r}\right)}} O(|Q_i|^r n^{s-r} \cdot \log(n)) \leq$$

$$\frac{n^{\left(s+\frac{c(r-s)}{r}\right)} \cdot \log^{-2}(n)}{\left(n^{\left(\frac{r-s}{r}\right)}\right)^c} \cdot O(\log(n)) + \frac{n^{\left(s+\frac{c(r-s)}{r}\right)} \cdot \log^{-2}(n)}{\left(n^{\left(\frac{r-s}{r}\right)}\right)^c} \cdot O(\log(n)) \leq$$

$$O(n^s \cdot \log^{-1}(n)).$$

Since we've shown this sum of conditional entropies is $o(n^s)$ for every run with small query complexity, but $\Omega(n^s)$ in expectation, we know that the algorithm must incur query cost at least $n^{\left(s+\frac{c(r-s)}{r}\right)} \cdot \log^{-2}(n)$ on some instances. ■

Since the all-s codegree problem is strictly harder than the all-s containment problem, Lemma 3 also gives some lower bound on the cost of solving all-s codegree with additive queries. Our following lemma strengthens that lower bound:

Lemma 4. Fix some algorithm which makes queries to a cost-n^c oracle for counting the number of edges in subhypergraphs of an unknown r-partite r-uniform hypergraph. If this algorithm solves the all-s codegree problem with high probability, then it requires cost $\widetilde{\Omega}\left(n^{\min(r, s+c)}\right)$ for some hidden hypergraphs.

Observe that it is trivial to achieve cost $O(n^r)$ (simply iterate over every r-tuple of vertices and make a constant-cost query to check if it is a hyperedge), or cost $O(n^{s+c})$ (for every s-tuple of vertices, remove all other vertices sharing parts with them and use a single cost-n^c oracle query to determine the number of hyperedges involving those s vertices). This lemma demonstrates that nothing can substantially improve on those algorithms.

Proof of Lemma 4. We consider a strictly stronger model of query. For every $(r-s)$-tuple of vertices spanning the last $(r-s)$ parts of the hypergraph, let the associated "bucket" be the set of all r-tuples involving those $(r-s)$ vertices. Observe that any additive query (Q, R) can intersect at most $|Q|^{r-s}$ buckets, and will reveal a $O(\log(n))$-bit value (namely, a single count). We could instead imagine providing the algorithm with a stronger oracle: for any q, at cost q^c, the algorithm can evaluate *any* $O(\log(n))$-bit-valued function on each of *any* collection of q^{r-s} many buckets. This oracle model is at least as powerful, as the algorithm could choose a set of buckets corresponding to all $(r-s)$-tuples of the vertices in Q, learn for each of them the number of hyperedges shared with $\mathcal{P}(Q) \cap R$, and then add up all those values.

Now, consider a hidden r-uniform r-partite hypergraph chosen by including each hyperedge with independent probability $1/2$. The set of hyperedges in each bucket is an independent random variable with n^s bits of entropy. Now, consider some (possible adaptive) strategy of querying our strengthened oracle, and suppose that for some i the ith bucket is queried $\widetilde{o}(n^s)$ times in expectation. After any sequence of queries, each bucket's set of hyperedges is still independent from all other buckets. So, only the queries we make to this bucket yield any information about its contents. Since each query returns $O(\log n)$ bits per bucket, this means that the results of all of our queries have mutual information $o(n^s)$ with the ith bucket's hyperedges, meaning that the conditional entropy of the ith bucket's hyperedges is $\Omega(n^s)$ given the results of all queries. We claim that this implies that the algorithm must fail the all-s codegree problem with overwhelming probability. Indeed, suppose that following the algorithm's last query, we subsequently revealed the contents of every bucket other than the ith. Now, determining all-s codegrees entails exactly determining the set of hyperedges in the ith bucket, which remain distributed as before as all buckets are independent. As conditioning cannot increase conditional entropy, this means that the correct output to the all-s codegree problem

has conditional entropy $\Omega(n^s)$ given the results of all of the algorithm's queries. By Fano's inequality, any estimator for the output of the all-s codegree problem given those query results must fail with high probability.

Finally, we observe that, since a query of cost $x = q^c$ involves at most $q^{r-s} = x^{(r-s)/c}$ buckets, and queries must have sizes at least r and at most rn, by convexity any sequence of queries of cost $\widetilde{o}(n^{\min(r,s+c)})$ must query the average bucket at most $\widetilde{o}(n^s)$ times. So, an algorithm whose query cost is always this low must fail with high probability on the uniform distribution. ■

The final question we consider is whether all-s containment gives any use in simulating additive queries. It is again easy to see that, at cost $O(n^r)$, one can learn the entire hypergraph, just by querying every r-tuple of vertices individually. The following straightforward lower bound shows that no improvement to this is possible:

Lemma 5. Fix some algorithm which makes queries to a cost-n^c all-s containment oracle for an unknown r-partite r-uniform hypergraph, with $s < c < r$. If this algorithm returns the number of hyperedges in the hypergraph with high probability, then it requires cost $\Omega(n^r)$ for some hidden hypergraphs.

Proof. Again, we use Yao's principle: it suffices to define a distribution on which any deterministic algorithm with $o(n^r)$ query complexity is likely to err. The distribution we choose will with $1/2$ probability be the complete r-partite r-uniform hypergraph on n vertices, and with $1/2$ probability be that hypergraph with one random hyperedge removed. For a given query (Q, R), observe that any s-tuple of vertices in Q that belongs to at least two distinct sets of size r in R will necessarily return true, as at least one of those sets must be an edge. So, the query will only provide information distinguishing these two distributions if some s-tuple of vertices in Q belongs to a *unique* r-tuple in R, and that r-tuple happens to be the missing edge of the graph. There can be at most $\binom{|Q|}{s}$ many r-tuples tested this way by a single query, so since $\sum_i |Q_i|^s \leq \sum_i |Q_i|^c \leq o(n^r)$, an algorithm with $o(n^r)$ query cost will with high probability never make such a distinguishing query. ■

Lemma 1, Lemma 3, Lemma 4, and Lemma 5 together constitute our Theorem 1.

4 Upper bounds for exact recovery in the case $r = 3, s = 2$

In this section, we give algorithms for the problem of recovering a 3-uniform, 3-partite hypergraph given access to a cost-n^c induced all-pairs containment oracle, in the case when the oracle is a specified-parts oracle and in the case when the oracle is an all-parts oracle. These results are particularly relevant for the fine-grained complexity discussion. (In the language of F-triangles, we give black-box algorithms for the problem of listing F-triangles in a tripartite graph given access to a cost-n^c oracle for all-edge triangle detection.)

Here, we use $G = (V = I \cup J \cup K, E)$ for the 3-uniform, 3-partite graph and $\mathcal{A} = \mathcal{A}_G$ for the oracle, and when $\mathcal{A}$ is an oracle for a specific pair of parts, we take I and J to be these parts. We use n for $|V|$ as usual. We write $|E| = t = n^\tau$, where $0 \leq \tau \leq 3$. We write $c = 3 - \varepsilon$, for some $0 \leq \varepsilon \leq 1$, so that $\mathcal{A}$ is a cost-$n^{3-\varepsilon}$ oracle. To start, we give a few subroutines that both of our algorithms use.

The first subroutine uses the oracle to estimate for every pair of vertices in $I \times J$ the number of hyperedges that it is in.

Lemma 6. Given G and access to $\mathcal{A}$, we can compute in randomized $\widetilde{O}(n^{3-\varepsilon})$ time a list of estimates $a_{i,j}$ for every pair of vertices $(i, j) \in I \times J$, such that with high probability every pair (i, j) belongs to at least $\frac{a_{i,j}}{10}$ and at most $10a_{i,j}$ distinct hyperedges in G.

Proof. For each $\ell \in \{0, \ldots, \log n\}$, for $1000 \log n$ iterations, we sample a uniformly random subset $K' \subseteq K$, where $|K| = 2^\ell$, and call $\mathcal{A}$ on $I \cup J \cup K'$. For a particular pair $(i, j) \in I \times J$, let x be the number of distinct hyperedges in G that contain (i, j), and let y_ℓ be the fraction of these $1000 \log n$ iterations for which $\mathcal{A}$ detected a hyperedge containing (i, j). Observe that standard concentration inequalities imply the following:

- If $x < \frac{2^\ell}{10}$, then $\Pr\left[y_\ell < \frac{1}{4}\right] \geq 1 - \frac{1}{n^3}$.
- If $x > 10 \cdot 2^\ell$, then $\Pr\left[y_\ell > \frac{3}{4}\right] \geq 1 - \frac{1}{n^3}$.
- If $2^{\ell-1} \leq x \leq 2^\ell$, then $\Pr\left[\frac{1}{4} \leq y_\ell \leq \frac{3}{4}\right] \geq 1 - \frac{1}{n^3}$.

So, our algorithm can set $a_{i,j}$ to be 2^ℓ, where ℓ is the smallest value such that $1/4 \leq y_\ell \leq 3/4$. Our guarantees ensure that with high probability such an $a_{i,j}$ will exist for every pair in $I \times J$, and will be within a factor of 10 of the true number of hyperedges containing (i, j). ■

The next subroutine uses the oracle to list, for every pair of vertices in $I \times J$ that is in a *unique* hyperedge, the hyperedge that it is in.

Lemma 7. Given G and access to $\mathcal{A}$, there exists an $\widetilde{O}(n^{3-\varepsilon})$-time algorithm that, for every $(i, j) \in I \times J$ involved in *exactly* one hyperedge in G, lists that hyperedge.

Proof. Let $g : K \to [|K|]$ be an ordering on the vertices in K. We run a binary search on K simultaneously for each $(i, j) \in I \times J$ to find a vertex k such that $(i, j, k) \in E$ (if one exists).

We initialize an array L of empty strings, one for each $(i, j) \in I \times J$. For $\ell = 1, \ldots, \log n$, set $K' \subseteq K$ to consist of the vertices $k \in K$ such that the ℓth bit of $g(k)$ is 0. We call $\mathcal{A}$ on $I \cup J \cup K'$. For each (i, j), if $\mathcal{A}$ returns that (i, j) is in some hyperedge, we append 0 to $L[(i, j)]$; otherwise, we append 1. Upon exiting the outer for-loop, for each (i, j), we check whether the vertex k such that $g(k) = L[(i, j)]$ is in a hyperedge with i and j, and if k is, we output (i, j, k).

As we call $\mathcal{A}$ $O(\log(n))$ times on instances of size $O(n)$, this algorithm runs in time $\widetilde{O}(n^{3-\varepsilon})$.

■

Finally, we give a subroutine that bootstraps the algorithm described in Lemma 7 to list, for every pair of vertices $(i, j) \in I \times J$ that's in not-too-many hyperedges, all the hyperedges that it is in.

Lemma 8. Given G, access to $\mathcal{A}$, and some parameter $\alpha > 0$, there exists a randomized $\widetilde{O}(n^{3-\varepsilon+\varepsilon\alpha})$-time algorithm such that the following guarantee holds with high probability: for pair of vertices $(i, j) \in I \times J$ involved in at most n^α distinct hyperedges in G, the algorithm lists all hyperedges involving (i, j).

Proof. Call a pair of vertices $(i, j) \in I \times J$ ***unpopular*** if it is involved in at most n^α distinct hyperedges. Note that it suffices to give an $\widetilde{O}(n^{3-\varepsilon+\varepsilon\alpha})$-time algorithm with the guarantee that every hyperedge involving an unpopular pair is listed with constant probability. This is because repeating $O(\log n)$ independent iterations of that algorithm will w.h.p. list all such hyperedges.

The procedure is to partition each of I, J, and K into random subsets $I_1, \ldots, I_{n^\alpha}$, $J_1, \ldots, J_{n^\alpha}$, and $K_1, \ldots, K_{n^\alpha}$, each of size $n^{1-\alpha}$, and for every triple of $q, r, s \in [n^\alpha]$, we run the algorithm of Lemma 7 on $G[I_q, J_r, K_s]$.

To verify this works, fix some particular hyperedge (i, j, k) with (i, j) unpopular, and let $I^* \in \{I_1, \ldots, I_{n^\alpha}\}$, $J^* \in \{J_1, \ldots, J_{n^\alpha}\}$, $K^* \in \{K_1, \ldots, K_{n^\alpha}\}$ be the subsets in the random partition that i, j, and k are mapped to, respectively. Since (i, j) is unpopular, there are at most $10n^\alpha$ distinct values $k' \neq k$ such that (i, j, k') is a hyperedge. So, the probability that any such k' is assigned to K^* is at most $1 - \left(1 - \frac{1}{n^\alpha}\right)^{10n^\alpha} \leq 1 - e^{-20}$. As long as this doesn't happen, (i, j) will be involved in a unique hyperedge in $G[I^*, J^*, K^*]$, and so the algorithm will list (i, j, k).

Now, to calculate cost, observe that we run the algorithm of Lemma 7 on $(n^\alpha)^3$ instances, each on $n^{1-\alpha}$ vertices. So, the overall runtime is $\widetilde{O}\left((n^\alpha)^3 \cdot \left(n^{1-\alpha}\right)^{3-\varepsilon}\right) = \widetilde{O}\left(n^{3-\varepsilon+\varepsilon\alpha}\right)$.

■

With these subroutines, our specified-parts oracle algorithm is pretty straightforward.

Theorem 8. Given cost-n^c specified-parts induced all-pairs containment oracle access to an unknown 3-partite, 3-uniform hypergraph $H = (V, E)$, we can exactly recover H with high probability at cost $\widetilde{O}\left(|V|^{3(1-\gamma)}|E|^{\gamma}\right)$, where $\gamma = \frac{3-c}{4-c}$.

Proof of Theorem 8. Let $G = (V = I \cup J \cup K, E)$, where $|E| = n^\tau$, be an unknown, 3-partite, 3-uniform hypergraph, and let $\mathcal{A} = \mathcal{A}_G$ be a cost-n^c induced specified-parts all-2 containment oracle, where $c = 3 - \varepsilon$. First, we call the algorithm given in Lemma 6 on G to obtain an estimate $a_{i,j}$ for each $(i, j) \in I \times J$ of the number of hyperedges that (i, j) is in. Set $\alpha = (\tau + \varepsilon - 2)/(1 + \varepsilon)$, and call a pair of vertices $(i, j) \in I \times J$ ***popular*** if $a_{i,j} \geq n^\alpha$, and ***unpopular*** otherwise. The idea is to list the hyperedges involving popular pairs and unpopular pairs separately. First, for each popular pair (i, j), we iterate through all $k \in K$ and make a constant-cost call to $\mathcal{A}$ to check whether (i, j, k) is a hyperedge. Then we call the algorithm described in Lemma 8 on G and α to list all hyperedges involving an unpopular pair.

Since each popular pair is in at least n^α hyperedges and there are n^τ hyperedges total, there are at most $n^{\tau-\alpha}$ popular pairs. So listing the hyperedges involving a popular pair takes time $O(n^{\tau-\alpha+1})$. By Lemma 8, listing all the hyperedges involving an unpopular pair takes time $\widetilde{O}(n^{3-\varepsilon+\varepsilon\alpha})$. (Additionally, estimating the number of hyperedges that each pair is in is time $\widetilde{O}(n^{3-\varepsilon})$ by Lemma 6, but since $\alpha > 0$ this cost is dominated by the cost of Lemma 8.) So the total runtime of the algorithm is $\widetilde{O}(n^{\tau-\alpha+1} + n^{3-\varepsilon+\varepsilon\alpha}) = \widetilde{O}(n^{3-\varepsilon(3-\tau)/(1+\varepsilon)})$. ∎

Now we give our all-parts oracle algorithm. In this case, we will once again identify some collection of "popular" pairs, which are involved in many hyperedges. But now, we will do some additional processing to ensure that these popular pairs are not too highly clustered. That is, while the graph on popular pairs (which is an explicit graph we have access to from our estimates, as opposed to being queried via the oracle) has too many triangles, we will process all of the pairs involved in triangles of popular pairs with a single particular vertex, and then remove those popular pairs from consideration in all later queries. This introduces two downsides over Theorem 8: one is that, although our query cost is low, we introduce substantial computational overhead from finding triangles in the graph of popular pairs. The other is that, since we must remove some popular pairs from consideration, this algorithm needs the general (i.e. not-necessarily-induced) version of all-pairs containment queries. The statement is as follows:

Theorem 5. Given cost-n^c specified-parts induced all-pairs containment oracle access to an unknown 3-partite, 3-uniform hypergraph $H = (V, E)$, we can exactly recover H with high probability at cost $\widetilde{O}\left(|V|^{\frac{-1+\sqrt{9+4\varepsilon^2}}{2}-\varepsilon}|E| + |V|^{\frac{3+\sqrt{9+4\varepsilon^2}}{2}-\varepsilon}\right)$, where $\varepsilon = 3 - c$.

Proof of Theorem 5. Again, let $G = (V = I \cup J \cup K, E)$, where $|E| = t = n^\tau$, be an unknown, 3-partite, 3-uniform hypergraph, and let $\mathcal{A} = \mathcal{A}_G$ be a cost-n^c induced specified-parts all-2 containment oracle, where $c = 3 - \varepsilon$. We first call the algorithm given in Lemma 6 on G, this time for all pairs of parts, to obtain an estimate $a_{u,v}$ for every $(u, v) \in V^2$. Set parameters $\alpha = (x + \tau - 3 + \varepsilon)/\varepsilon$ if $2 \leq \tau \leq 3$ and $\alpha = (x - 1 + \varepsilon)/\varepsilon$ if $0 \leq \tau \leq 2$, and $\beta = x + \tau - 1$ if $2 \leq \tau \leq 3$ and $\beta = x + 1$ if $0 \leq \tau \leq 2$, where $x = -\varepsilon + (-1 + \sqrt{9 + 4\varepsilon^2})/2$. Call a pair of vertices (u, v) ***popular*** if $a_{u,v} \geq n^\alpha$ and ***unpopular*** otherwise. We call the algorithm from Lemma 8 on G and α to list all hyperedges that involve an unpopular pair. So from this point on we can think of the graph as having only popular pairs of vertices.

Note that now, since every pair of vertices is popular, there can be at most $n^{\tau-\alpha}$ pairs of vertices in each of $I \times J$, $J \times K$, and $I \times K$, since otherwise we would have $|E| > n^\tau$. We iterate through all pairs of vertices and count for each vertex the number of pairs that it is in. Call a vertex ***fashionable*** if it is in at least $1000n^{\tau-1-\alpha}$ popular pairs and ***unfashionable*** otherwise. Note that I can contain at most $n/10$ fashionable vertices, since otherwise there would be more than $n/10 \cdot 1000n^{\tau-1-\alpha} = 100n^{\tau-\alpha}$ popular pairs of vertices in either $I \times J$ or $I \times K$. Let $I^* \subseteq I$ be the set of fashionable vertices in I, and break J into two halves J_0 and J_1 and K into two halves K_0 and K_1. We recurse on $G[I^* \cup J_b \cup K_{b'}]$ for every $b', b' \in \{0, 1\}$ to list all hyperedges involving an fashionable vertex i. So now all that remains is to list the hyperedges that contain an unfashionable vertex in I.

To do this, we construct an auxiliary tripartite graph $G' = (V', E')$ (an ordinary graph) on vertex parts $I' = I \backslash I^*$, $J' = J$, and $K' = K$. The edges in G' are exactly the popular pairs of vertices in G in $(I' \times J') \cup (I' \times K') \cup (J' \times K')$. Note that every hyperedge not yet listed in G is a triangle in G'. The plan now is to find all hyperedges that contain some $i \in I'$ that is in many triangles in G'; then, separately find the hyperedges containing i in not-so-many triangles.

Now we check whether there is an $i \in I'$ in at least n^β triangles. If so, we partition I' into parts $I'_1, \ldots, I'_{n^{2+\alpha-\tau}}$, each of size $n^{\tau-1-\alpha}$. While there is an $i \in I'$ in at least n^β triangles, we recurse on $G[I'_\ell \cup N(i)]$ for every $\ell \in [n^{2+\alpha-\tau}]$ (where we use $N(i) = N_{G'}(i)$ for the neighborhood of i in G') to list all the hyperedges in G that involve pairs of vertices that are edges in $G'[N(i)]$. Note that this includes all the hyperedges that involve vertex i. Then we delete the edges in $G'[N(i)]$ from G', and then check whether there is still an $i \in I'$ in at least n^β triangles. After exiting this while-loop, to list the remaining hyperedges, we iterate through all triangles (i, j, k) in G' that contain an i in at most n^β triangles and make a constant-cost call to $\mathcal{A}$ to check whether (i, j, k) is a hyperedge in G.

Then, the (worst-case) runtime of this algorithm, not considering computational costs, is given by the recurrence

$$T(n,t) \le n^{2-\beta} T\left(\frac{t}{n^{1+\alpha}}, \frac{tn^\beta}{n^2}\right) + 4T\left(\frac{n}{2}, \frac{t}{4}\right) + \widetilde{O}(n^{3-\varepsilon+\varepsilon\alpha}) + n^{1+\beta},$$

where, going step by step through the algorithm, we can account for the terms as follows:

- estimating the number of hyperedges that each pair of vertices is involved in and finding all hyperedges involving unpopular pairs takes time $\widetilde{O}(n^{3-\varepsilon+\varepsilon\alpha})$;
- recursing to find all hyperedges that involve a fashionable vertex takes time $\sum_{j=1}^4 T(n/2, t_j)$, where $\sum_j t_j$ is at most t; applying a convexity argument (Jensen's inequality), we get that $\sum_{j=1}^4 T(n/2, t_j)$ is maximized when the t_j's are all the same, i.e., $t_j = t/4$ for all j;
- the recursive calls made in the while-loop take time $\sum_{\ell_1}^K n/(t/n^{1+\alpha}) \cdot T(t/n^{1+\alpha}, t_{\ell_1,\ell_2})$, where K is the number of iterations and for each $\ell_1 \in [K]$, $\ell_2 \in [n/(t/n^{1+\alpha})]$, t_{ℓ_1,ℓ_2} is the number of triangles in the subgraph of G induced by $N(i_{\ell_1}) \cup I'_{\ell_2}$, where i_{ℓ_1} is the vertex being considered in the ℓ_1th iteration of the loop. We have that $K \le t/n^{\alpha+\beta}$, since G' initially contains at most t/n^α edges between parts J' and K', and in each iteration we remove n^β of these edges (and once all these edges are removed we have to have exited the loop). By applying a convexity argument, we can upper bound the running time of this step by this sum where we take K to be the maximum possible and set all t_{ℓ_1,ℓ_2} equal, which implies $t_{\ell_1,\ell_2} = tn^\beta/n^2$. This comes out to be $t/n^{\alpha+\beta} \cdot n/(t/n^{1-\alpha}) T(t/n^{1+\alpha}, tn^\beta/n^2) = n^{2-\beta} T(t/n^{1+\alpha}, tn^\beta/n^2)$; and
- finally, listing $n^{1+\beta}$ triangles in G' at the end requires time at least $n^{1+\beta}$.

Now we inductively show that this recurrence satisfies $T(n,t) \le K(tn^x + n^{2+x})$, where $x = -\varepsilon + (-1 + \sqrt{9+4\varepsilon^2})/2$ and K is some sufficiently large constant. Assume the inductive hypothesis that $T(n', t') \le K(t'n'^x + n'^{2+x})$ for all $t' < t$, $n' < n$. It suffices to show that each of the four summands (where we ignore polylogarithmic factors by taking K large enough) are at most $(K/4)(tn^x + n^{2+x})$. Applying the inductive hypothesis, we get that $4T(n/2, t/4) \le K((t/4)(n/2)^x + (n/4)^{2+x})$ and $n^{2-\beta} T(t/n^{1+\alpha}, tn^\beta/n^2) \le K(n^{2-\beta}(tn^{\beta-2}(t/n^{1+\alpha})^x + (t/n^{1+\alpha})^{2+x}))$. It is clear that the right-hand side of the first inequality is at most $(K/4)(tn^x + n^{2+x})$ when K is large enough. To see that the other terms are also at most $(K/4)(tn^x + n^{2+x})$, it suffices to show that our parameters satisfy the following systems of inequalities (which come from looking at the exponents in these inequalities):

- if $2 \le \tau \le 3$:
  - $\beta + 1 \le x + \tau$
  - $3 - \varepsilon + \varepsilon\alpha \le x + \tau$
  - $2 - \beta + \max(x(\tau - 1 - \alpha) + \tau - 2 + \beta, (2+x)(\tau - 1 - \alpha)) \le x + \tau$
- and otherwise, if $0 \le \tau \le 2$:

- $\beta + 1 \leq 2 + x$
- $3 - \varepsilon + \varepsilon\alpha \leq 2 + x$
- $2 - \beta + (2 + x)(\tau - 1 - \alpha) \leq 2 + x$

As chosen, α and β satisfy the first two constraints in both systems. So all that remains is to check that the last constraint in each system is satisfied. In the first system, first consider the case where $x(\tau - 1 - \alpha) + \tau - 2 + \beta \geq (2 + x)(\tau - 1 - \alpha)$. Then the constraint is $x(\tau - 2 - \alpha) \leq 0$, which, substituting, yields $\tau(1-\varepsilon) > 3 - 3\varepsilon - x$. Since $\tau \geq 2$, it suffices to show that $2(1-\varepsilon) > 3 - 3\varepsilon - x$, which simplifies to $\varepsilon > 1 - x$, which holds. Next we consider the case where $x(\tau - 1 - \alpha) + \tau - 2 + \beta \leq (2 + x)(\tau - 1 - \alpha)$. Then the constraint becomes $2 - \beta + (2+x)(\tau - 1 - \alpha) \leq x + \tau$, which, substituting, yields $(\tau - 3)x - \alpha(2+x) + 1 \leq 0$. With some algebra, it can be seen that this holds for for $\tau \geq 2$, $\varepsilon > 0$. Finally, the left-hand side of the last constraint in the second system is at most $2 - \beta + (2+x)(1-\alpha) \leq 2 + x$, taking $\tau = 2$, so it suffices to show this is at most the right hand side. This inequality yields $(x - 1 - +\varepsilon)/\varepsilon \geq (x-1)/(2+x)$, which holds since we have $x - 1 + \varepsilon \geq x - 1$ and $\varepsilon \leq 2 + x$. ■

5 Upper bounds for exact recovery in general

In this section, we generalize Theorem 8 to hypergraphs of higher uniformity. Our main result is as follows:

Theorem 2. Given cost-n^c specified-parts induced all-s containment oracle access to an unknown r-partite, r-uniform hypergraph $H = (V, E)$, we can exactly recover H with high probability at cost $\widetilde{O}\left(|V|^{r(1-\gamma)}|E|^{\gamma}\right)$, where $\gamma = \frac{r-c}{2r-s-c}$.

In Section 6, we show that this upper bound is tight (up to polylog factors) for any $s \leq c \leq r$ and any $|V| \leq |E| \leq |V|^r$. The proof of Theorem 2 is similar to that of Theorem 8, but there are several complications that arise in these other uniformities. For one, it is no longer as immediate that we can use our all-s containment oracle to approximate all-s codegrees (although this is in fact possible). Another somewhat more serious issue is that it is no longer the case that we can expect to "isolate" hyperedges for unpopular s-tuples by randomly partitioning all of the parts. When we had $r - s = 1$, for a fixed s-tuple of vertices we knew that the set of hyperedges containing those vertices simply corresponded to a subset of the vertices in the remaining part. But now fixing an s-tuple of vertices can induce some higher-order structures which can result in many hyperedges falling into the same collection of buckets. In order to deal with this, we argue that the only cases in which such collisions occur can be attributed to collections of vertices with badly-behaved codegrees, and that we can efficiently identify all such collections and handle them by brute force.

We start with the following important subroutine.

Lemma 9. For k-uniform, k-partite, n-vertex hypergraphs, for any positive integer $h \leq n$, there exists an algorithm in the independent set query model that with high probability returns a family $\mathcal{F}$ of vertex subsets such that

- every element $S \in \mathcal{F}$ has codegree at least $h^{k-|S|} \log^{-10k}(n)$, and
- every hyperedge is a superset of some element $S \in \mathcal{F}$.

The algorithm makes $\widetilde{O}\left(h^k\right)$ non-adaptive independent set queries of size at most n/h.

Proof. Partition each vertex part randomly into h chunks of size n/h. For every tuple of k of those chunks, we would like to determine the identity of some hyperedge contained in that k-tuple of chunks (if any such hyperedge exists). If the k-tuple contains a unique hyperedge, this is straightforward: we can simply binary search (that is, for all $i \in [k]$, $j \in [\log(n)]$, $b \in \{0, 1\}$, discard all vertices in the ith vertex part with a b in the jth binary digit of their name, and make an independent set query on the rest of the vertices in the chunks — this procedure is non-adaptive, and if there is a unique hyperedge one can read off the identities of all vertices in the hyperedge from the result). In order to deal with k-tuples of chunks that contain multiple hyperedges, we can run the same procedure, but first subsample the vertices in each of the parts. That is,

for every $r_1, \ldots, r_k \in [\log(n)]$, for $\log^{10}(n)$ iterations, we choose a random sample of 2^{-r_i} vertices from the ith chunk in the k-tuple for all i, and we run the above procedure to find a unique hyperedge if one exists. Observe that there is some setting of all the r_i such that a unique hyperedge will survive with constant probability — so, with high probability, for all k-tuples containing at least one hyperedge we will find one on some iteration of this procedure.

Now that we've found this list of representative hyperedges, for every subset of vertices in the hypergraph, we'll record their codegree *among these representative hyperedges*. If we find that some subset S has codegree at least $h^{k-|S|}\log^{-10k}(n)$ among these identified edges, we know that it has codegree at least that high in the full hypergraph, and so can safely add it to $\mathcal{F}$. We would now like to show that, for any hyperedge E of the hypergraph, with probability at least $1/\mathsf{polylog}(n)$ this process will result in some subset of E being added to $\mathcal{F}$. Note that this is sufficient for the overall lemma: if we then repeat the entire process some large $\mathsf{polylog}(n)$ many times, we can ensure that the probability of any given hyperedge failing to have a subset added to $\mathcal{F}$ is much less than n^{-k}, meaning that we can union bound over all of these events.

Say that a pair of hyperedges E and E' ***collide*** if they both belong to the same k-tuple of chunks. If the probability of E colliding with another edge is less than $1/2$, then we are already done: with at least $1/2$ probability, E will be the unique hyperedge in its k-tuple of chunks, meaning that it will be necessarily chosen as the representative. Any hyperedge found as one of the representatives has codegree $1 \gg h^{k-k}\log^{-10k}(n)$ among the representatives, so will be itself added to $\mathcal{F}$. Otherwise, say that a subset $U \subseteq E$ is ***hot*** if, with probability at least $\log^{10|U|-10k}(n)$, there will be a collision between E and some other edge E' with $E \cap E' = U$. Observe that by union bound at least one subset of E must be hot. Fix U to be some hot subset of E of minimal size. We will show that, with probability at least $1/\mathsf{polylog}(n)$, this process adds U to $\mathcal{F}$.

First observe that, for an edge E' overlapping with E exactly on U, the $k - |U|$ chunks the remaining vertices of E' map to are independent of those that the remaining vertices of E map to. So, since the probability that there exists some such E' colliding with E is at least $\log^{10|U|-10k}(n)$, we can also observe that with high probability the number of k-tuples of chunks that contain *some* edge containing U will be at least $O(h^{k-|U|}\log^{10|U|-10k}(n))$. Call these k-tuples of chunks the ***useful*** k-tuples.

Fix any proper subset $W \subseteq U$, and consider the expected number of useful k-tuples that contain some hyperedge E' such that $E' \cap U = W$. The probability that E' shares chunks with all of the vertices in U is $h^{|W|-|U|}$. So, if this expectation was greater than $\left(h^{|W|-|U|}\right)\left(h^{k-|W|}\log^{10|U|-10k}(n)\right) = h^{k-|U|}\log^{10|W|-10k}(n)$, then we would have that W is hot, contradicting the assumption that U has the minimum size among hot subsets of E. Thus, the expected number of useful k-tuples that contain *any* hyperedge E' with $E' \cap U \neq U$ is at most

$$\sum_{\substack{W \subseteq U \\ W \neq U}} h^{k-|U|}\log^{10|W|-10k}(n) \leq 2^{|U|}h^{k-|U|}\log^{10|U|-10k-10}(n) \ll h^{k-|U|}\log^{10|U|-10k}(n).$$

So, with high probability, there are at least $O(h^{k-|U|}\log^{10|U|-10k}(n)) \gg h^{k-|U|}\log^{-10k}(n)$ distinct k-tuples of chunks that contain some hyperedge containing U, and no hyperedge not containing U. For each of those chunks, our process will extract a representative hyperedge containing U. So, U will be added to $\mathcal{F}$. ■

We are now ready to show the main result.

Proof of Theorem 2. We will set $h = |V|^{\frac{r-s-c}{2r-s-c}}|E|^{\frac{1}{2r-s-c}}$, and partition each of the specified parts arbitrarily into h chunks. For each s-tuple of those chunks, we will do the following. Observe that, if we look at the output of our all-s containment oracle for one specific s-tuple of vertices, it is exactly an independent set oracle on the $(r-s)$-uniform hypergraph induced by the restriction. So, setting $k = r - s$, we can run the algorithm from Lemma 9. That is, for every s-tuple of chunks among the specified parts, we run the algorithm from Lemma 9 on the remaining $r - s$ parts of the graph. Every time the algorithm would make

an independent set query on a vertex subset of those $r-s$ parts, we instead make a query to the all-s containment oracle, including that subset along with the chosen chunks from the remaining s parts. Since the queries are made non-adaptively, this process is well-defined, and since all queries are made to vertex subsets of size at most $O(|V|/h)$, the total cost is $h^s \cdot \widetilde{O}\left(h^k\right) \cdot O(|V|/h)^c = \widetilde{O}\left(h^{r-c}|V|^c\right) = \widetilde{O}\left(|V|^{r(1-\gamma)}|E|^\gamma\right)$.

Now, for every s-tuple X of vertices among the specified parts, we have determined a family $\mathcal{F}_X$ such that every hyperedge containing X also contains some element of $\mathcal{F}_X$, and every element $S \in \mathcal{F}_X$ satisfies that the codegree of $S \cup X$ is at least $\widetilde{\Omega}(h^{r-s-|S|})$. For every s-tuple X among the first s parts, for every $S \in \mathcal{F}_X$, our algorithm will now brute force over all $|V|^{r-s-|S|}$ many $(r-s-|S|)$-tuples Y of vertices in the remaining parts, and make a constant-size query to the oracle to check whether $X \cup S \cup Y$ is a hyperedge. Observe that, since every hyperedge containing X also contains some $S \in \mathcal{F}_X$, this procedure will find all of the hyperedges of the hypergraph. To determine the cost of this procedure, note that every time we process an $S \in \mathcal{F}_X$, we expend cost $O(|V|^{r-s-|S|})$, and we discover at least $\widetilde{\Omega}(h^{r-s-|S|})$ new hyperedges we had not previously found. So, the total cost of this step is at most

$$\max_{0 \le \ell \le r-s} \left(\frac{|E|}{\widetilde{\Omega}(h^{r-s-\ell})} \cdot O\left(|V|^{r-s-\ell}\right) \right) = \widetilde{O}\left(|E| \cdot \left(\frac{|V|}{h}\right)^{r-s} \right) = \widetilde{O}\left(|V|^{r(1-\gamma)}|E|^\gamma\right).$$

■

6 Lower bounds for exact recovery

In Section 5, we gave an algorithm for solving exact recovery of sparse hypergraphs using induced all-s containment queries. Here, we demonstrate that upper bound to be tight even with specified-parts all-s *codegree* queries. Our proof will make use of the following simple combinatorial fact:

Lemma 10. Let K be the random s-partite, s-uniform hypergraph with n vertices per part, obtained by including each hyperedge with independent probability $n^{-\beta}$. Then, with high probability there does not exist a subhypergraph on x vertices and more than $10x^s n^{-\beta} \log n + 10x \log n$ hyperedges for any x.

Proof. For a given set of x vertices, the number of induced hyperedges is distributed as $\mathrm{Bin}(x^s, n^{-\beta})$. By a standard Chernoff bound, we have

$$\Pr[\mathrm{Bin}(x^s, n^{-\beta}) \ge 10x^s n^{-\beta} \log n + 10x \log(n)] \le 2^{-3x^s n^{-\beta} \log(n) - 3x \log(n)} \le n^{-3x-2}.$$

So, union bounding over all choices of x and all n^x sets of vertices, the probability of any such dense subhypergraph existing is at most $\sum_{x=1}^{n} n^x \cdot n^{-3x-2} < o(1)$. ■

Theorem 3. For any values of $s \le c < r$, and any sufficiently large values of $|V|$ and $|E| < \frac{1}{2}\binom{|V|/r}{r}$, any algorithm that exactly recovers unknown r-partite, r-uniform hypergraphs $H = (V, E)$ with high probability using cost-n^c specified-parts all-s codegree queries must incur cost $\widetilde{\Omega}\left(|V|^{r(1-\gamma)}|E|^\gamma\right)$ on some instances, where $\gamma = \max\left(\frac{r-c}{2r-s-c}, \frac{r-c}{r-1}\right)$.

Proof. Once again, we will use Yao's lemma. We would like to design a distribution over $r|V|$-vertex, $|E|$-edge, r-partite, r-uniform hypergraphs, such that any deterministic algorithm is unlikely to successfully recover the hypergraph from specified-parts all-s codegree queries. To construct such a distribution, we will start by declaring each s-tuple of vertices spanning the first s parts of the hypergraph as "admissible" with independent probability p, for $p = (|V|^{-r}|E|)^{\max\left(\frac{r-c}{2r-s-c}, \frac{s-1}{r-1}\right)}$. This choice is fixed and revealed to the algorithm, as opposed to being a part of the distribution — the only property we require of those s-tuples is that guaranteed by Lemma 10 (so in particular, any explicit construction of such a quasirandom s-uniform hypergraph would also suffice to define the admissible s-tuples). Now, our distribution on hypergraphs will simply be to let each of the spanning r-tuples of vertices whose restriction to the first s parts is an admissible

s-tuple be a hyperedge with probability $|E||V|^{-r}p^{-1}$ independently[1].

The analysis at this point will proceed similarly to the proof of Lemma 3. We observe that the distribution we've defined over hypergraphs has entropy $|V|^r pH(|E||V|^{-r}p^{-1})$. We set a threshold of $t = 2|E||V|^{-r}p^{-1}$, and agree to reveal whether or not any given r-tuple is a hyperedge once its probability conditioned on our queries thus far of being so exceeds t. Letting X_i be the results of the ith query and Y_i be the additional information revealed about threshold-exceeding r-tuples after the ith query, as in Lemma 3 correctness of the algorithm implies that we have

$$\sum_i H(X_i|X_1, \ldots, X_{i-1}, Y_1, \ldots, Y_{i-1}) \geq \Omega(|E|).$$

Now, for a sequence of queries $(Q_1, R_1), \ldots, (Q_q, R_q)$, and the resulting responses $X_i = x_i$ and $Y_i = y_i$, we upper bound

$$\sum_i H(X_i|X_1 = x_1, \ldots, X_{i-1} = x_{i-1}, Y_1 = y_1, \ldots, Y_{i-1} = y_{i-1})$$

under the assumption that

$$\sum_i |Q_i|^c < \widetilde{o}\left(|V|^{r(1-\gamma)}|E|^\gamma\right).$$

Consider a query of size $|Q_i|$. Since each query is made to a subhypergraph of the original hypergraph, by Lemma 10 we know that Q_i can involve at most $10|Q_i|^s p \log(|V|) + 10|Q_i| \log(|V|)$ admissible s-tuples. We already know that no non-admissible s-tuple is involved in a hyperedge, so the query will reveal information only for these tuples. For each admissible s-tuple, at most $\log(|E|) = O(\log(|V|))$ bits of entropy are revealed, as the support of the random variable is at most size $|E|$. We also know that each admissible s-tuple is involved in at most $|Q_i|^{r-s}$ distinct r-tuples in Q, and that for each of these that hasn't yet been determined the probability of being a hyperedge is at most t. So, by subadditivity of entropy we can also upper bound the conditional entropy of any given s-tuple's response by $|Q_i|^{r-s}t = 2|Q_i|^{r-s}|E||V|^{-r}p^{-1}$. This overall lets us write

$$\begin{aligned}
&\sum_i H(X_i|X_1 = x_1, \ldots, X_{i-1} = x_{i-1}, Y_1 = y_1, \ldots, Y_{i-1} = y_{i-1}) \leq \\
&\sum_i 10\log(|V|) \cdot (|Q_i|^s p + |Q_i|) \cdot \min\left(\log(|V|), |Q_i|^{r-s}|E||V|^{-r}p^{-1}\right) \leq \\
&\sum_{i \atop |Q_i| > \left(|V|^r|E|^{-1}p\right)^{\left(\frac{1}{r-s}\right)}} \widetilde{O}\left(|Q_i|^s p + |Q_i|\right) + \sum_{i \atop |Q_i| < \left(|V|^r|E|^{-1}p\right)^{\left(\frac{1}{r-s}\right)}} \widetilde{O}\left((|Q_i|^s p + |Q_i|) \cdot |Q_i|^{r-s}|E||V|^{-r}p^{-1}\right) \leq \\
&\left(\sum_i |Q_i|^c\right) \cdot \left(\frac{1}{\left(|V|^r|E|^{-1}p\right)^{\left(\frac{c}{r-s}\right)}} \cdot \widetilde{O}\left(\left(|V|^r|E|^{-1}p\right)^{\left(\frac{s}{r-s}\right)} p + \left(|V|^r|E|^{-1}p\right)^{\left(\frac{1}{r-s}\right)}\right) + |E||V|^{-r}p^{-1}\right) = \\
&\left(\sum_i |Q_i|^c\right) \cdot \widetilde{O}\left(\left(|V|^r|E|^{-1}p\right)^{\left(\frac{s-c}{r-s}\right)} p + \left(|V|^r|E|^{-1}p\right)^{-\max\left(1, \frac{c-1}{r-s}\right)}\right) \leq \\
&\widetilde{o}\left(|V|^{r(1-\gamma)}|E|^\gamma\right) \cdot \widetilde{O}\left(\left(|V|^r|E|^{-1}p\right)^{\left(\frac{s-c}{r-s}\right)} p + \left(|V|^r|E|^{-1}p\right)^{-\max\left(1, \frac{c-1}{r-s}\right)}\right) \leq \\
&\widetilde{o}(|E|).
\end{aligned}$$

Thus, runs where the query cost is less than this bound cannot generate enough entropy to achieve our $\Omega(|E|)$ lower bound on $\sum_i H(X_i|X_1, \ldots, X_{i-1}, Y_1, \ldots, Y_{i-1})$, meaning that the algorithm must sometimes require larger query cost. ■

[1]Note that this defines a distribution with $|E|$ edges in *expectation* as opposed to *exactly* $|E|$. For those concerned about getting exactly the right value of $|E|$, one modification could be to apply this construction using half of the vertices and $1/2^r$ of the edges, and then place the remaining edges arbitrarily among the other vertices. It is vanishingly unlikely that more than $|E|$ edges will be needed if we're doing this construction at half scale, and the observation of the other half of the graph will provide at most $\log(|V|)$ bits about the distribution (namely, the number of edges chosen).

Theorem 4. For any values of r, $s < r$, and any sufficiently large values of $|V|$ and $|E| < \frac{1}{2}\binom{|V|/r}{r}$, any algorithm that exactly recovers unknown r-partite, r-uniform hypergraphs $H = (V, E)$ with high probability using cost-n^c all-s codegree queries must incur cost $\widetilde{\Omega}\left(|V|^{r(1-\gamma)}|E|^{\gamma}\right)$ on some instances, where $\gamma = \max\left(\frac{\binom{r}{s}(r-c)}{(r-s)+\binom{r}{s}(r-c)}, 1 - \frac{c-1}{(r-s)+(s-1)\binom{r}{s}}\right)$.

Proof. Now, we'll take $p = (|V|^{-r}|E|)^{\min\left(\frac{r-c}{(r-s)+(r-c)\binom{r}{s}}, \frac{s-1}{(r-s)+(s-1)\binom{r}{s}}\right)}$, and we'll declare each s-tuple of vertices in the entire graph to be admissible with independent probability p. Fixing the admissible s-tuples, we will say an r-tuple of vertices is admissible if every s of them are admissible. Our distribution over hypergraphs will be to let each admisible spanning r-tuple of vertices be a hyperedge with independent probability $|E||V|^{-r}p^{-\binom{r}{s}}$.

The analysis now follows exactly as in the proof of Theorem 3. We bound the conditional entropy of a query's response given the responses of all previous queries by the sum of the conditional entropies of the response for each admissible s-tuple in the query. In a query Q, each admissible s-tuple yields at most $\widetilde{O}\left(\max\left(1, |Q|^{r-s}|E||V|^{-r}p^{-\binom{r}{s}}\right)\right)$ bits of entropy, and our query can contain at most $\widetilde{O}\left(|Q|^s p + |Q|\right)$ of them by Lemma 10. Appealing to convexity as in the previous case, one finds that in order for all queries to yield sufficient entropy we must sometimes have $\sum_i |Q_i|^c \geq \widetilde{\Omega}\left(|V|^{r(1-\gamma)}|E|^{\gamma}\right)$. ∎

References

[AA05] Noga Alon and Vera Asodi. "Learning a hidden subgraph". In: *SIAM Journal on Discrete Mathematics* 18.4 (2005), pp. 697–712.

[Aba18] Hasan Abasi. "Error-tolerant non-adaptive learning of a hidden hypergraph". In: *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. 2018, pp. 3–1.

[ABM18] Hasan Abasi, Nader H Bshouty, and Hanna Mazzawi. "Non-adaptive learning of a hidden hypergraph". In: *Theoretical Computer Science* 716 (2018), pp. 15–27.

[AC05] Dana Angluin and Jiang Chen. "Learning a hidden hypergraph". In: *International Conference on Computational Learning Theory*. Springer. 2005, pp. 561–575.

[ACK21] S Assadi, D Chakrabarty, and S Khanna. "Graph Connectivity and Single Element Recovery via Linear and OR Queries". In: *29th Annual European Symposium on Algorithms (ESA 2021)*. 2021.

[Alm+25] Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. "More asymmetry yields faster matrix multiplication". In: *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2025, pp. 2005–2039.

[Alo+04] Noga Alon, Richard Beigel, Simon Kasif, Steven Rudich, and Benny Sudakov. "Learning a hidden matching". In: *SIAM Journal on Computing* 33.2 (2004), pp. 487–501.

[AN19] Hasan Abasi and Bshouty Nader. "On learning graphs with edge-detecting queries". In: *Algorithmic Learning Theory*. PMLR. 2019, pp. 3–30.

[ART25] Bethany Austhof, Lev Reyzin, and Erasmo Tani. "Non-adaptive Learning of Random Hypergraphs with Queries". In: *arXiv preprint arXiv:2501.12771* (2025).

[Bea+20] Paul Beame, Sariel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. "Edge estimation with independent set oracles". In: *ACM Transactions on Algorithms (TALG)* 16.4 (2020), pp. 1–27.

[BGM23] Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. "On the Complexity of Triangle Counting Using Emptiness Queries". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques* (2023).

[Bha+19] Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. "Hyperedge estimation using polylogarithmic subset queries". In: *arXiv preprint arXiv:1908.04196* (2019).

[BHW22] Eric Balkanski, Oussama Hanguir, and Shatian Wang. "Learning low degree hypergraphs". In: *Conference on Learning Theory*. PMLR. 2022, pp. 419–420.

[BM09] Nader H Bshouty and Hanna Mazzawi. "Reconstructing weighted graphs with minimal query complexity". In: *International Conference on Algorithmic Learning Theory*. Springer. 2009, pp. 97–109.

[BM10] Nader H Bshouty and Hanna Mazzawi. "Optimal Query Complexity for Reconstructing Hypergraphs". In: *27th International Symposium on Theoretical Aspects of Computer Science-STACS 2010*. 2010, pp. 143–154.

[CEV25] Keren Censor-Hillel, Tomer Even, and Virginia Vassilevska Williams. "Output-Sensitive Approximate Counting via a Measure-Bounded Hyperedge Oracle, or: How Asymmetry Helps Estimate k-Clique Counts Faster". In: *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025, Prague, Czechia, June 23-27, 2025*. Ed. by Michal Koucký and Nikhil Bansal. ACM, 2025, pp. 1985–1996.

[CFS14] Huilan Chang, Hung-Lin Fu, and Chih-Huai Shih. "Learning a hidden graph". In: *Optimization Letters* 8.8 (2014), pp. 2341–2348.

[CFS18] Huilan Chang, Hung-Lin Fu, and Chih-Huai Shih. "Learning a hidden uniform hypergraph". In: *Optimization Letters* 12.1 (2018), pp. 55–62.

[Cha07] Timothy M Chan. "More algorithms for all-pairs shortest paths in weighted graphs". In: *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*. 2007, pp. 590–598.

[CK08] Sung-Soon Choi and Jeong Han Kim. "Optimal query complexity bounds for finding graphs". In: *Proceedings of the fortieth annual ACM symposium on Theory of computing*. 2008, pp. 749–758.

[CLW20] Xi Chen, Amit Levi, and Erik Waingarten. "Nearly optimal edge estimation with independent set queries". In: *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2020, pp. 2916–2935.

[DL21] Holger Dell and John Lapinskas. "Fine-grained reductions from approximate counting to decision". In: *ACM Transactions on Computation Theory (TOCT)* 13.2 (2021), pp. 1–24.

[DLM22] Holger Dell, John Lapinskas, and Kitty Meeks. "Approximately counting and sampling small witnesses using a colorful decision oracle". In: *SIAM Journal on Computing* 51.4 (2022), pp. 849–899.

[DLM24] Holger Dell, John Lapinskas, and Kitty Meeks. "Nearly Optimal Independence Oracle Algorithms for Edge Estimation in Hypergraphs". In: *51st International Colloquium on Automata, Languages, and Programming (ICALP 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. 2024, pp. 54–1.

[Dur+20] Lech Duraj, Krzysztof Kleiner, Adam Polak, and Virginia Vassilevska Williams. "Equivalences between triangle and range query problems". In: *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2020, pp. 30–47.

[Dya+16] Arkadii G D'yachkov, Ilya V Vorobyev, NA Polyanskii, and V Yu Shchukin. "On multistage learning a hidden hypergraph". In: *2016 IEEE International Symposium on Information Theory (ISIT)*. IEEE. 2016, pp. 1178–1182.

[KLP24] Kacper Kluk, Hoang La, and Marta Piecyk. "Graph Reconstruction with Connectivity Queries". In: *International Workshop on Graph-Theoretic Concepts in Computer Science*. Springer. 2024, pp. 343–357.

[KOT25] Christian Konrad, Conor O'Sullivan, and Victor Traistaru. "Graph Reconstruction via MIS Queries". In: *16th Innovations in Theoretical Computer Science Conference (ITCS 2025)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. 2025, pp. 66–1.

[Mat91] Jiří Matoušek. "Computing dominances in E^n (short communication)". In: *Inf. Process. Lett.* 38.5 (June 1991), pp. 277–278. ISSN: 0020-0190. DOI: 10.1016/0020-0190(91)90071-O. URL: https://doi.org/10.1016/0020-0190(91)90071-O.

[MZ13] Claire Mathieu and Hang Zhou. "Graph reconstruction via distance oracles". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2013, pp. 733–744.

[WW10] Virginia Vassilevska Williams and Ryan Williams. "Subcubic equivalences between path, matrix and triangle problems". In: *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*. IEEE. 2010, pp. 645–654.