Worst-Case and Average-Case Hardness of Hypercycle and Database Problems

Cheng-Hao Fu [*] Andrea Lincoln [†] Rene Reyes [‡]

April 25, 2025

Abstract

In this paper we present tight lower-bounds and new upper-bounds for hypergraph and database problems. We give tight lower-bounds for finding minimum hypercycles. We give tight lower-bounds for a substantial regime of unweighted hypercycle. We also give a new faster algorithm for longer unweighted hypercycles. We give a worst-case to average-case reduction from detecting a subgraph of a hypergraph in the worst-case to counting subgraphs of hypergraphs in the average-case. We demonstrate two applications of this worst-case to average-case reduction, which result in average-case lower bounds for counting hypercycles in random hypergraphs and queries in average-case databases. Our tight upper and lower bounds for hypercycle detection in the worst-case have immediate implications for the average-case via our worst-case to average-case reductions.

[*]Boston University, `chenghao@bu.edu`.

[†]Boston University, `andrea2@bu.edu`.

[‡]Boston University, `rdreyes@bu.edu`. Supported by NSF Grant No. 2209194.

Contents

1 Introduction

The fine-grained hardness of hypergraph problems is known to have interesting connections to both theoretical and practical problems in computer science. For example, hypergraph problems have deep connections to solving constraint satisfaction problems: algorithms for hypercycle problems in hypergraphs can be used to solve various Constraint Satisfaction Problems (CSPs), including Max-k-SAT [LWW18]. On the practical side, a recent line of work in database theory (initiated by Carmeli and Kröll [CK21]) has shown that certain queries of interest can be interpreted as problems about detecting, listing and counting small structures in hypergraphs. Nevertheless, the fine-grained study of hypergraph problems remains largely under-explored, especially when compared to the volume of work on fine-grained hardness of graph problems. In this paper, we aim to extend the understanding of hypergraphs in the average-case setting. We focus on two main directions: the worst-case hardness of hypercycle problems and the average-case hardness of counting small subhypergraphs.

To our knowledge, this work is the first to show that the conditional hardness of hypercycle detection is dependent on both the length of the hypercycle and the size of the hyperedges. This is somewhat surprising, given that for other problems, such as hyperclique detection, hardness is conjectured to depend only on the size of the hyperclique. In the weighted case we get tight lower bounds for all hypercycle lengths. In the unweighted case, we show that brute-force is necessary for short hypercycles, resulting in tight upper and lower bounds. For the parameter regime where we can't show the brute force lower bound, we give a new faster algorithm for longer unweighted hypercycles using matrix multiplication. This shows the brute-force lower bound is actually false past the point where our reduction stops working.

For our average-case results, we build on a line of work that has used the error-correcting properties of polynomials to show average-case fine-grained hardness. This technique, introduced by Ball et al. [BRSV17] has already been applied to hypergraph problems by Boix-Adserà et al. [BBB19], who show that counting small hypercliques in random hypergraphs where all hyperedges have the same size is as hard as in the worst-case. In the graph setting, their approach was generalized by Dalirroyfard et al.[DLW20] who showed similar worst-case-to-average-case reductions for counting any small subgraph. We further generalize their result to show that this is true for counting small subhypergraphs in random hypergraphs, including ones where the hyperedges have different sizes. The worst-case to average-case reduction also allows us to prove that counting queries on random databases are hard on average.

These two results are highly complementary. Applying our worst-case-to-average-case reduction to our hypercycle hardness results immediately gives us average-case hardness of hypercycle problems. Furthermore, understanding the worst-case hardness of other hypergraph substructures would shed light on what types of counting queries are hard on average. Finally, in the process of exploring these areas, we have found many new avenues to explore, which range from understanding the parameter regimes where we do not get tight upper and lower bounds to improving some generic techniques from traditional worst-case reductions.

1.1 Hypercycle Algorithms and Lower Bounds

A k-hypercycle in a u-uniform hypergraph is a list of k distinct nodes $v_1, \ldots, v_k$ such that every hyperedge of u adjacent nodes exists in the hypergraph. That is, a k-hypercycle consists of the edges $(v_i, \ldots, v_{i+u-1 \mod k})$ for all $i \in [k]$. In the literature, this is often called a tight hypercycle, but we will omit the 'tight' and call these hypercycles throughout this paper (following the norm of [LWW18]).

We demonstrate new algorithms and conditional lower bounds for the minimum hypercycle and unweighted hypercycle problems. Our upper and lower bounds for minimum hypercycle are tight, as shown in Figure 1. In the unweighted case, we show that the hardness of hypercycle detection depends on the relationship between the hyperedge size and hypercycle length, as shown in Figure 2. The conditional lower bounds are derived from the minimum k-clique hypothesis and the (k,u)-hyperclique hypothesis:

Definition 1. MIN WEIGHT k-CLIQUE HYPOTHESIS (e.g. [AWW14, BT17]) There is a constant $c > 1$ such that, on a Word-RAM with $O(\log(n))$-bit words, finding a k-Clique of minimum total edge weight in an n-node graph with non-negative integer edge weights in $[1, n^{ck}]$ requires $n^{k-o(1)}$ time.

Definition 2. (k,u)-HYPERCLIQUE HYPOTHESIS [LWW18] Let $k > u > 2$ be integers. On a Word-RAM with $O(\log(n))$ bit words, finding a k-hyperclique in a u-uniform hypergraph on n nodes requires $n^{k-o(1)}$ time.

As evidence for this hypothesis, [LWW18] give a reduction from the maximum degree-u-CSP problem to the (k,u)-hyperclique problem. They also give a reduction from hyperclique detection to hypercycle detection. The hard instances output by this reduction only cover a specific hypercycle length k for each uniformity u and this seems inherent to the technique. A natural question to ask is whether the hardness of finding k-hypercycles in u-uniform hypergraphs depends on the relationship between u and k. We answer this question in the affirmative for both weighted and unweighted hypercycle problems. This is surprising since the hardness of other hypergraph problems such as (k,u)-hyperclique is conjectured to be independent of this relationship.

For min-weight k-hypercycle, we show that brute-force is required when k is less than $2u-1$. When $k \geq 2u-1$, we give an algorithm with runtime independent of k. For unweighted hypercycle, we show that brute force is needed for all k up to the length of the instances output by the [LWW18] reduction. For longer hypercycles, we show this is not the case by giving novel algorithms which use matrix multiplication to get a speedup.

We now provide a more detailed overview of our techniques along with formal theorem statements for our upper and lower bounds.

1.1.1 Weighted Hypercycle

In Section 5 we give the results for weighted hypercycle, depicted in Figure 1. The matching lower bounds come from a reduction between cliques in graphs (2-uniform hypergraphs) and hypercycles. Informally the minimum k-clique hypothesis states that finding a minimum k-clique in a graph can't be done faster than the naive algorithm which takes $n^{k-o(1)}$ time (see definition 1). We prove the following.

Theorem 1.1. *If the minimum k-clique hypothesis holds then the minimum k-hypercycle problem in a u-uniform hypergraph requires $n^{k-o(1)}$ time for $k \in [u+1, 2u-1]$.*

1.1.2 Unweighted Hypercycle

For the unweighted version of the problem we get tight upper and lower bounds in a u-uniform graph for all $k \leq \gamma_3^{-1}(u)$, where we define $\gamma_3(k) = k - \lceil k/3 \rceil + 1$ as is done in [LWW18] and $k = \gamma_3^{-1}(u)$ is the *largest* k such that $\gamma_3(k) = u$. Intuitively, this is the largest k such that any three nodes in the hypercycle are

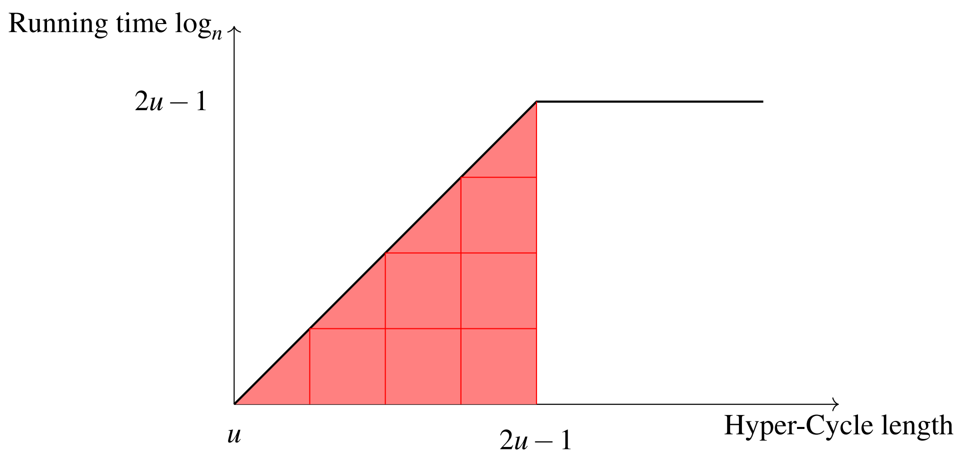

Figure 1: The running time of minimum weight hyper-cycle in a weighted u-uniform hyper-graph. The exponent of the running time is indicated by the black line and the lower bound is matching, as indicated by the hatched red area. Note the running time is $O(n^u)$ for a u length hyper-cycle and $O(n^{2u-1})$ for a $2u-1$ length hyper-cycle. Then for all hyper-cycles of length $k > 2u-1$ the running time continues to be $O(n^{2u-1})$.

covered by at least one hyperedge of size u, which happens when $k \approx 3u/2$. For larger hypercycles where $k \geq \gamma_3^{-1}(u)+1$, we can pick three partitions such that no hyperedge covers all three. This property allows us to reduce the problem to triangle-counting and then use matrix multiplication to obtain an algorithmic speedup. This algorithm, given in Section 4, runs in time $n^{k-3+\omega}$ where ω is the matrix multiplication constant. We depict the upper and lower bounds for unweighted hypergraphs in Figure 2.

We get tight lower bounds for hypercycle up to the "phase transition" point at $\gamma_3(k)$. These lower bounds come from the (u,k)-hyperclique hypothesis. Informally, this says detecting a k-hyperclique in a u-uniform hypergraph if $k > u > 2$ requires $n^{k-o(1)}$ (see Definition 2). We prove the following theorem in section 3.2.

Theorem 1.2. *Under the $(3,k)$-hyperclique hypothesis, an algorithm for finding (counting) (u,k)-hypercycle for $k \in [u, \gamma_3^{-1}(u)]$ requires $O(n^{k-o(1)})$ time.*

When $k > \gamma_3(k)$ we show an algorithm which is better than brute force by a factor dependent on ω (where ω is the matrix multiplication constant).

Theorem 1.3. *There exists a time-$\tilde{O}(k^k(n+m+n^{2u-1-(3-\omega)}))$ algorithm for finding k-hypercycles in any n-node hypergraph G when $k \geq 2u-1$.*

Proving conditional lower bounds in this regime might unexpectedly imply conditional lower bounds on ω. In short, we get tight lower bounds for all values of k where lower bounds wouldn't have implications for matrix multiplication's running time, ω.

1.1.3 Average-Case Implications

We can apply our worst-case to average-case reduction techniques to get lower bounds for random hypergraphs. Notably, for constant-length unweighted-hypercycle, the hardness of counting in the worst-case is equivalent to the hardness in the average-case up to polylog factors. This gives tight upper and lower

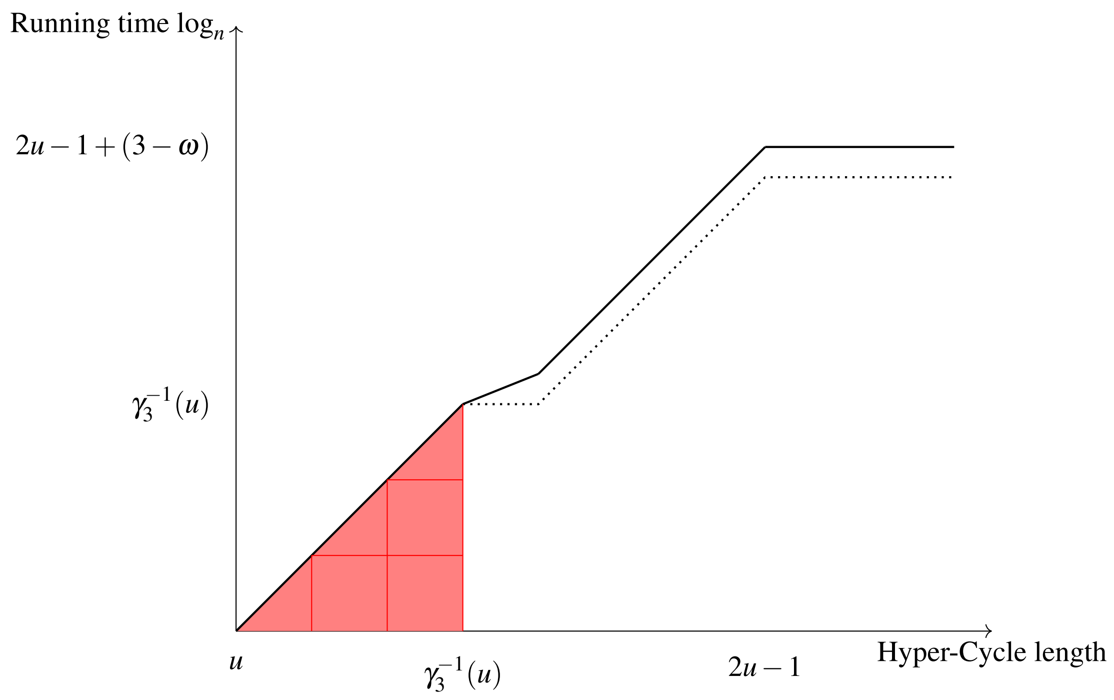

Figure 2: The running time of *unweighted* hyper-cycle in a u-uniform hyper-graph. The exponent of the running time is indicated by the black line. The lower bound is matching for the hatched red area from cycles of length u to $\gamma_3^{-1}(u)$ and the running time is $O(n^k)$ for a k-hypercycle. Then from $\gamma_3^{-1}(u)$ to $2u-1$ the running time is $\tilde{O}(n^{k-3+\omega})$ for k-hypercycle where ω is the matrix multiplication constant ($\omega < 2.3716$ [WXXZ23]). The dotted line represents the running time the algorithm would achieve if $\omega = 2$. There is an algorithm running in $\tilde{O}(n^{2u-1-(3-\omega)})$ time for all $k \geq 2u-1$. The shading stops after $\gamma_3^{-1}(u)$ because we don't know of any tight lower bounds past this point.

bounds for the average-case hardness of counting k-hypercycles in the same regime of cycle length for a given uniformity when $k \leq \gamma_3^{-1}(u)$. Informally, if the $(3,k)$-hyperclique hypothesis holds then counting k-hypercycles in u-uniform Erdős-Rényi hypergraphs when $k \in [u, \gamma_3^{-1}(u)]$ requires $\tilde{O}(n^{k-o(1)})$ time. We prove this in Section 3.

1.2 Counting Subhypergraphs on Average is as Hard as Detecting them in the Worst Case

To enable our average-case results for databases and for counting hypercycles we provide a new reduction. Our reduction can handle non-uniform hypergraphs, meaning hypergraphs with edges of *different* sizes (see Figure 3). This is necessary for the results to apply to the databases setting. We transform the problem into a low degree polynomial and use a known framework to show average-case hardness.

However, to get back to average-case graphs that aren't color-coded we use a new form of inclusion exclusion. Prior work introduced inclusion-edgesculsion [LWW18]. We present a simultaneously generalized and simplified proof which allows us to show hardness for counting these subhypergraphs in uniformly randomly sampled hypergraphs (even those where there are mixed hyperedge sizes).

Theorem 1.4 (Informal)**.** *Let k be a constant. Counting the number of k-node hypergraphs H made up of nodes $v_1, \ldots, v_k$ in k-partite hypergraphs G with partitions $V_1, \ldots, V_k$ in the worst case where we only count*

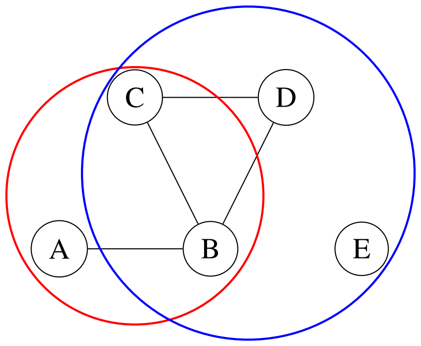

Figure 3: An example of a small subgraph with hyperedges of multiple different sizes. This graph contains hyperedges $\{A,B\},\{B,C\},\{B,D\},\{C,D\},\{A,B,C\}$, and $\{B,C,D,E\}$. We depict the 3-width edge as a red circle, the 4-width edge as a blue circle, and the 2-width edges as black lines.

copies of H where $v_i \in V_i$ can be solved with polylog calls to a counter for hypergraphs H in uniformly random hypergraphs.

Via color-coding this means that if detecting a hypergraph H in the worst case is $T(n)$ hard then counting that hypergraph in the uniform average-case is hard. So, starting from the hardness of either detection or the counting problem in a k-partite graph implies the hardness of the counting problem in the uniform average-case.

These approaches allow us to show hardness for database problems and the problem of counting sub-hypergraphs. We demonstrate the usefulness of the improved approach by showing its applications to these two problem areas.

1.3 Database Results

We now provide a high-level introduction to the databases setting we are interested in. For full definitions of these problems, see Section 7. A table in a database is called a relation. A relation, R_i, is a set of tuples. For example, a relation on $(user\_id, user\_name)$ would be a set of tuples grouping user ids and user names. A database may have many relations, and the relations may have tuples of different sizes (e.g. the example database could also include a table of $(user\_id, purchase\_id, purchase\_date)$).

Queries over a database can take many forms, but a classic form is a join query. To be concrete, if we have a table with two relations $R_1(a,b)$ and $R_2(a,c,d)$ then we can have a query $Q_1(a,b,c,d) \leftarrow R_1(a,b), R_2(a,c,d)$. If we ask to enumerate all answers to Q_1 then we want all tuples (a',b',c',d') such that $(a',b') \in R_1$ and $(a',c',d') \in R_2$. Some query types (e.g. 'FIRST' in SQL) ask to give a single answer. In this paper we will focus on queries that ask to count the number of matching outputs in the query (e.g. 'COUNT' in SQL). In database theory, enumeration queries are the most studied. While these queries are often trivial in the average-case, non-enumerating query types are still often hard-on-average. See Appendix A.2 for more discussion of enumeration and counting on average.

You might notice that relations look like lists of hyperedges and queries look like requests to enumerate, detect, or count subhypergraphs of the implied hypergraph. However, these database "hyperedges" are directed. The tuple $('Jane','Doe')$ and $('Doe','Jane')$ have different meanings, whereas in an undirected hypergraph the edges are simply sets. However, these problems are similar enough that we can apply analogous worst-case to average-case reduction techniques to show hardness on average for these database problems.

We now work through an example to help solidify the similarity between databases and hypergraphs. The hypergraph from Figure 3 could be represented as a database by the following list of relations:

$$R_1(A,B), R_2(B,C), R_3(B,D), R_4(C,D), R_5(A,B,C), R_6(B,C,D,E).$$

Then, to count the number of appearances of the hypergraph from Figure 3 within the hypergraph representing the entire database, we could make the following query:

$$Q_1(A,B,C,D,E) \leftarrow R_1(A,B), R_2(B,C), R_3(B,D), R_4(C,D), R_5(A,B,C), R_6(B,C,D,E).$$

A natural question for databases is this: how hard are queries on uniformly random databases? If our real world case involves uniformly distributed data, when can we hope to find algorithmic improvements? We answer this question by giving lower bounds for counting queries in Section 7 and explaining why *small* enumeration queries will be easy in Appendix A.

1.4 Context and Prior Works

There has been a lot of work on average-case fine-grained complexity. Ball et al. [BRSV17] showed that starting from popular fine-grained hypotheses, evaluating certain functions could be shown to be hard on average. Later Ball et al. [BRSV18] showed that you can build a fine-grained proof of work using these functions. [BBB19] showed that there is an equivalence, up to polylog factors, between counting constant sized hypercliques in the worst-case and in the average-case of Erdős-Rényi Hypergraphs. Goldreich presented an alternative method to count cliques mod 2 [Gol20]. Dalirrooyfard et al. [DLW20] generalized these ideas and presented a method to produce a worst-case to average-case reduction for any problem which can be represented by a ‘good’ low degree polynomial. Additionally they showed that counting small subgraphs (not just cliques) is equivalently hard in the worst-case and in Erdős-Rényi graphs. In this paper we further generalize this result to the hypergraph setting. Specifically, we show that counting small sub**hyper**graphs is equivalently hard in the worst-case and average-case. We also give worst-case to average-case hardness for many classes of **counting database queries**. Our contributions include expanding the previous results and connecting fine-grained average-case complexity to database theory.

Recently, the database theory community has had increased interest in the fine-grained hardness of various database queries. This is highlighted by many recent papers in the area. Carmeli and Kröll [CK21] use fine-grained hypotheses to show that answering unions of conjunctive queries are hard. Carmeli and Segoufin [CS23] explore the fine-grained complexity of enumeration queries with self joins. Bringman et. al. [BCM22] characterize the amount of preprocessing time needed to get polylogarithmic access time. We will show that for counting queries over a constant number of relations that don’t involve self-joins, the worst-case and average-case hardness are equivalent up to polylogarithmic factors.

Carmeli et al [CTG+23] eschew enumeration to explore logarithmic time access to the k^{th} answer to a conjunctive query after a quasi-linear database preprocessing. They provide fast algorithms for these problems. Recent work from Wang, Willsey, and Suciu [WWS23] proposed a *free join* which achieves worst-case optimality. These are algorithms where the runtime matches the worst-case output size. As we explore counting queries where the output size is a single word, $O(\lg(n))$ bits, worst-case optimality is impossible as long as the full input must be read to answer the query. We thus focus on efficiency more generally. A 2023 Simons program on Logic and Algorithms in Database Theory and AI ran a workshop on Fine-Grained Complexity, Logic, and Query Evaluation[NPRW23]. In this workshop, the open problem of the average-case hardness of database queries was brought up in the open problem session on September 27th. In this paper we have explored and proved this average-case hardness for many kinds of counting queries in databases. In this paper we seek to give a new definition of an average-case for database theory and provide a worst-case to average-case reduction using fine-grained complexity techniques.

Hypercycles ("tight hypercycles" in much of the literature) have been well-studied in both math and computer science. Allen et. al. [ABCM15] showed that for k-hypercycles in a u-uniform graph where $k \equiv 0 \mod u$ must exist if there are at least $(k/n+\delta)\binom{n}{u}$ hyperedges for some constant $\delta > 0$. It is unclear if when $\delta = \Theta(1/n)$ you can still guarantee a hypercycle exists. In this paper we show hardness for $k < 2u$, where these results do not apply. Huang and Ma [HM19] continued the exploration of this existential hypercycle question, showing that, in some sense, the previous result is tight up to constants. Later, Allen et al. [AKPP18] gave a faster algorithm for tight Hamiltonian hypercycles in random graphs, that is in average-case graphs. The results in our paper rely on k being small, for larger k the worst-case to average-case reduction becomes increasingly expensive. So their work and ours shows an interesting dynamic where short cycles have more similar hardness in the worst-case and average-case and there is a divergence as the cycle length grows. Lincoln, Vassilevska and Williams [LWW18] give a reduction from max-k-SAT to hypercycle and then to cycle. However, their results for cycle are not tight. They give a bound of $m^{3/2-o(1)}$ for k-cycle. They give a tighter bound for 7-cycle of $m^{7/5-o(1)}$ in graphs with $m = n^{5/4}$. In theorem C.1 they show a lower bound for hypercycle. However, this is *dense* hypercycle. In our paper we present *tight* bounds for sparse hypercycle. We also present a new faster algorithm for unweighted hypercycle. Our weighted lower bounds are tight. Our unweighted lower bounds are tight until the regime where matrix multiplication is used in the fastest algorithm.

1.5 Paper Structure

We present the basic background and definitions in Section 2. We show tight lower bounds for hypercycles in Section 3. We give fast algorithms for hypercycles in Section 4. We give the tight upper and lower bounds for minimum hypercycle in Section 5. We give the worst-case to average-case reduction for counting subhypergraphs in Section 6. We show the average-case hardness for (self-join-free) counting queries in Section 7. We present open problems in Section 8. Finally, we include extra discussion in Appendix A.

2 Preliminaries

In this paper we combine ideas and techniques from average-case fine-grained complexity, databases, and worst-case fine-grained complexity and algorithms. As a result we will mostly define the needed terms in the corresponding sections. Our preliminaries are short and focus on hypergraphs as these appear in most sections.

Definition 3. A u-uniform hypergraph G has a vertex set V and a set of hyperedges E where each hyperedge is a set of u vertices from V.

A hypergraph of mixed sizes with hyperedge set sizes of $u_1, \ldots, u_k$ has a vertex set V and a set of hyper edges E where each hyperedge is a set of vertices from V and the size of that set must be $u_1, \ldots,$ or u_k.

Definition 4. A tight k-hypercycle in a u-uniform hypergraph G is a list of k nodes $v_1, \ldots, v_k \in V$ where every hyperedge

$$(v_i, v_{i+1 \mod k} \cdots, v_{i+u-1 \mod k})$$

exists in E. That is, every hyperedge of u adjacent vertices on the cycle exists.

Definition 5. A k-hypercycle in a u-uniform hypergraph G is a list of k nodes $v_1, \ldots, v_k \in V$ where every hyperedge $(v_{i_1}, \ldots, v_{i_u})$ where $i_j \in [1,k]$ exists E. That is, every possible hyperedge between the k nodes exists in the graph.

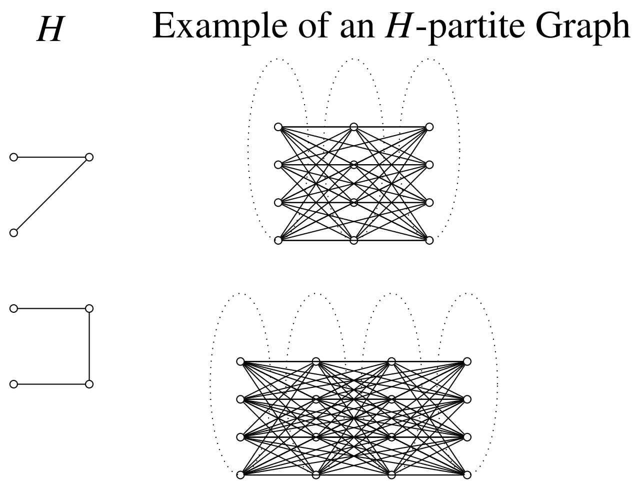

Figure 4: An example of two small graphs and corresponding H-partite graphs.

In this paper we use the term k-hypercycle to refer to a tight k-hypercycle. We use these concepts throughout the paper. We also use a more general notion of a subgraph of a hypergraph.

Definition 6. A subhypergraph or subgraph of a hypergraph G (the hypergraph could have mixed uniformity or not) is a graph H whose vertex set and edge set is a subset of G's vertex and edge set. That is, if H has a vertex set V_H and edge set E_H then H is a subgraph of G if $V_H \subset V$ and $E_H \subset E$.

Theorem 2.1 (Theorem 3.1 from [LWW18]). *Let G be a u-uniform hypergraph on n vertices V, partitioned into k parts $V_1, \ldots, V_k$.*

Let $\gamma_u(k) = k - \lceil k/u \rceil + 1$.

In $O(n^{\gamma_u(k)})$ time we can create a $\gamma_u(k)$-uniform hypergraph G' on the same node set V as G, so that G' contains an k-hypercycle if and only if G contains an k-hyperclique with one node from each V_i.

If G has weights on its hyperedges in the range $[-W,W]$, then one can also assign weights to the hyperedges of G' so that a minimum weight k-hypercycle in G' corresponds to a minimum weight k-hyperclique in G and every edge in the hyperclique has weight between $[-\binom{\gamma_u(k)}{u}W, \binom{\gamma_u(k)}{u}W]$. Notably, $\binom{\gamma_u(k)}{u} \leq O(k^u)$.

For intuition on this theorem see Appendix 3.1.

Definition 7. From [DLW20] Let $H = (V_H, E_H)$ be a k-node graph with $V_H = \{x_1, \dots, x_k\}$.

An H-partite graph is a graph with k partitions $V_1, \dots, V_k$. This graph must only have edges between nodes $v_i \in V_i$ and $v_j \in V_j$ if e $(x_i, x_j) \in E_H$. (See Figure 4 in the appendix.)

3 Short Unweighted Hypercycles Require Brute-Force

We will begin by showing that short hypercycles are tightly hard. We will do so for hypercycles of lengths at most $\gamma_3^{-1}(u)$. Recall that $\gamma_3(k) = k - \lceil k/3 \rceil + 1$, so $\gamma_3^{-1}(u) \approx 3u/2$. On an intuitive level $\gamma_3^{-1}(u)$ corresponds to the largest length of cycle such that all sets of three nodes are included in at least one hyperedge. In this section we will show that given this constraint, the best algorithm for k-hypercycle is the naive brute force n^k algorithm. In the next section we will show that if the cycle length is even one larger than $\gamma_3^{-1}(u)$, then we can beat brute force using fast matrix multiplication. In this sense our brute force lower-bound is tight, we couldn't extend the brute force lower bound to cycles of any longer length.

Our lower bounds are based on the min-weight k-clique and (u,k)-hyperclique hypotheses, which states that these problems require $O(n^{k-o(1)})$ time (see Definition 2). We use a reduction from [LWW18] to show hardness for our hypercycle problems; to make this work self-contained, we now give the necessary intuition.

3.1 Reduction from k-clique

Our lower bounds are based on the min-weight k-clique and (u,k)-hyperclique hypotheses, which states that these problems require $O(n^{k-o(1)})$ time (see Definition 2). We will use a reduction from [LWW18] to show hardness for our hypercycle problems.

Theorem 3.1 (Theorem 3.1 from [LWW18])**.** *Let G be a u-uniform hypergraph on n vertices V, partitioned into k parts $V_1, \dots, V_k$.*

Let $\gamma_u(k) = k - \lceil k/u \rceil + 1$.

In $O(n^{\gamma_u(k)})$ time we can create a $\gamma_u(k)$-uniform hypergraph G' on the same node set V as G, so that G' contains an k-hypercycle if and only if G contains an k-hyperclique with one node from each V_i.

If G has weights on its hyperedges in the range $[-W,W]$, then one can also assign weights to the hyperedges of G' so that a minimum weight k-hypercycle in G' corresponds to a minimum weight k-hyperclique in G and every edge in the hyperclique has weight between $[-\binom{\gamma_u(k)}{u}W, \binom{\gamma_u(k)}{u}W]$. Notably, $\binom{\gamma_u(k)}{u} \leq O(k^u)$.

To give the reader context and intuition we will explain the core ideas of this previous work. The main idea behind this reduction is to output k-partite hypergraphs such that the vertex partitions can be arranged in a circle and hyperedges only exist between adjacent partitions. Then, the value $\gamma_u(k)$ corresponds to the

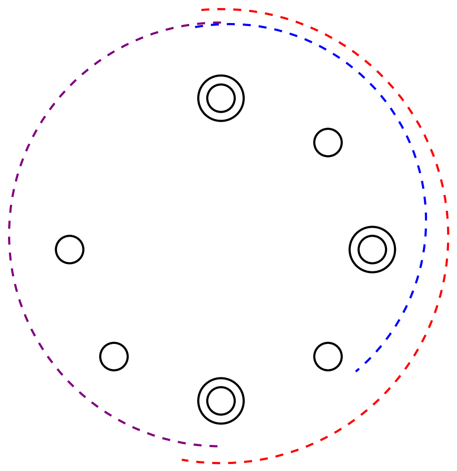

Figure 5: The case of $k = 7$ where we start with a 3-uniform graph. The double circles indicate the 'farthest apart' three nodes can be. Note the purple and blue arcs indicate the furthest a uniformity 4 hyperedge can go while including the top node, and that neither hyperedge includes all three double circled nodes. Further note that the red edge, which is a hyperedge of uniformity 5, covers all three double circled nodes. This is what $\gamma(\cdot)$ is capturing.

smallest edge uniformity such that this type of edge will cover any possible subset of u vertex partitions. This allows this more restricted type of hyperedge to capture information about any possible hyperedge in the original u-uniform hypergraph. See Figure 5 for a small illustrative example.

While this reduction shows a relationship between the hardness of hypercycle detection and hypercliques, it is important to observe that it requires a very specific uniformity $\gamma_u(k)$. A natural question to ask is: what happens for other combinations of u and k?

We show that the hardness varies based on the relationship between uniformity u and cycle length k. Note that for fixed u and sufficiently large constant k, algorithms for finding or counting hypercycles yield runtimes independent of k. This can be seen as a "plateau" in the hardness of hypercycle detection for any fixed u seen in Figure 2 and Section 4. For all k where brute force is the best known algorithm, we give a matching lowerbound.

3.2 Tight Hardness for Short Hypercycles

We begin by showing that for finding relatively short hypercycles, brute-force algorithms are optimal unless the (u,k)-hyperclique hypothesis is false. Note that the reduction from Theorem 2.1 preserves the number of vertices in the graph G. We use this second fact throughout the proof of our lower bound for this parameter regime.

Now, what exactly do we mean by "short hypercycles"? To formally define the range in which we get tight results, we must introduce some notation, which allows us to reason about the uniformities we show hardness for.

Definition 8. Recall the function $\gamma_u(k) = k - \lceil \frac{k}{u} \rceil + 1$ from Theorem 2.1. Then, for constant c define:

$$\gamma_c^{-1}(u) = \max\{k : \gamma_c(k) = u\}$$

This function will make it easier to discuss the hypercycle lengths for which the hyperclique reduction yields hardness. These correspond to the range $k \in [u, \gamma_3^{-1}(u)]$, which we will sometimes refer to as *short hypercycles*. In particular, we show that improving over brute-force search for short hypercycles would yield faster algorithms for finding hypercliques. More formally we will prove that:

Theorem 1.2. *Under the $(3,k)$-hyperclique hypothesis, an algorithm for finding (counting) (u,k)-hypercycle for $k \in [u, \gamma_3^{-1}(u)]$ requires $O(n^{k-o(1)})$ time.*

This conditional lower bound follows from the following ideas, which we prove as intermediate lemmas. First, the function γ_3 is monotonically increasing, so applying the hyperclique reduction for $k \leq \gamma_3^{-1}(u)$ yields a uniformity $u' \leq u$. Furthermore, we show a self-reducibility property of the hypercycle problem, which is that algorithms solving k-hypercycle on a given uniformity can be used to solve it on smaller uniformities. Putting these ideas together, we are able to show that any hardness given by the hyperclique reduction on uniformities $u' < u$ can be extended to u.

3.2.1 Understanding the Hyperclique to Hypercycle Reduction

We begin by showing monotonicity of γ.

Lemma 3.2. *For all $c \geq 2$, the function $\gamma_c(k)$ is monotonically increasing.*

Proof. We will show that for any k, $\gamma_c(k+1) \geq \gamma_c(k)$. Note that $\gamma_c(k+1) = (k+1) - \lceil \frac{k+1}{c} \rceil + 1$. Then, since $\lceil \frac{k+1}{c} \rceil \leq \lceil \frac{k}{c} \rceil + 1$, we get $\gamma_c(k+1) \geq (k+1) - (\lceil \frac{k}{c} \rceil + 1) + 1 = \gamma_c(k)$. $\square$

Corollary 3.3. *For every k such that $u \leq k < \gamma_3^{-1}(u) : \gamma_3(k) \leq u$.*

We next want to show that if finding hypercycles is hard on u-uniform graphs, it is also hard in graphs with uniformity $u' > u$. To do this, we will make use of k-circle-layered hypergraphs:

Definition 9. k-circle-layered hypergraphs We say that a hypergraph G is k-circle-layered if it is k-partite and its vertex partitions $V_1, \cdots, V_k$ can be arranged in a circle such that hyperedges only exist between adjacent vertex partitions. That is between u partitions $V_i, V_{i+1 \mod k}, \ldots, V_{i+u-1 \mod k}$.

While this seems like a highly restrictive definition, [LWW18, Lemma 2.2] shows that a time-$O(T(n,m))$ algorithm for finding k-hypercycles in this type of hypergraph can be used to find k-hypercycles in arbitrary hypergraphs using time $\tilde{O}(k^k(n+m+T(n,m)))$. Thus, for practical purposes we will treat the problems as equivalent. Moreover, when indexing into a vertex v_i in a k-hypercycle or a partition V_i in a k-circle-layered graph, the index i should be interpreted as shorthand for $(i \mod k)$.

Now, we need one more property of k-circle-layered hypergraphs before we prove our self-reducibility result. The structure of these graphs are useful for reasoning about hypercycles when we can ensure that any hypercycle in the graph uses exactly one vertex from each partition. When this is true, we can think of the hypercycle as "going around" the circular structure of the hypergraph. Nonetheless, there are some scenarios in which more than one vertex from some partition may appear in the hypercycle. We can think of these as "backward cycles", since they must turn around at some point to re-visit the repeated partition.

We will now show that such hypercycles cannot exist when $k \not\equiv 0 \mod u$.

Lemma 3.4. *Suppose there exists a k-hypercycle $v_0, \ldots, v_{k-1}$ in a k-circle layered hypergraph with uniformity u that does not use exactly one node from each partition. Then, we have that $k \equiv 0 \mod u$.*

Proof. We begin with the observation that such a hypercycle must have at least one partition where it does not have a node. Locate one partition such that it has a node from the cycle, but where one of its adjacent partitions does not. WLOG, label this partition V_0, and the adjacent partition that does not have a vertex from the cycle V_{k-1}, and the vertices between them V_1 to V_{k-2} in order. This means there is no edge in the cycle that contains both a vertex from V_0 and V_{k-1}.

This choice allows us to make another observation. For every single edge in the cycle, it must be the case that there is exactly one vertex within it that comes from a partition where the label is a multiple of u. This is because all such partitions in consideration for the cycle are at least a distance of u away from each other, and no edge spans that distance around the circle. However, every edge must also intersect the partition somewhere, as there are only $u-1$ partitions between two partitions labeled with multiples of u.

This allows us to make the following claim: For all a such that $au < k-1$, $v_{au} \in V_{ru}$ for some $r \in \mathbb{Z}$.

We prove this claim using induction. The base case is trivial. Given that the statement is true for a, we aim to show that it is true for $a+1$. We have that $v_{au}, v_{au+1} \ldots, v_{a(u+1)-1}, v_{a(u+1)}$ is a sequence in the hypergraph, and that the first u vertices in the sequence form an edge, as do the last ones. This means that $v_{au+1} \ldots, v_{a(u+1)-1}$ cannot contain a vertex from a partition with a multiple of u, and thus $v_{a(u+1)}$ must come from such a partition.

This proves the claim. Now, towards a contradiction, we suppose that k is not a multiple of u. We observe the hyperedge $v_{k-u+1}, \ldots, v_{k-1}, v_0$. We note that there must be some vertex in the first $u-1$ vertices that has a label that is a multiple of u. Call this v_i. We also note that this hyperedge must span the partitions $V_0, \ldots V_{u-1}$, as no vertex is in V_{k-1}. This, combined with the claim, implies that v_i and v_0 are both in V_0, a contradiction. □

3.2.2 Increasing Uniformity Preserves Hardness

We can now show the following self-reducibility lemma about finding and counting k-hypercycles:

Lemma 3.5. *Let k be a hypercycle length such that $k \not\equiv 0 \mod u$. Suppose there exists $\varepsilon > 0$ such that there is an algorithm for finding (counting) (u,k)-hypercycles in k-circle-layered graphs that runs in time $O(n^{k-\varepsilon})$. Then, for any uniformity $u' < u$, there exists an algorithm for finding (counting) (u',k)-hypercycle in k-circle-layered graphs running in time $O(n^{k-\varepsilon} + n^u)$.*

Proof. We will show how to map an n-node u'-uniform hypergraph $G = (V,E)$ to an n-node u-uniform hypergraph G' such that G' has a k-hypercycle if and only G has one.

We construct G' as follows. We let its vertex set $V' = V$, and we will add edges so that G' is also k-circle-layered. Then, we construct hyperedges in G' as follows.

For each hyperedge $(v_i, v_{i+1}, \cdots, v_{i+u'-1})$, we create $n^{u-u'}$ hyperedges by creating all possible "extensions" of the hyperedge to the next $u-u'$ partitions. More concretely, we know that the vertices of this hyperedge satisfy $v_i \in V_i, v_{i+1} \in V_{i+1}, \cdots, v_{i+u'-1} \in V_{i+u'-1}$. Then, for all possible combinations of vertices $y_{i+u'} \in V_{i+u'}, \cdots, y_{i+u'-1} \in V_{i+u-1}$ we will add the hyperedge $(v_i, \cdots, v_{i+u'-1}, y_{i+u'}, \cdots, y_{i+u-1})$ to E.

This mapping results in a hypergraph G' such that $|V'| = |V| = n$ and $|E'| = |E|^{u-u'}$. This blowup in the number of edges will not be a problem since the algorithm we are assuming for (u,k)-hypercycle is agnostic to sparsity.

Now we must show that G' preserves k-hypercycles from G and does not create any new ones. First, assume there exists a hypercycle $v_1, v_2, \cdots, v_k$ in G. Then, for all consecutive sets of u' vertices, we have that the hyperedge $(v_i, v_{i+1}, \cdots, v_{i+u'-1}) \in E$. Since we added all possible extensions of this hyperedge to E', this necessarily implies that for the specific extension corresponding to the next $u - u'$ vertices of the hypercycle, the hyperedge $(v_i, v_{i+1}, \cdots, v_{i+u'-1}, v_{i+u'}, \cdots, v_{i+u-1}) \in E$. By applying this reasoning to all of the edges in the hypercycle, we get that $v_i, \cdots, v_k$ also form a k-hypercycle in G'.

Now we show why G' does not contain any hypercycles that cannot be traced back to a hypercycle in G. The key idea for this will be the fact that for every hyperedge in G', the first u' nodes in the hyperedge form a hyperedge in G.

Assume there exists a hypercycle $v_1, \cdots, v_k$ in G'. We want to show that these vertices also form a hypercycle in G, which means we want to show that for each consecutive set of u' vertices we have $(v_i, \cdots, v_{i+u'-1}) \in E$. Now, we know that for any set of consecutive u' vertices in the hypercycle, there is a hyperedge in G' of the form $(v_i, \cdots, v_{i+u'-1}, v_{i+u'}, \cdots, v_{i+u-1})$. This is due to the fact that, by Lemma 3.4, any k-hypercycle in this parameter regime must use exactly one vertex from each partition. Then, if this edge was added to E', this means the first u' vertices must form a hyperedge in G. Consequently, $v_1, \cdots, v_k$ form a k-hypercycle in G.

Now that our mapping is complete, we can specify how an algorithm A solving (u,k)-hypercycle in $n^{k-\varepsilon}$ time can be used to solve (u',k)-hypercycle in the same runtime. Given an n-node, u'-uniform hypergraph G, we can compute a u-uniform hypergraph G' following our reduction. This requires iterating through all hyperedges in G then creating all $n^{u-u'}$ possible extensions and there can be up to $O(n^{u'})$ hyperedges, so the reduction takes $O(n^u)$ time. Then, since G' also has n nodes, we can run A on G' to determine whether there is a k-hypercycle in $O(n^{k-\varepsilon})$ time.

The total runtime of this algorithm is $O(n^{k-\varepsilon} + n^u)$, which is what we wanted to show. $\square$

3.2.3 Getting the Tight Lower Bound

We finally prove that short hypercycles require brute force:

Theorem 1.2. *Under the $(3,k)$-hyperclique hypothesis, an algorithm for finding (counting) (u,k)-hypercycle for $k \in [u, \gamma_3^{-1}(u)]$ requires $O(n^{k-o(1)})$ time.*

Proof. We want to show that for all $k \in [u, \gamma_3^{-1}(u)]$, finding a (u,k)-hypercycle requires $n^{k-o(1)}$ time. We will do this by applying Theorem 2.1 to go from hardness of $(3,k)$-hyperclique to hardness of $(\gamma_3(k), k)$-hypercycle, then use monotonicity of γ_3 and the self-reducibility lemma to get the desired hardness for (u,k)-hypercycle.

First, observe that the reduction from $(3,k)$-hyperclique to $(\gamma_3(k), k)$-hypercycle preserves the number of vertices in the hypergraph, and this reduction can be computed in time $O(n^{\gamma_3(k)})$. Moreover, this reduction outputs a k-circle-layered hypergraph. Consequently, an algorithm solving $(\gamma_3(k), k)$-hypercycle in $n^{k-\varepsilon}$ time would solve $3,k$-hyperclique in the same runtime. This would violate the $(3,k)$-hyperclique conjecture, so we get a conditional lower bound for $(\gamma_3(k), k)$-hypercycle. Now, since $k \le \gamma_3^{-1}(u)$, Lemma 3.2 tells us that $\gamma_3(k) \le u$. In the case these are equal, we already have the desired lower bound.

If the inequality is strict, we know from Lemma 3.5 that an $O(n^{k-\varepsilon})$ algorithm for (u,k)-hypercycle would imply an algorithm with the same runtime for $(\gamma_3(k), k)$-hypercycle since $k \le \gamma_3^{-1}(u) < 2u$ so it is not a multiple of u. This violates our conditional lower bound, so we can conclude that under the $(3,k)$-hyperclique assumption, finding a (u,k)-hypercycle requires $n^{k-o(1)}$ time. $\square$

From out worst-case to average-case reduction in Lemma 6.11, we can conclude that the average-case version of counting (u,k)-hypercycles also requires brute-force:

Corollary 3.6. *If the $(3,k)$-hyperclique hypothesis holds then counting k-hypercycles in u-uniform Erdős-Rényi hypergraphs when $k \in [u, \gamma_3^{-1}(u)]$ requires $\tilde{O}(n^{k-o(1)})$ time for constant u.*

Proof. If u is constant then so is k. Thus the number of edges and nodes in our subhypergraph is constant.

Recall we are limiting ourselves to $k \in [u, \gamma_3^{-1}(u)]$. By Theorem 1.2 we have that deciding if a k-hypercycle exists in a k-partite graph is $\tilde{O}(n^{k-o(1)})$ hard if the $(3,k)$-hyperclique hypothesis is true.

We can then apply Lemma 6.11 to conclude that counting k-hypercycles in u-uniform Erdős-Rényi hypergraphs is $\tilde{O}(n^{k-o(1)})$ hard as well.

□

4 Beating Brute Force for Longer Hypercycles

In this section we give a new faster algorithm for hypercycle. While the range in which we get tightness in the previous section may have seemed arbitrary, the brute-force lower bound is actually *false* for any longer hypercycle. In particular, we show two different algorithms which leverage matrix multiplication to beat brute force when k is sufficiently bigger than u. One of these algorithms allows improvement over $n^{k-o(1)}$ starting at exactly $k \geq \gamma_3^{-1}(u)+1$, which is the point at which we can no longer prove tightness. The second algorithm exploits the sparsity of its input, and yields even better runtimes than the first when $k \geq 2u-1$.

These algorithms suggest the following relationship between hypercycle length and potential lower bounds. When $k \in [\gamma_3^{-1}(u), 2(u-1)]$, hardness is still dominated by the hypercycle length, but any reasonable lower bounds must account for fast matrix multiplication. Then, when $k \in [2u-1, \infty]$, since the algorithm exploits sparsity and the total number of possible hyperedges is $O(n^u)$, hardness becomes independent of hypercycle length and is dominated by uniformity.

4.1 Faster Algorithm via Triangle-Detection

We now show how matrix multiplication can be used to speed up hypercycle algorithms. The main idea behind this algorithm is that when $k > \gamma_3^{-1}(u)$, it is possible to find three vertex partitions that are sufficiently "spread out" that no hyperedge covers all three partitions. This lets us treat the relationship between these three partitions as edges, not hyperedges. Specifically, we can represent the problem with many 3-partite graphs which contain a triangle if and only if the original hypergraph contains a k-hypercycle. Then, since triangle-detection can be solved in n^ω time using matrix multiplication, we can get some savings.

To present our algorithm, we will need the following lemma.

Lemma 4.1. *Let $\delta = \lceil k/3 \rceil$.*

Let G be an n-vertex, u-uniform k-circle layered hypergraph with vertex partitions $V_1, \cdots, V_k$. Then, for all $k > \gamma_3^{-1}(u)$ and $i \in [1,k]$, every hyperedge covers at most two out of the three vertex sets $V_i, V_{i+\delta}, V_{i+2\delta}$.

Proof. Without loss of generality let $i = 1$.

Recall that $\gamma_3^{-1}(u)$ is the maximum hypercycle length such that the hyperclique reduction outputs uniformity u when starting with a 3-uniform graph. Thus, if $k > \gamma_3^{-1}(u)$, applying the reduction to $(3,k)$-hyperclique yields a uniformity $u' > u$. Note that the three vertex sets $V_1, V_{1+\delta}, V_{1+2\delta}$ are chosen to be maximally spaced out, meaning that if we replace one of them with any other partition, the smallest arc covering the new triple will be shorter than the one covering these three.

We have that $u < k - \lceil k/3 \rceil + 1$, because we are choosing a k above $\gamma_3^{-1}(u)$. Note that any hyperedge which covers all three partitions must cover all the partitions except for those between some pair of partitions. So, the question is what is this gap? First, the gap between V_1 and $V_{1+\delta}$ is $\delta - 1$ partitions. Second, the gap between $V_{1+\delta}$ and $V_{1+2\delta}$ is $\delta - 1$ partitions. Finally, the gap between $V_{1+2\delta}$ and V_1 is $k - 2\delta - 1$ partitions. To cover all three partitions a hyperedge must cover all k partitions except those in the gap. So, no hyperedge covers all three partitions as long as

$$u < \min(k - \delta + 1, 2\delta + 1).$$

We chose δ to be $\lceil k/3 \rceil$. So

$$u < k - \lceil k/3 \rceil + 1 = k - \delta + 1 = k - \lceil k/3 \rceil + 1.$$

Now we need to check the second case how does $u < k - \lceil k/3 \rceil + 1$ compare to $2\lceil k/3 \rceil + 1$? We can quickly work through the three cases. If k is a multiple of 3

$$u < 2k/3 + 1 = 2k/3 + 1.$$

If k is congruent to 1 mod 3 then

$$u < k - (k+2)/3 + 1 = 2k/3 + 1/3 < 2k/3 + 7/3 = 2 \cdot (k+2)/3 + 1.$$

If k is congruent to 2 mod 3 then

$$u < k - (k+1)/3 + 1 = 2k/3 + 2/3 < 2k/3 + 5/3 = 2 \cdot (k+1)/3 + 1.$$

So, regardless of the congruence class of k if $\delta = \lceil k/3 \rceil$ then these partitions are not covered by a single hyperedge.

□

Our algorithm uses this idea to combine brute-force search with known triangle-counting techniques, which yields a saving of $(3 - \omega)$ in the exponent of the brute-force runtime.

Theorem 4.2. *Let G be an n-vertex u-uniform k-circle-layered hypergraph and $k > \gamma_3^{-1}(u)$. Then, there exists an $O(n^{k-3+\omega})$-time algorithm for finding (counting) k-hypercycles in G.*

Proof. Let $\delta = \lceil k/3 \rceil$.

Let $V_1, \cdots, V_k$ be the vertex partitions of G. The algorithm proceeds as follows.

First, fix three vertex sets $V_1, V_{1+\delta}, V_{1+2\delta}$, which we call *triangle partitions*. These satisfy that no hyperedge covers all three partitions, because of Lemma 4.1. Now, we will use brute force to search over the remaining $k-3$ non-triangle partitions. For convince we call the brute force chosen vertex from partition V_i b_i. In particular, for all n^{k-3} possible combinations of vertices, one from each partition, we do the following:

Create a tri-partite graph G' with three vertex sets $T_1 = V_1, T_2 = V_{1+\delta}, T_3 = V_{1+2\delta}$. Then, there will exist an edge between two vertices if and only if there is a hyperpath between them that goes through the current choice of vertices. In other words, we only include an edge between a node in e.g. $v_{\in} T_1$ and $v_2 \in T_2$ if every edge which involves only brute forced nodes b_i, v_1, and v_2 exists in the original hypergraph[1]. This graph can be constructed in $O(n^2)$ time since we need to look at all pairwise combinations of the three vertex sets. Finally, use matrix multiplication to run triangle detection (or counting) on G' which takes $O(n^{\omega})$.

If the algorithm finds a triangle for any of the n^{k-3} graphs, then there is a hypercycle in G and we output YES.

We complete this proof by analyzing the runtime of the algorithm. For each combination of vertices, we take $O(n^2 + n^{\omega})$ time. Since this must be done for $O(n^{k-3})$ combinations, our total runtime is $O(n^{k-3+\omega})$. □

This algorithm shows that for hypercycles which are sufficiently longer than the hyperedge size, it is actually possible to beat brute force. It is worth noting that the algorithm's complexity still increases with k. This leaves open the possibility that we can find reasonably tight lower bounds dependent on k for $k \geq \gamma_3^{-1}(u)$.

Nonetheless, we next show that such a lower bound is only possible for hypercycles up to length $2(u-1)$, since there is an algorithm which only depends on u and is faster for $k \geq 2u-1$.

4.2 A Faster Algorithm for Longer Hypercycles

In this section, we explore a different approach to the hypercycle problem. Once again, we exploit the structure of a k-circle-layered graph. The main idea behind previous algorithms for e.g. weighted cycle is to start at some partition V_1, then for each vertex v_1 in V_1 iterate through $V_2, \cdots, V_k$ while keeping track of whether v_1 has a path to each of these [LWW18]. When going from V_i to V_{i+1}, a node in V_{i+1} will be reachable if it shares an edge with a node in V_i that is reachable from v. This reduces the cycle problem to this reachability subproblem between adjacent partitions.

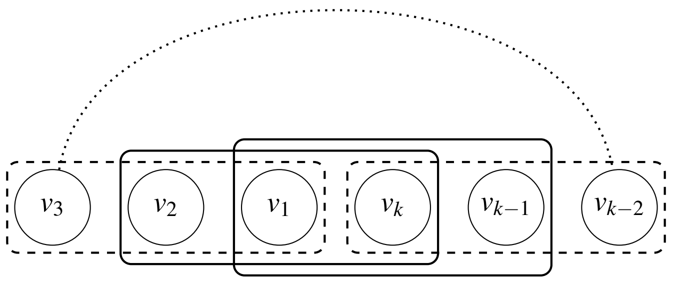

Figure 6: Example with uniformity $u = 3$ to demonstrate why you need $u-1$ nodes. These are the four hyperedges which involve only the vertices $v_1, v_2, v_2, v_k, v_{k-1}$, and v_{k-2}. The two thick hyperedges include at least one vertex from v_1, v_2, v_3 and at least one vertex from v_k, v_{k-1}, v_{k-2}, the dashed hyperedges include only vertices from one side. Note that only v_2, v_1, v_k, and v_{k-1} are involved in the crossover edges. Note that $2 = u-1$ in this context.

We define a more general reachability subproblem, modified in ways that are necessary to use it for hypercycle detection. The key difference is that instead of considering all possible starting vertices for the hypercycle, we have to consider all possible sets of $u-1$ starting vertices. This is due to the fact that at the

[1]For a concrete example consider $u = 3$, $k = 5$, so $\delta = 2$ where the triangle partitions are V_1, V_3, V_5. There will be an edge between v_1 and v_3 iff hyperedges (v_1, b_2, v_3), (b_2, v_3, b_4) exist in the original graph.

end of the hypercycle, we will need the last $u-1$ edges to contain all of these vertices. See Figure 6 for a visual representation of why $u-1$ partitions are needed to finish the cycle.

We define a reachability problem in circle-layered graphs to capture this idea.

4.2.1 Reachability Problems in Uniform Hypergraphs

Intuitively the following problem asks for all hyperpaths that cross $2u-1$ partitions in a $(2u-1)$-circle-layered hypergraph. The output will answer for all tuples of nodes of the first $u-1$ nodes and last $u-1$ nodes if a path exists, or what the minimum weight path is.

Definition 10. (Weighted) u-uniform circle-layered reachability problem ((Weighted) u-CLR) Let $G=(V,E)$ be an n-vertex u-uniform $(2u-1)$-circle-layered (weighted) hypergraph with vertex partitions $V_1,\cdots,V_{2u-1}$. Let $L=V_1\times\cdots\times V_{u-1}$ and $R=V_{u+1}\times\cdots\times V_{2u-1}$ be sets of vertex partitions. Then, the u-uniform circle-layered reachability problem asks for an output of size n^{2u-2}. For all $2u-2$ tuples of nodes $(x_1,\cdots,x_{u-1},y_1,\cdots,y_{u-1})$ where $(x_1,\cdots,x_{u-1})\in L$ and $(y_1,\cdots,y_{u-1})\in R$ you must return the following:

- If unweighted return if there exists a $v\in V_u$ such that edges $(x_1,\cdots x_{u-1},v)$, $(x_2,\cdots x_{u-1},v,y_1)$, …, $(x_{u-1},v,y_1,\cdots,y_{u-2})$, $(v,y_1,\cdots,y_{u-1})$ exist in E. (That is if a hyperpath $x_1,\cdots,x_{u-1},v,y_1,\cdots,y_{u-1}$ exists.)

- If weighted let $w()$ be a function which returns the weight of a hyperedge. Then return

$$min_{v\in V_u}\left(w(x_1,\cdots x_{u-1},v)+w(v,y_1,\cdots,y_{u-1})+\sum_{i\in[2,u-2]}w(x_i,x_{i+1},\ldots,x_{u-1},v,y_1,\ldots,y_{i-1})\right)$$

The output of this problem is an $n^{u-1}\times n^{u-1}$ matrix, which we denote CLR(L,R) (we will also call this output matrix CLR$_{2u-1}(L,R)$). This matrix is binary if unweighted and is over field of the weights if weighted.

For convenience we will also define a second version of this which takes in a CLR$_{k-1}(L,R)$ matrix and outputs another reach-ability matrix, CLR$_k(L,R)$, for a graph with one more circle layer. The key intuition for why we define the problem this way is that if you have reachability information from the first $u-1$ nodes to the last $u-1$ nodes you can extend this to another partition because no hyperedge in the path will need information from the 'middle' nodes as the hyperedges are of uniformity u.

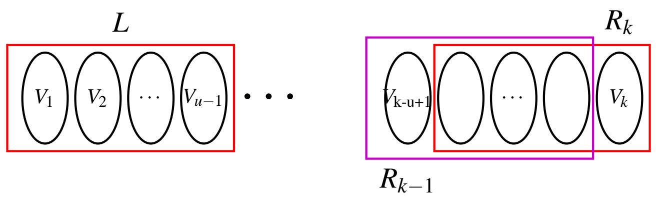

Figure 7: The (u,k)-ECLR problem.

Definition 11. (Weighted) (k,u)-extension uniform circle-layered reachability problem ((Weighted) (u,k)-ECLR) Let $G=(V,E)$ be an n-vertex u-uniform k-circle-layered (weighted) hypergraph with vertex partitions $V_1, \cdots, V_k$. Let $L = V_1 \times \cdots \times V_{u-1}$ and $R_k = V_{k-(u-1)+1} \times \cdots \times V_k$ be sets of vertex partitions. Additionally let $\mathrm{CLR}_{k-1}(L, R_{k-1})$ be a reachability matrix which gives (minimum weight) hyperpath reachability from L to $V_{k-(u-1)-1} \times \cdots \times V_{k-1}$.

Then, the (u,k)-ECLR problem asks for an output of size n^{2u-2}. For all $2u-2$ tuples of nodes $(x_1, \cdots, x_{u-1}, y_1, \cdots, y_{u-1})$ where $(x_1, \cdots, x_{u-1}) \in L$ and $(y_1, \cdots, y_{u-1}) \in R$ you must return the following:

- If unweighted return if there exists a hyperpath which starts at nodes $x_1, \cdots, x_{u-1}$ and ends at nodes $y_1, \cdots, y_{u-1}$.
- If weighted then return the minimum weight hyperpath which starts at $x_1, \cdots, x_{u-1}$ and ends at nodes $y_1, \cdots, y_{u-1}$.

The output of this problem is an $n^{u-1} \times n^{u-1}$ matrix, which we denote $\mathrm{CLR}_k(L,R)$. This matrix is binary if unweighted and is over field of the weights if weighted. See Figure 7 for a visual of this problem.

4.2.2 Hypercycle Algorithm from Reachability Algorithm

We now show that a pair of algorithms solving these problems can be used efficiently to solve hypercycle.

Lemma 4.3. *Suppose there exists a $T_1(n)$-time algorithm for u-CLR and a $T_2(n)$-time algorithm for (u,k)-ECLR. Then, there exists an $O(T_1(n) + T_2(n) + n^{2(u-1)})$-time algorithm for finding (counting) k-hypercycles in n-vertex u-uniform k-circle-layered hypergraphs.*

Proof. Let A_1 be the algorithm for u-CLR, A_2 be the algorithm for (u,k)-ECLR and let $V_1, \cdots, V_k$ be the vertex partitions of the k-circle-layered hypergraph. Let L and R_i be as defined in Definition 11.

Note that given the matrix $u\text{-}\mathrm{CLR}_k(L, R_k)$, we can solve hypercycle by "completing the hyperpath" as follows. For each possible sequence $(x_1, \cdots, x_{u-1})$ in L, we iterate through each possible sequence $(y_1, \cdots, y_{u-1})$ in R_k and check its reachability. If the value in the matrix is a 1, we can then check whether all of the hyperedges $(y_1, \cdots, y_{u-1}, x_1), \cdots, (y_{u-1}, x_1, \cdots, x_{u-1})$ exist. If they do, we have found a hypercycle.

Note that this procedure requires $O(1)$ work for each possible pair of sequences, and there are $n^{2(u-1)}$ pairs, so this requires $O(n^{2(u-1)})$ time.

Now we must show how to efficiently obtain $u\text{-}\mathrm{CLR}_k(L, R_k)$ using the algorithms A_1 and A_2. First, we use $A_1(L, R_{2u-1})$ to get $u\text{-}\mathrm{CLR}_{2u-1}(L, R_{2u-1})$ in $T_1(n)$ time. Next, we can repeat the following for $i \in [2u, k]$: Run A_2 on inputs L, R_i and $u\text{-}\mathrm{CLR}_{i-1}(L, R_{i-1})$ to get $u\text{-}\mathrm{CLR}_i(L, R_i)$. Once we get to $i = k$, we can carry out the hyperpath completion mentioned previously and check for hypercycles.

This process requires running A_1 once and A_2 a constant $k - 2u + 1$ number of times, which takes $T_1(n) + (k - 2u + 1)T_2(n) = O(T_1(n) + T_2(n))$ time. Thus, the total runtime of the algorithm is $O(T_1(n) + T_2(n) + n^{2(u-1)})$. □

Note that if $T_1(n)$ and $T_2(n)$ have runtimes independent of k, then so will this algorithm. Inutitively this should be true since one problem is completely characterized by $(2u-1)$-circle-layered graphs, and the other is simply asking about extending the reachability by one layer. In the next subsection, we actually prove this by constructing algorithms for u-CLR and (u,k)-ECLR which will let us prove the following theorem:

Theorem 4.4. *There exists a time-$O(n^{2u-1-(3-\omega)})$ algorithm for finding (counting) k-hypercycles in n-node u-uniform k-circle layered hypergraphs when $k \geq 2u-1$.*

4.2.3 Algorithms for Unweighted Reachability

To achieve the desired runtime, we need algorithms for u-CLR and (u,k)-ECLR running in time $O(n^{2u-1-(3-\omega)})$. We will show that this can be achieved using matrix multiplication in a similar way as done in Theorem 4.2.

We first show how to do this for $k = 2u-1$.

Lemma 4.5. *There exists a time-$O(n^{2u-1-(3-\omega)})$ algorithm for solving u-CLR in n-node u-uniform hypergraphs.*

Proof. The algorithm heavily relies on the fact that any given hyperedge will cover at most two out of the three vertex partitions V_1, V_u, V_{2u-1}. We fix these three, then for each of the n^{2u-1-3} possible combination of vertices from the other $2u-1-3$ vertex partitions, we will create a graph G which captures the reachability between these three partitions.

The graph will have three vertex partitions, corresponding to V_1, V_u, V_{2u-1}, with edges between V_1 and V_u as well as between V_u and V_{2u-1}. We add an edge between any pair of vertices in V_1 and V_u which has a hyperpath going through the current choice of vertices in $V_2, \cdots, V_{u-1}$. Similarly, we add an edge between a pair of vertices in V_u and V_{2u-1} if there is a hyperpath between them going through the current choice of vertices in $V_{u+1}, \cdots, V_{2u-2}$.

Once the graph has been constructed, assume we have two adjacency matrices, one for each pair of vertex sets with edges between them. Then, we can use matrix multiplication to obtain the reachability between V_1 and V_{2u-1} in $O(n^\omega)$ time. Since we do this for n^{2u-1-3} graphs, the total runtime of the algorithm is $O(n^{2u-1-(3-\omega)})$ time. ☐

We now show how to extend the technique to solve reachability when $k > 2u-1$.

Lemma 4.6. *There exists a time-$O(n^{2u-1-(3-\omega)})$ algorithm for solving (u,k)-ECLR in n-node u-uniform hypergraphs.*

Proof. Recall that we want to produce the matrix u-CLR$_k(L, R_k)$ given L, R_k and the reachability matrix u-CLR$_{k-1}(L, R_{k-1})$, where L, R_{k-1}, R_k all consist of n^{u-1} tuples of $(u-1)$ vertices. The algorithm essentially creates a reachability matrix from R_{k-1} to R_k, then combines this with u-CLR$_{k-1}(L, R_{k-1})$ to extend the hyperpath.

To create the reachability matrix A from R_{k-1} to R_k, we must look at all possible pairs of $(u-1)$-tuples in $R_{k-1} \times R_k$. If the two tuples have the form $(x, v_2, \cdots, v_{u-1})$ and $(v_2, \cdots, v_{u-1}, y)$ then we check whether the hyperedge $(x, v_2, \cdots, v_{u-1}, y) \in E$. If it is, then we set the corresponding entry in A to 1. Note that doing this for all possible pairs takes $n^{2(u-1)}$ time.

To obtain u-CLR$_k(L, R_k)$ from A and u-CLR$_{k-1}(L, R_{k-1})$ we use the same approach as in the previous lemma. We will fix three vertex sets $V_1, V_{k-(u-1)}, V_k$ and then create 2^{2u-1-3} different graphs by taking all possible combinations of vertices in $V_2, \cdots, V_{u-1}, V_{k-u}, \cdots, V_{k-1}$. For each combination, the graph will have three vertex partitions corresponding to $V_1, V_{k-(u-1)}, V_k$ and edges will be added between V_1 and $V_{k-(u-1)}$ as well as between $V_{k-(u-1)}$ and V_k. Crucially, this can be done because k is big enough so that any given hyperedge will cover at most two of these To determine reachability between V_1 and $V_{k-(u-1)}$, we can check the corresponding entry in u-CLR$_{k-1}(L, R_{k-1})$. Even though this matrix contains a stricter condition, this condition is necessary for the reachability to V_k, so we enforce it with this connection. To determine reachability between $V_{k-(u-1)}$ and V_k, we can use the corresponding entries in A.

Once we have this graph, we can use matrix multiplication to determine 2-hop reachability and fill out the appropriate entry in u-CLR$_k(L, R_k)$ in $O(n^\omega)$ time. Since we do this for all n^{2u-1-3} combinations of vertices, the total runtime of the algorithm is $O(n^{2u-1-(3-\omega)})$. □

We can now prove that k-hypercycle can be solved in $O(n^{2u-1-(3-\omega)})$ for sufficiently large k.

Theorem 4.4. *There exists a time-$O(n^{2u-1-(3-\omega)})$ algorithm for finding (counting) k-hypercycles in n-node u-uniform k-circle layered hypergraphs when $k \geq 2u-1$.*

Proof. Recall from Lemma 4.3 that given a $T_1(n)$-time algorithm for u-CLR and a $T_2(n)$-time algorithm for (u,k)-ECLR we can solve k-hypercycle in time $O(n^{2(u-1)} + T_1(n) + T_2(n))$. Applying Lemma 4.6, we know we can set $T_1(n) = O(n^{2u-1-(3-\omega)})$. From Lemma 4.5 we get $T_2(n) = O(n^{2u-1-(3-\omega)})$.

Combining these, we conclude that k-hypercycle in u-uniform hypergraphs can be solved in $O(n^{2u-1-(3-\omega)})$ time when $k \geq 2u-1$. □

We can now prove our desired theorem.

Theorem 1.3. *There exists a time-$\tilde{O}(k^k(n+m+n^{2u-1-(3-\omega)}))$ algorithm for finding k-hypercycles in any n-node hypergraph G when $k \geq 2u-1$.*

Proof. Using the color-coding approach from [LWW18, Lemma 2.2] we can randomly color the vertices in G to obtain a k-circle-layered hypergraph. Then, we can run our time-$O(n^{2u-1-(3-\omega)})$ algorithm on this new hypergraph. Finally, this process can be repeated $k^k \log n$ times to boost the success probability, giving us our runtime of $\tilde{O}(k^k(n+m+n^{2u-1-(3-\omega)}))$. □

5 Tight Results for Min-Weight Hypercycle

In this section we present the tight matching upper and lower bounds for minimum hypercycle represented in Figure 1. We start by providing the algorithm for min-weighted hypercycle and then prove the lower bounds.

Notably, we demonstrate that the best algorithm for weighted k-hypercycle for $k \in [u+1, 2u-1]$ is the naive algorithm. Our algorithm relies on tracking the first $u-1$ nodes of a hyperpath and the last $u-1$ nodes of a hyperpath. This lower bound shows that doing so is, in some sense, required. The tight upper and lower bound gives credence to the intuition that when thinking about hypercycles you should think of sets of $(u-1)$ nodes as being analogous to single nodes in a normal graph with uniformity 2. Furthermore, this suggests that, perhaps surprisingly, for $k \leq 2u-1$ the k-hypercycle problem requires the same running time as the k-hyperclique problem in a u uniform graph.

5.1 Algorithms for Minimum Hypercycle

We will present two algorithms in this section. The first is the naive algorithm, presented for completeness. The second uses the definitions of u-CLR and (u,k)-ECLR from the previous section (see Definitions 10 and 11) and demonstrates fast algorithms for them. We can then use the (u,k)-ECLR algorithm to solve the k-hypercycle problem for $k \geq 2u-1$.

5.1.1 Algorithm for k less than 2u-1

This is the totally naive algorithm for the problem. We simply try all possible orders of nodes and check if they form a hypercycle.

Lemma 5.1. *The minimum k-hypercycle problem in a u-uniform hypergraph can be solved in time $O(k \cdot n^k)$, when k is constant this is $O(n^k)$.*

Proof. Say we are given a hypergraph G with vertex set V and hyperedge set E.

For all choices of nodes $v_1, \dots, v_k$ where $v_i \in V$ and $v_i \neq v_j$ if $i \neq j$ we check if they form a hypercycle and if it is a hypercycle we compute its weight. Specifically for all $i \in [1,k]$ we check if $(v_i, v_{(i+1 \mod k)}, \dots, v_{(i+k-1 \mod k)}) \in E$ and if they all exist we sum the associated weights. We track the minimum weight we have seen so far. This takes $O(k)$ time for each choice of nodes. There are $O(n^k)$ choices of nodes.

This gives a running time of $O(kn^k)$. Note that this is $O(n^k)$ if k is constant. □

5.1.2 Algorithm for k at least 2u-1

We start with the base case, which can be solved with the naive algorithm. We present an algorithm which gives a speedup if the graph is sparse[2].

Lemma 5.2. *The weighted u-CLR problem has a $O(u \cdot n^{u-1}|E| + n^{2u-2})$ algorithm, where $|E|$ is the number of hyperedges in the graph. Note this is also $O(n^{2u-1})$.*

Proof. Recall the input is a graph $G = (V,E)$ which is a n-vertex u-uniform $(2u-1)$-circle-layered (weighted) hypergraph with vertex partitions $V_1, \cdots, V_{2u-1}$. Recall that $L = V_1 \times \cdots \times V_{u-1}$ and $R = V_{u+1} \times \cdots \times V_{2u-1}$ are sets of vertex partitions defined in the problem.

We first initialize the output matrix $\text{CLR}_{2u-1}(L,R)$ with all entries being ∞. This takes $O(n^{2u-2})$ time. Now in $O(n^{u-1} \cdot |E|)$ time we will consider all tuples in L combined with all hyperedges which span partitions $V_u, \dots, V_{2u-1}$. A combination of L and such a hyperedge specifies all nodes in a path, in $O(u)$ time check the length of the path. We then lookup the corresponding entry in $\text{CLR}_{2u-1}(L,R)$ and overwrite the existing entry if this new path length is shorter.

Note that $|E| = O(n^u)$, there are only $O(n^u)$ possible sets of u nodes. Thus the overall running time is $O(n^{2u-1})$. □

The key for this next lemma is that a minimum hyperpath of length k can be computed quickly given minimum reachability information for hyperpaths of length $k-1$. Specifically, by tracking the $u-1$ nodes on the 'end' of the $(k-1)$-hyperpath we can extend by one node as the only hyperedge the new node is involved in only concern the last $u-1$ nodes of the previous hyperpath. This lets us extend our path information.

Lemma 5.3. *The weighted (u,k)-ECLR problem has a $O(n^{2u-2} + n^{u-1} \cdot |E|)$ algorithm. Note this running time is also $O(n^{2u-1})$.*

[2]We will note without proof that a $n^{u-1}|E|$ algorithm exists for finding the minimum hypercycle if instead of keeping a n^{2u-2} sized output matrix you replace that matrix with a hashtable and ask only for the minimum between those sets of nodes in L and R which have a path.

Proof. First, note that given the information of the shortest hyperpaths from all tuples of nodes in $L = V_1 \times \cdots \times V_{u-1}$ to $R_{k-1} = V_{k-(u-1)} \times \cdots \times V_{k-1}$ the shortest hyperpath from L to R_k can be computed with just the hyperedges which span $V_{k-(u-1)}, \ldots, V_k$. There is optimal substructure, given that we are tracking the first and last $u-1$ nodes of the path. This is because the last new edge we want to add on to the path will depend on the last $u-1$ nodes.

Now, given this we want to compute CLR_k from CLR_{k-1} and the hyperedges which span $V_{k-(u-1)}, \ldots, V_k$ in $O(n^{u-1} \cdot |E| + n^{2u-2})$ time. We will call the set of hyperedges which span $V_{k-(u-1)}, \ldots, V_k$ E_k for convenience.

Initialize $\text{CLR}_k(L,R)$ in $O(n^{2u-2})$ time with every entry starting as ∞ distance. For a given entry $(v_1, \ldots, v_{u-1}) \in L$ and a hyperedge $e = (v_{k-(u-1)}, \ldots, v_k) \in E_k$ we will update CLR_k. For convenience we will name the left tuple $\ell = (v_1, \ldots, v_{u-1})$, and both right tuples $r_{k-1} = (v_{k-(u-1)}, \ldots, v_{k-1})$ and $r_k = (v_{k-(u-1)+1}, \ldots, v_k)$. We will let $w(e)$ be the weight of the edge we are considering. Given a previous value of an entry of CLR_k and our new value computed from CLR_{k-1} and an edge we update the new value to:

$$CLR_k(\ell, r_k) \leftarrow \min\left(CLR_k(\ell, r_k), CLR_{k-1}(\ell, r_{k-1}) + w(e)\right).$$

After performing these updates for every edge in E_k CLR_k will be populated with the lengths of the shortest paths from L to R_k.

The total running time includes $O(n^{2u-2})$ for initializing the output and $O(n^{u-1} \cdot |E|)$ for updating for every $\ell \in L$ and $e \in E_k$. This gives a running time of $O(n^{2u-2} + n^{u-1} \cdot |E|)$. Because $|E| = o(n^u)$ we can bound this as $O(n^{2u-1})$. ☐

Now we will use these split up problems solve the hypercycle problem. Note that the problems as they have been split are basically a dynamic programming setup.

Lemma 5.4. *The minimum k-hypercycle problem in a u-uniform graph has a* $O(k^k \cdot u \cdot k \cdot (n^{u-1}|E| + n^{2u-2}))$ *algorithm* $k \geq 2u-1$ *which succeeds with probability at least* $2/3$. *Note that* $|E| = O(n^u)$ *and thus the run time is also* $O(k^k \cdot u \cdot k \cdot n^{2u-1})$

Proof. The first step we will need to do is color-coding. We are given a graph with no layers and will turn the graph into a layered graph by randomly partitioning. We will take the minimum cycle we return over all possible partitionings.

The procedure works as follows, we take each node in our graph and randomly assign it to one of the partitions $V_1, \ldots, V_k$. We only include hyperedges that have exactly one node from $V_i, \ldots, V_{i+u \mod u}$ (that is we only include the circle edges). Note that we are simply deleting hypercycles, any hypercycle that exists in this graph also existed in the original graph. Further note that if a particular hypercycle exists with nodes $a_1 \rightarrow a_2 \rightarrow \cdots \rightarrow a_k$ then if we happen to assign a_i to V_i the hypercycle will exist in our new graph. So, with probability at least k^{-k} such a cycle exists. If we repeat this procedure $10 + 10 \cdot k^{-k}$ times the chance we successfully include the minimum cycle will be at least $2/3$.

First we will show that we can compute $\text{CLR}_k(L,R)$ in $O(k \cdot u \cdot (n^{u-1}|E| + n^{2u-2}))$ time. To compute $\text{CLR}_{2u-1}(L,R)$ we simply use the algorithm from Lemma 5.2 which takes $O(u \cdot (n^{u-1}|E| + n^{2u-2}))$ time. From there given $\text{CLR}_i(L,R)$ we can compute $\text{CLR}_{i+1}(L,R)$ in time $O(u \cdot n^{2u-1})$ using the algorithm from Lemma 5.3. We need to run this extension procedure $k - 2u - 1$ times which is $O(k)$ giving us a bound of $O(k \cdot u \cdot (n^{u-1}|E| + n^{2u-2}))$ time.

Given $\mathrm{CLR}_k(L,R)$ we will now find the minimum hypercycle. Note that the minimum hypercycle which starts on nodes $v_1,\dots,v_{u-1}$ and ends on nodes $v_{k-u+2},\dots,v_k$ has a length of the minimum hyperpath which starts and ends on those same nodes plus the $u-1$ edges which span between partitions V_{k-u+2} to V_{u-1}. Which means the minimum hypercycle starting on nodes $v_1,\dots,v_{u-1}$ and ends on nodes $v_{k-u+2},\dots,v_k$ has length

$$CLR_k(L,R)[v_1,\dots,v_{u-1},v_{k-u+2},\dots,v_k] + \sum_{i\in[0,u-2]} (v_{u-1-i},\dots,v_1,v_k,\dots,v_{k-i}).$$

Note that given $\mathrm{CLR}_k(L,R)$ this computation takes time $O(n^{2u-2}\cdot u)$ to complete for all possible choices of nodes $v_1,\dots,v_{u-1},v_{k-u+2},\dots,v_k$. We can track the minimum cycle over all of these options and return that value.

In total for each random color coding we take time $O(u\cdot(n^{u-1}|E|+n^{2u-2})+k\cdot u\cdot(n^{u-1}|E|+n^{2u-2})+n^{2u-2}\cdot u)$ which is $O(k\cdot u\cdot(n^{u-1}|E|+n^{2u-2}))$ time. If we return the minimum cycle length we returned over all $10+10\cdot k^k$ random color codings we will return the minimum cycle length in the original graph with probability at least $2/3$. This gives a total time of $O(k^k\cdot k\cdot u\cdot(n^{u-1}|E|+n^{2u-2}))$. □

5.1.3 Putting Both Algorithms Together

We can now do the very simple operation of using the faster algorithm in the appropriate situation.

Theorem 5.5. *An algorithm for finding the minimum k-hypercycle in a u-uniform hypergraph with n nodes with probability at least* $2/3$ *exists which runs in* $O\left(\min(n^k,n^{2u-1})\right)$ *time for constant k.*

Proof. For $k<2u-1$ we apply Lemma 5.1 to get a $O(n^k)$ algorithm.

Note that if k is constant then so is u. For $k\geq 2u-1$ we apply Lemma 5.4 getting a $O(n^{2u-1})$ algorithm. □

5.2 Lower-Bounds for Minimum Hypercycle

Corollary 5.6. *Let G be a* 2*-uniform hypergraph on n vertices V, partitioned into k parts* $V_1,\dots,V_k$.

Let $\gamma_2(k)=k-\lceil k/2\rceil+1$.

In $O(n^{\gamma_2(k)})$ *time we can create a* $\gamma_2(k)$*-uniform hypergraph* G' *on the same node set V as G, so that* G' *contains an k-hypercycle if and only if G contains an k-hyperclique with one node from each* V_i.

If G has weights on its hyperedges in the range $[-W,W]$*, then one can also assign weights to the hyperedges of* G' *so that a minimum weight k-hypercycle in* G' *corresponds to a minimum weight k-hyperclique in G and every edge in the hyperclique has weight between* $[-\binom{\gamma_2(k)}{2}W,\binom{\gamma_2(k)}{2}W]$*. Notably,* $\binom{\gamma_2(k)}{2}\leq O(k^2)$.

Proof. Simply plugging in $u=2$ to Theorem 2.1 (originally from [LWW18]). □

5.2.1 Odd Cycle Lengths

We can now consider what value of k corresponds to a given value of u. That is, if $u=k-\lceil k/2\rceil+1$ given u what is the k we should consider? Consider $k=2\ell+1$ then note that $u=2\ell+1-\ell-1+1=\ell+1$. So, for a given u we should consider $\ell=u-1$ which corresponds to $k=2u-2+1=2u-1$.

Corollary 5.7. *Let G be a 2-uniform hypergraph on n vertices V, partitioned into $k = 2u-1$ parts $V_1,\dots,V_{2u-1}$. Note that $u = k - \lceil k/2 \rceil + 1$.*

In $O(n^u)$ time we can create a u-uniform hypergraph G' on the same node set V as G, so that G' contains an $(2u-1)$-hypercycle if and only if G contains an k-hyperclique with one node from each V_i.

If G has weights on its hyperedges in the range $[-W,W]$, then one can also assign weights to the hyperedges of G' so that a minimum weight k-hypercycle in G' corresponds to a minimum weight $(2u-1)$-hyperclique in G and every edge in the hyperclique has weight between $[-\binom{u}{2}W, \binom{u}{2}W]$. Notably, $\binom{u}{2} \le O(u^2)$.

Proof. We simply plug in $k = 2u-1$ and this makes $\gamma_2(k) = u$. □

This gives an immediate implication

Corollary 5.8. *The minimum $(2u-1,u)$-hypercycle problem requires $n^{2u-1-o(1)}$ time if the minimum $(2u-1)$-clique hypothesis holds.*

Proof. The minimum $(2u-1)$-clique hypothesis states that finding the weight of the minimum $(2u-1)$-clique requires $n^{2u-1-o(1)}$ time. Using the reduction from Corollary 5.7 which takes $O(n^u)$ time we can show that a $T(n)$ time algorithm for the minimum $(2u-1)$-hypercycle problem produces a $O(T(n)+n^u)$ time algorithm for minimum $(2u-1)$-clique. So the minimum $(2u-1)$-hypercycle problem requires $n^{2u-1-o(1)}$ time. □

5.2.2 Even Cycle Lengths

Corollary 5.9. *Let G be a 2-uniform hypergraph on n vertices V, partitioned into $k = 2u-2$ parts $V_1,\dots,V_{2u-2}$. Note that $u = 2(u-1) - \lceil 2(u-1)/2 \rceil + 1 = u-1+1$.*

In $O(n^u)$ time we can create a u-uniform hypergraph G' on the same node set V as G, so that G' contains an $(2u-2)$-hypercycle if and only if G contains an k-hyperclique with one node from each V_i.

If G has weights on its hyperedges in the range $[-W,W]$, then one can also assign weights to the hyperedges of G' so that a minimum weight k-hypercycle in G' corresponds to a minimum weight $(2u-1)$-hyperclique in G and every edge in the hyperclique has weight between $[-\binom{u}{2}W, \binom{u}{2}W]$. Notably, $\binom{u}{2} \le O(u^2)$.

Proof. We simply plug in $k = 2u-2$ and this makes $\gamma_2(k) = u$. □

This gives an immediate implication

Corollary 5.10. *The minimum $(2u-2)$-hypercycle problem requires $n^{2u-2-o(1)}$ time if the minimum $(2u-2)$-clique hypothesis holds in a graph with uniformity u.*

Proof. The minimum $(2u-2)$-clique hypothesis states that finding the weight of the minimum $(2u-2)$-clique requires $n^{2u-2-o(1)}$ time. Using the reduction from Corollary 5.9 which takes $O(n^u)$ time we can show that a $T(n)$ time algorithm for the minimum $(2u-2)$-hypercycle problem produces a $O(T(n)+n^u)$ time algorithm for minimum $(2u-2)$-clique. So the minimum $(2u-2)$-hypercycle problem requires $n^{2u-2-o(1)}$ time. □

5.2.3 Combining to Produce Lower Bound

To prove the same statement for shorter cycles we will use the following strategy. Consider, for example $k = 2u-3$. We can show hypercycles of this length are hard when the uniformity is $u-1$. We will show that if the problem is hard with smaller uniformity then it is hard for larger uniformity. Together this will let us show hardness for smaller k for uniformity u.

The intuition for this next lemma is that a $u' > u$ hypercycle is more constraining than a u hypercycle. Notably we have edges which are 'longer'. So, we can simply include all possible extensions of edges to compute the original problem. We use color-coding to make our graph k-circle-layered to make the proof more straightforward.

Lemma 5.11. *Let $k^k = n^{o(1)}$. If the k-hypercycle problem is $n^{k-o(1)}$ hard for some uniformity $u < k$ then the k-hypercycle problem is $n^{k-o(1)}$ hard for all uniformities u' where $2u' > k > u' \geq u$.*

Proof. We will take a given hypergraph, G_o, with uniformity u and transform it into a circle layered graph G. We will do this process randomly where if no hypercycles exist in G then none exist in G_o. Also, there is at least a $1/k^k$ probability that a hypercycle exists in G if one existed in G_o. To do this we will use color-coding. We randomly assign each node in the graph to one of the partitions $V_1, \ldots, V_k$. Then we delete any edge that doesn't span from $V_i, V_{i+1 \mod k}, \ldots, V_{i+u-1 \mod k}$. As we are deleting edges no new hypercycles will exist. If a hypercycle existed in the original graph in the order $v_1, \ldots, v_k$ then it will exist in G if v_i is assigned to V_i for all i (any rotation of this assignment will also work) so the probability the cycle exists in G_o is at least $1/k^k$. So, we can repeat this process $k^k \lg^2(n)$ times to solve the origional problem with high probability. Note that we can consider hypercycles which must respect the k-circle-layered graph structure, that is, must have exactly one node in each partition.

The core idea is that we will take an edge $(v_i, ..., v_{i+u-1}) \in G$ and create $n^{u'-u}$ edges from it one for all $(v_i, ..., v_{i+u-1}, \ldots v_{i+u'-1})$ for all $v_j \in V_j$ where $j \in [i+u, i+u'-1]$ which we put in G'. This creates at most $n^{u'}$ edges which preserves hardness. We will show that using this method iff a k-cycle exists in G then at least one hypercycle exists in G'. Note that because we are in a circle-layered hypergraph and k is not a multiple of u' for there to be a k-hypercycle the edges must include exactly one edge from V_i to $V_{i+u'-1 \mod k}$ for each i.

If $v_1, \ldots, v_k$ is a hypercycle in G then there is a hypercycle in G' which is formed by the k hyperedges:

$$(v_i, \ldots, v_{i+u'-1 \mod k}).$$

These will exist in G' because the edges

$$(v_i, \ldots, v_{i+u-1 \mod k})$$

exist in G and we add all completions of the $u'-u$ nodes onto edges.

If $v_1, \ldots, v_k$ is a hypercycle in G' then the following hyperedges must exist in G':

$$(v_i, \ldots, v_{i+u'-1 \mod k}).$$

Further, note that we only add such an edge spanning V_i to $V_{i+u'-1 \mod k}$ when $(v_i, \ldots, v_{i+u-1 \mod k}) \in G$.

So, a hypercycle in G' corresponds to a single hypercycle in G over the same nodes. Further note that a single hypercycle in G has only one completion in G' (because the nodes are fixed). Thus the count of the number of hypercycles in G and G' will be be the same. So, there is a one-to-one correspondence between hypercycles which respect the circle layered graph (have one node from each partition). □

Now we can combine our hardness results for $2u-1$ and $2u-2$ hypercycle with the above lemma to show hardness for k-hypercycle when k is small.

Theorem 1.1. *If the minimum k-clique hypothesis holds then the minimum k-hypercycle problem in a u-uniform hypergraph requires $n^{k-o(1)}$ time for $k \in [u+1, 2u-1]$.*

Proof. From Corollary 5.8 we have that minimum $(2\ell-1)$-hypercycle requires $n^{2\ell-1-o(1)}$ time in a graph with uniformity ℓ if the minimum $(2\ell-1)$-clique hypothesis holds. Then applying Lemma 5.11 we have that if $2\ell-1 > u \geq \ell$ then $(2\ell-1)$-hypercycle requires $n^{2\ell-1-o(1)}$ time in a u uniform graph. We can re-state this constraint in terms of $k = 2\ell-1$ as k-hypercycle requires $n^{k-o(1)}$ time if $k > u \geq (k+1)/2$. This corresponds to odd k where $k \geq u+1$ and $2u-1 \geq k$ which gives us the range of odd k in $[u+1, 2u-1]$.

We can then apply Corollary 5.10 which states that minimum $(2\ell-2)$-hypercycle requires $n^{2\ell-2-o(1)}$ time in a graph with uniformity ℓ if the minimum $(2\ell-2)$-clique hypothesis holds. We can re-state this constraint in terms of $k = 2\ell-2$ as k-hypercycle requires $n^{k-o(1)}$ time if $k > u \geq (k+2)/2$. Which says that for even k where $k \geq u+1$ and $2u-2 \geq k$ the problem is $n^{k-o(1)}$ hard.

Combining the even and odd cases we get that all for all k where $k \in [u+1, 2u-1]$ the minimum k-hypercycle problem in a u-uniform graph requires $n^{k-o(1)}$ time. □

6 Worst-Case to Average-Case Reductions for Counting sub-Hypergraphs

6.1 Overview of Approach

In this section, we aim to give results on worst-case to average-case reductions for counting instances of a subhypergraph. We do this with an extension of the Inclusion-Edgesclusion technique (among others) introduced in [DLW20]. In fact, we show here that the result in [DLW20] can be extended to hypergraphs with some modifications to the approach. We begin with some definitions for the problems in question.

Definition 12. The **counting H subgraphs in an H-partite hypergraph (#H)** problem takes as input a hypergraph H and a H-partite n-node graph G with the vertices partitioned into k components, $V_1, \ldots, V_k$, and asks for the count of the number of (non-induced) subgraphs of G that have exactly one node from each of the k partitions and contain the hypergraph H.

Definition 13. The **uniform counting H subgraphs in an H-partite hypergraph (U#H)** problem takes as input a hypergraph H and a H-partite n-node graph G with the vertices partitioned into k components, $V_1, \ldots, V_k$, where every hyperedge between any partitions that have edges in H is chosen to exist iid with probability μ. The problem then asks for the count of the number of (non-induced) subgraphs of G that have exactly one node from each of the k partitions and contain the hypergraph H.

Note that both problems only consider G that are H-partite, and that U#H is the uniform distribution over inputs to #H.

Definition 14. The **counting H subgraphs in a k-partite hypergraph (#HK)** problem takes as input a hypergraph H and a k-partite n-node graph G with the vertices partitioned into k components, $V_1, \ldots, V_k$, and asks for the count of the number of (non-induced) subgraphs of G that have exactly k nodes and contain hypergraph H, where each partition contains exactly one node.

Definition 15. The **counting H subgraphs in a Erdős-Rényi hypergraph (#HER)** problem takes as input a hypergraph H and a Erdős-Rényi hypergraph G where every possible hyperedge exists with probability $\frac{1}{b}$, and asks for the count of the number of (non-induced) subgraphs of G that have exactly k nodes and contain hypergraph H.

We now state the main result of Section 6, the formalization of Theorem 1.4.

Theorem 6.1. *Let H have e edges and $k = O(1)$ edges, Let A be an average-case algorithm for counting subgraphs A in Erdős-Rényi hypergraphs with edge probability $1/b$ which takes $T(n)$ time with probability at least $1 - 2^{-2^k} \cdot b^{-2^k} \cdot (\lg(e) \lg\lg(e))^{-\omega(1)}$.*

Then there exists an algorithm A' that runs in time $\tilde{O}(T(n))$ that solves the #HK problem with probability at least $1 - \tilde{O}(2^{\lg^2(n)})$.

Our goal is to do the following chain of reductions:

$$\underbrace{\#\text{HK} \rightarrow \#\text{H}}_{(1)} \rightarrow \underbrace{\text{U\#H} \rightarrow \#\text{HER}}_{(3)} \qquad \overbrace{\#\text{H} \rightarrow \text{U\#H}}^{(2)}$$

Completing the 3 reductions will show the desired result. We will do reduction 2, then reduction 3, then reduction 1 in the following sections.

6.2 Reducing #H to U#H

We begin by defining the notion of a *good low-degree polynomial*, introduced by [DLW20].

Definition 16. [DLW20] Let n be the input size of a problem P, let P return an integer in the range $[0, p-1]$ where p is a prime and $p < n^c$ for some constant c. A good low-degree polynomial is a polynomial f over a finite prime field F_p where:

- If $\vec{I} = b_1, ..., b_n$, then $f(b_1, ..., b_n) = f(\vec{I}) = P(\vec{I})$, where b_i is a zero or a one in the field.
- The function f has degree $d = o(\lg(n)/\lg\lg(n))$.
- The function is strongly $d-$partite, meaning that the inputs can be partitioned into d sets where no monomial contains more than one input that comes from the same set.

Definition 17. Let H be a k-node graph with vertices V_H and G an H-partite n-node hypergraph with vertex set partition $V_1, ..., V_k$. Let E be the set of variables $\{e(v_{a_1}, ..., v_{a_c}) | v_i \in V_i, a_x < a_y \iff x < y\}$ such that e is 1 when the hyperedge between the vertices exists in G and 0 when it does not. Let $h(v_1, ..., v_k)$ be a function that multiplies all corresponding e for hyperedges in H for the selection of vertices in the input. Define f as follows:

$$f(E) = \sum_{v_1 \in V_1, ... v_k \in V_k} h(v_1, ..., v_k) \mod p$$

Where p is some prime.

Lemma 6.2. *The function in the above definition returns the output of #H, given that p is a prime in $[2n^k, n^{2k}]$.*

Proof. Consider an arbitrary collection of inputs $v_1, \ldots v_k$ into h. If this collection of vertices indeed contains H, then we see the output must be 1, as every corresponding edge variable in H for that permutation of vertices must have value 1. We then take the sum over all possible selections of vertices to arrive at our count.

Every single possible subgraph H is found and counted in this way, as for every such subgraph, there must be a selection of vertices that contains it. Also, no subgraph is counted more than once, as only one selection of nodes can count any particular subgraph. Since the maximum number of subgraphs is upper bounded by n^k, the prime does not affect the correctness of the calculation. $\square$

Lemma 6.3. *f is a good low-degree polynomial for #H if the number of edges in H is $o(\lg(n)/\lg\lg(n))$.*

Proof. To prove the lemma, we note that f is a polynomial over a prime finite field, and the number of monomials is $O(n^k \cdot k!)$, which is polynomial. By Lemma 6.2, the function returns the same value as #H.

Let $|E_H|$ be the number of edges in H. The function f has degree $|E_H| = O(2^k - 1)$. This is because h multiplies exactly as many variables together as the number of edges it has. We note that when k is constant, the degree is constant.

Finally, f is strongly $|E_H|$-partite. There are $|E_H|$ partitions of edges. f is a sum over calls to h where h takes as input one variable from each edge partition and multiplies all of them. $\square$

We can now apply the following result, also from [DLW20].

Theorem 6.4. *[DLW20] Let μ be a constant such that $\mu \in (0,1)$. Let P be a problem such that a function f exists that is a good low-degree polynomial for P, and let d be the degree of f. Let A be an algorithm that runs in time $T(n)$ such that when $\vec{I}$ is formed by n bits each chosen iid from Ber$[\mu]$:*

$$Pr[A(\vec{I}) = P(\vec{I})] \geq 1 - 1/\omega\left(\lg^d(n)\lg\lg^d(n)\right).$$

Then there is a randomized algorithm B that runs in time $\tilde{O}(n + T(n))$ such that for any $\vec{I} \in \{0,1\}^n$:

$$\Pr[B(\vec{I}) = P(\vec{I})] \geq 1 - O\left(2^{-\lg^2(n)}\right).$$

This immediately gives the following corollaries that finish the reduction.

Corollary 6.5. *Let $d = 2^k$ and $k = o(\sqrt[c]{\lg(n)/\lg\lg(n)})$. If an algorithm exists to solve U#H in time $T(n)$ with probability $1 - 1/\omega(\lg^d(n)\lg\lg^d(n))$, then an algorithm exists to solve #H in time $\tilde{O}(T(n) + n^2)$ with probability at least $1 - O\left(2^{-\lg^2(n)}\right)$.*

Corollary 6.6. *Let H be such that $|E_H| = o(\lg(n)/\lg\lg(n))$ and define $d = |E_H|$. If an algorithm exists to solve U#H in time $T(n)$ with probability $1 - 1/\omega(\lg^d(n)\lg\lg^d(n))$, then an algorithm exists to solve #H in time $\tilde{O}(T(n) + n^2)$ with probability at least $1 - O\left(2^{-\lg^2(n)}\right)$.*

6.3 Reducing U#H to Average-Case Erdős-Rényi

We now desire to reduce U#H to Average-Case Erdős-Rényi - meaning that, we want to show that counting H in an Erdős-Rényi hypergraph can be used to solve U#H. We make a note here that, with more precise counting, the below techniques work for $k = o(\lg \lg n)$, but for ease of discussion, we will proceed assuming that k is constant. We also note that we do not put any restrictions on the size of the hyperedges.

Definition 18. Let G be a k-partite Erdős-Rényi hypergraph with every hyperedge included with probability $1/b$, where b is an integer. Let the vertex partitions of G be $V_1, \ldots, V_k$, and the edge partitions be $E_{a_1,\ldots,a_c}$, where $a_i < a_j \iff i < j$.

Label all hyperedges in $E_{a_1,\ldots,a_c}$ with $\ell \in [1,b]$ as follows: Edges that exist in G are given label 1. The rest of the edges are uniformly assigned labels from $[2,b]$. Let $E^{\ell}_{a_1,\ldots,a_c}$ be the set of all edges of label ℓ between these c vertex sets.

Let $G^{\ell_1,\ldots\ell_{\binom{k}{c}}}$ be the following graph: We select all edges with label ℓ_i from the i^{th} edge partition by lexicographical ordering on the labels of the vertex sets associated with the edge partition. We note that there are then b^{2^k} such hypergraphs, as we have b choices for each distinct edge partition. Define this set of hypergraphs as S_G. We note that, due to symmetry, all hypergraphs in S_G are drawn from the same distribution.

Essentially, we are looking at G through the lens of a complete k-partite hypergraph with labels on the edges, labeling all the edges in G with 1. We then focus on hypergraphs where we choose a label for every edge partition, and take edges from each partition with the corresponding label.

Definition 19. Let G be defined as above. We define a labeled subgraph L of H in G to be a subgraph of H where every vertex is assigned a unique label in $[1,k]$. Define the count of the number of L in G to be the number of not-necessarily induced subgraphs L where every vertex with label ℓ in L comes from V_ℓ in G.

Again, we want to reduce U#H to counting subgraphs in Erdős-Rényi hypergraphs. The main barrier that we must overcome comes from the fact that, in an Erdős-Rényi graph, there exist edges that don't exist in H-partite graphs. This leads to overcounting subgraphs. We solve this problem by creating correlated hypergraphs that individually look Erdős-Rényi, through which we can get the true count of H in the H partite graph.

6.3.1 Counting Small Subgraphs

Our argument will make use of recursion to count labeled H. We begin here by solving the base cases. We define counting labeled H in the same way as outlined in Definition 19.

Lemma 6.7. *Let G be a hypergraph with n nodes, m edges, and k labeled partitions of the vertices, $V_1, \ldots, V_k$ (G is not necessarily k-partite).*

If we have the counts of all labeled subgraphs of H in G of size less than s vertices, we can compute the number of labeled subgraphs in G that are the union of two disconnected labeled subgraphs of H of size s or less in constant time.

Proof. Let one be labeled subgraph L, and the other L'. Given that they share no vertices, we can simply multiply the numbers of the two subgraphs. This clearly takes constant time. □

Lemma 6.8. *Let G be defined as above. We can compute the count of any subgraph H in G with 1 edge or fewer in $\tilde{O}(m)$ time. This applies regardless of whether or not H is labeled.*

Proof. We can count these by iterating over all edges, and by counting the number of ways to select vertices using basic combinatorics. This takes $\tilde{O}(km) = \tilde{O}(m)$ time. □

6.3.2 Recursion

This step forms the main technical difficulty of this counting process. We will use all counts of subgraphs with a smaller number of hyperedges to count those with more hyperedges.

Lemma 6.9. *Let G be a labeled k-partite hypergraph with n nodes per partition.*

Say we are given the counts of the number of subgraphs H in all hypergraphs in S_G.

Additionally, say we are given the counts of all labeled subgraphs of H with $x \in [0,v]$ vertices and $y \in [0,e]$ hyperedges.

Let L be a labeled subgraph of H with v vertices and $e+1$ edges.

Using all of these counts, we can count the number of not-necessarily induced subgraphs L in G in time $O(k! \cdot 2^{2^k} + b^{2^k})$.

The techniques we use in this proof essentially comes from a careful extension of the technique used to prove Lemma 5.9 in [DLW20]. Of course, with there being hyperedges instead of edges, there are key parameters that change (resulting in a slightly different Lemma statement), but the key ideas are the same. For this reason, we leave the proof for this Lemma to Section A.1.

6.3.3 Reducing to Erdős-Rényi

We reduce counting labeled copies of H in a k-partite Erdős-Rényi hypergraph to counting H in Erdős-Rényi hypergraphs. Note that picking a particular labeling solves the problem of U#H, as we can treat labeled hypergraph partitions as being H-partite.

Lemma 6.10. *Let H have e edges and k vertices. Let A be an average-case algorithm for counting unlabeled subgraphs H in k-partite Erdős-Rényi graphs with edge probability $1/b$ which takes $T(n)$ time with probability $1 - \varepsilon/b^{2^k}$.*

The number of labeled copies of subgraph H in k-partite Erdős-Rényi graphs with edge probability $1/b$ can be computed in time $\tilde{O}\left(2^{2^k} \cdot \left(m + k! \cdot 2^{2^k} + b^{2^k}\right) + b^{2^k} \cdot T(n)\right)$ with probability $1 - \varepsilon$.

We note that the technique we use here is slightly simpler than the one used to achieve the analogous statment in [DLW20].

Proof. Define G to be a k-partite Erdős-Rényi hypergraph with edge probability $1/b$. Let S_G and $G^{\ell_1,\ldots\ell_{\binom{k}{c}}}$ be defined as above.

We aim to meet the conditions of Lemma 6.9 for labeled subgraphs of size $v = k$ and $e = 1$. First, we require the counts of the number of subgraphs H in S_G. We call A on each one of the hypergraphs in S_G, again noting that each of them are drawn from the same distribution, including G. We make $b^{2^k - 1}$ such calls.

Next, we require the counts of all labeled subgraphs of H with $x \in [0,k]$ vertices and $y \in [0,1]$ edges. By Lemma 6.8, we can do this in $\tilde{O}(2^{2^k} m)$ time.

By Lemma 6.9, we can now have the counts of any labeled subgraph of size 2, using $O(k! \cdot 2^{2^k} + b^{2^k})$ time. We do this same process for all subgraphs of H, doing subgraphs of fewer edges first, to ensure we have all counts we need. Keep in mind that we do not need to call A again, as the same counts still work.

In the end, we will have made b^{2^k-1} calls to A. By union bound, this gives us $1-\varepsilon$ probability of failure.

We invoke Lemma 6.9 once for every subgraph of H, of which there are 2^{2^k}, giving us the desired runtime. □

Lemma 6.11. *Let H have e edges and k vertices, and let A be an average-case algorithm for counting subgraphs H in Erdős-Rényi graphs where each hyperedge exists with probability $1/b$ which takes $T(n)$ time with probability $1 - 2^{-2k} \cdot b^{2^k} \cdot \left(\log(e)\log\log(e)\right)^{-\omega(1)}$.*

Then, there exists an algorithm to count subgraphs in uniform H-partite graphs in time $\tilde{O}(T(n))$ (U#H) with probability at least $1 - O(2^{-\log^2 n})$.

Proof. We begin with a note that, if we can count labeled subgraphs H in Erdős-Rényi k-partite subgraphs, we can count subgraphs in uniform H-partite graphs with the same success probability. We simply add in the “missing" hyperedges from the partitions not in H in an Erdős-Rényi fashion, taking $\tilde{O}(m)$ time.

Thus, by Lemma 6.10, it suffices to count H in k-partite Erdős-Rényi graphs using A. Beginning with a k-partite Erdős-Rényi graph (G), we need to add random edges within each partition to make it look like a unpartitioned Erdős-Rényi graph. After adding the edges within the partitions with probability $1/b$ (call this resultant graph G'), we note that the count of H in G' that only uses one node from each partition in G is still our answer. We can thus use inclusion exclusion to eliminate the counts of all instances of H that do not use exactly one node from each partition in G. We call A on every subset of the k-partitions, of which there are 2^k, and we perform the inclusion-exclusion calculation. By union bound, the probability of this is at least $1 - (\log(e)\log\log(e))^{-\omega(1)}$.

□

6.4 #HK to #H and the proof of Theorem 6.1

Lemma 6.12. *Let A be an algorithm that solves #H on H-partite G with n vertices and H with $v = k = O(1)$ and $e = O(1)$ edges in time $T(n)$ with probability of success at least $1-\varepsilon$.*

Then there exists an algorithm that solves #HK on k-partite G' with n vertices and the same H that runs in $O(T(n))$ time with probability of success at least $1 - 2^{2^k}\varepsilon$.

Proof. Intuitively, we are using the fact that every instance of H that appears in G' must have come from one labeling of the k partitions that happens to be H-partite. Essentially, we are iterating over all possible labelings of the k-partitions, and adding up all of the counts. However, doing this naively can yield double-counting if there is the right symmetry in the structure of H and the edges in G'. Thus, we iterate over all collections of e of the 2^k edge partitions, using A on the ones that are H-partite. There are at most 2^{2^k} such selections.

It’s clear that each valid instance of H in G' will appear exactly once in one of the edge partition choices, namely, the one that contains each of the edges in that particular H. We run the algorithm once per valid edge partition, giving the desired runtime. Taking a union bound over the probability of failure, we get the desired probability of success.

□

We can now prove the main theorem in this section.

Theorem 6.1. *Let H have e edges and $k = O(1)$ edges, Let A be an average-case algorithm for counting subgraphs A in Erdős-Rényi hypergraphs with edge probability $1/b$ which takes $T(n)$ time with probability at least $1 - 2^{-2^k} \cdot b^{-2^k} \cdot (\lg(e)\lg\lg(e))^{-\omega(1)}$.*

Then there exists an algorithm A' that runs in time $\tilde{O}(T(n))$ that solves the #HK problem with probability at least $1 - \tilde{O}(2^{\lg^2(n)})$.

Proof. This immediately follows from applying Lemma 6.11, Corollary 6.6, and Lemma 6.12. □

7 Applications to Database Problems

The aim of this section is to demonstrate the applicability of the above techniques to solve existing problems in the databases setting. We begin by defining the relevant terms.

A *database* D is a relational structure with a set of objects V, and relations $R_i(\vec{s})$, where $\vec{s} \in V^r$. We refer to r as the *arity* of R_i. A relational term that exists in the database is referred to as a *fact*.

A *conjunctive query* $Q(R_a(\vec{s}_a), \ldots)$ is defined as a conjunction of the relations R_j in its input. The variables in X_i are referred to as *free variables*. The query evaluates to true if there is some assignment of variables in V to the free variables in the sets X_i such that every relation in Q exists in the database. We note that some of the sets X_i may be share free variables. Naturally, every single free variable with the same label must be the same. In this paper, this is the only type of query we consider, and so we will simply refer to them as *queries*.

A *count query* simply answers with the number of unique assignments of variables that are valid for Q. A query contains *self-joins* if it contains more than one copy of the same relation. We say that a query is *self-join-free* if it has no self-joins. For this paper, we assume that the size of the queries is constant.

7.1 Worst case to Average Case reduction for self-join-free count queries

Definition 20. The **self-join-free count query (SCQ)** problem takes as input a database D and a query Q and asks for the count of the number of unique assignments of elements in the domain of D to free variables in Q such that Q is satisfied. Relations in D can be limited to those that appear in Q.

Definition 21. The **uniform self-join-free count query (USCQ)** problem is the same as the SCQ problem, only that the database is such that every fact within a relation exists with some probability μ_{R_i}. This is the average-case version of the SCQ problem.

Much like the reduction we did in Section 6.2, we start by constructing a good low-degree polynomial for SCQ.

Definition 22. Let D be a database with domain V and relations R_i. Let Q be a count query as defined above, where the number of relations in Q is k and the number of free variables is r. Note that $r \le k$. Let $\vec{E}$ be a vector such that every entry corresponds to a potential fact in the database, where it is 1 if the fact is in the database, and 0 otherwise. Let $h(v_1, \ldots, v_r)$ be a function that multiples all corresponding entries for facts in the query for the selection of domain elements v_1 through v_r. Note that this function returns 1 if the selection of variables in the input form a valid response to the query and 0 otherwise. Define f as follows:

$$f(\vec{E}) = \sum_{v_1,\dots,v_r \in V} h(v_1,\dots,v_r) \mod p$$

where p is some prime.

Lemma 7.1. *The above function returns the output of SCQ given that p is a prime in $[2n^k, n^{2k}]$*

Proof. Consider an arbitrary term in the sum, defined by a selection of r elements. As discussed in the definition of f, h correctly identifies when the inputs form a valid response to the query. Also, every valid response to the query corresponds to one selection of free variable assignments, which is represented in only one term in the sum. Thus, acquiring the sum of all of these terms yields the correct count, before taking the modulo.

The number of possible correct queries is upperbounded by the total number of possible assignments to the free variables, which is $n^r \leq n^k$, so the modulo does not affect the count. □

Lemma 7.2. *Let Q be such that $|Q| = O(1)$ and define $d = |Q|$. f is a good low-degree polynomial for SCQ of degree d if the size of the query is bounded by a constant.*

Proof. We recall from Definition 16 that we need to show that f has three properties.

First, by Lemma 7.1, we get the first property.

Second, the degree of the function is bounded by the degree of each of the terms. This is then the number of relations in the query, which is constant, as desired for the second property.

Lastly, the fact that each relation only appears once in the set (the query is self-join free) means that no monomial can ever appear more than once in a term, meaning that it is d-partite, where d is the number of relations in the query. □

We now apply Theorem 6.4 to get the result desired.

Corollary 7.3. *Let Q be such that $|Q| = O(1)$ and define $d = |Q|$. Let the size of the query be constant. If an algorithm exists to solve USCQ in time $T(n)$ with probability $1 - 1/\omega(\lg^d(n)\lg\lg^d(n))$, then an algorithm exists to solve SCQ in time $\tilde{O}(T(n) + n^2)$ with probability at least $1 - O\left(2^{-\lg^2(n)}\right)$.*

Proof. We can apply Lemma 7.2 to use f as a GLDP of degree d for SCQ. Then we can apply Theorem 6.4 for the worst-case to average-case reduction given our GLDP for SCQ. □

8 Conclusion and Open Problems

We have given a worst-case to average-case reduction for counting sub-hypergraphs and for counting database queries. We demonstrate the usefulness of our improved reduction by giving tight lower-bounds for average-case hypercycle counting for short hypercycles.

Additionally, we present new algorithms and new lower bounds for hypercycle in the worst-case. We get tight bounds for the weighted problem of minimum-hypercycle. We get tight bounds for short hypercycle lengths for the unweighted problem. We show a new faster algorithm for k-hypercycle which depends on the running time of fast matrix multiplication. However, these results leave many problems open.

8.1 Unweighted Hypercycle Open Problems

Fundamentally the open problems here represent getting clear answers about the running time of the unweighted hypercycle problem when $k \geq \lambda_3^{-1}(u) + 1$. Some specific open problems which we think would mark progress towards understanding this longer-cycle regime are:

1. When $k = \lambda_3^{-1}(u) + 1$ can you show a lower bound of $n^{\lambda_3^{-1}(u) - o(1)}$ for unweighted k-hypercycle? Note that if you could show this you would be giving a tight lower bound assuming $\omega = 2$.
2. **(Extending the previous question)** When $2u - 1 > k \geq \lambda_3^{-1}(u) + 1$ can you show a lower bound of $n^{k-1-o(1)}$ for unweighted k-hypercycle?
3. Can you give a reduction which shows that if (counting, directed) k-hypercycle in a u-uniform graph requires $T(n)$ time then so does (counting, directed) $(k+1)$-hypercycle? Note that we have a reduction which shows that if directed k-hypercycle in a u-uniform graph requires $T(n)$ time then so does directed $(k+u-1)$-hypercycle (directedness or something similar is needed for hypercycle lengths which are a multiple of u). This generalizes the idea of adding a partition to your graph with a matching which works for $u = 2$.
4. **(Proving the previous point can't be done in general)** Can you show that for some k and u (counting, directed) k-hypercycle in a u-uniform graph requires $T(n)$ time and (counting, directed) $(k+1)$-hypercycle in a u-uniform graph can be solved in $o(T(n))$ time? For example, if you could show for some u that $k = \lambda_3^{-1}(u) + 1$ hypercycle could be solved in $n^{\lambda_3^{-1}(u) - \varepsilon}$ for some $\varepsilon > 0$ this would be satisfied.
5. Can you give a faster algorithm for the unweighted k-hypercycle problem than what we offer in this paper for any k and u?

8.2 Existential Hypercycle Open Problems

There are interesting patterns we have noticed in k-hypercycles in u-uniform hypergraphs which relate to the existence of graphs instead of the existence of algorithms. We note in the introduction that a theorem exists showing that k-hypercycles must exist in sufficiently dense graphs [ABCM15]. However, as that theorem is currently written it doesn't show that e.g. graphs with n^{u-1} edges must have a $(2u)$-hypercycle. We note that when k is not a multiple of u you can create an extremely dense graph with no hypercycles. Specifically, create a u partite with n/u nodes in each partition and add every possible hyperedge which includes one node from each partition. This is metaphorically similar to no odd-cycles existing in a complete bi-partite graph. We will now give some concrete open problems related to this issue of multiple-of-u-hypercycles and density.

1. In $u = 2$ uniform undirected graphs there is an interesting pattern which emerges. A fully dense bipartite graph contains no odd cycles, however, even cycles must exist even in relatively sparse graphs. Specifically, for constant k in a graph with sparsity $\Omega(n^{1+1/k})$ there must exist a $2k$-cycle. How does this relate to hypergraphs of larger uniformity? Hypercycles of length $k = 0 \mod u$ can exist in a u-partite hypergraph, however hypercycles of length $k' \neq 0 \mod u$ do not exist in even a complete u-paritite hypergraph. This looks like it follows a metaphorically similar structure to graphs where

$u = 2$. Certainly it does for $k' \neq 0 \mod u$. However, we have not yet been able to characterize the density at which k-hypercycles are guaranteed to exist when $k = 0 \mod u$. A construction with $\Theta(n^u)$ edges where no k-hypercycle exists would settle the question, as would a proof that a k-hypercycle must exist in any graph with $\Omega(n^{u-\varepsilon})$ hyperedges for some constant $\varepsilon > 0$.

2. To make the above question more concrete: what is the lowest hyperedge density such that a 6-hypercycle is guaranteed to exist in any graph with that density in a 3-uniform hypergraph? Is this density $\Theta(n^3)$? Is this density $\Omega(n^2)$?

3. To make the above question (potentially) easier: You are given a tripartite 3-uniform hypergraph, G, with partitions V_1, V_2, V_3. Furthermore you are guaranteed that for every pair of nodes $(v_i, v_j) \in V_i \times V_j$ there are exactly d edges of the form (v_i, v_j, v_k) where $v_k \in V_k$ and $i \neq j \neq k$. This is a generalization of degree. What is the degree d at which a 6-hypercycle is guaranteed?

4. Given a close reading of the proof of Theorem 1 from 'Tight cycles in hypergraphs'([ABCM15]) can you show that in a u-uniform hypergraph G with at least $|E| \geq \frac{2k}{n}\binom{n}{u}$ hyperedges there must be a k-hypercycle? (This is what would happen if you could set $\delta = 2k/n$ and still have the proof go through.)

5. For even cycles in 2-uniform graphs the longer the cycle the smaller the sparsity at which it is guaranteed to exist. Can something similar be proven for u-uniform graphs in general?

8.3 Counting Color Coding

We would love to say that counting constant sized subhypergraphs in the worst-case is equivalent in hardness up to log factors to counting constant sized subhypergraphs in the average-case. The issue we have here is, surprisingly, in the worst-case. We would like to show that counting a subhypergraph H in a hypergraph is equivalent to counting H in a $|H|$-partite graph. The issue is that standard approaches for this, even in 2-uniform graphs, allow this reduction for *detection* but not for counting. For specific graph structures (notably cliques) there are ways to build this reduction. However, for most graph structures getting the counts to line up seems untenable. Such a reduction would strengthen our results, but, also strengthen the many results which use color-coding a sub-routine.

References

[ABCM15] Peter Allen, Julia Böttcher, Oliver Cooley, and Richard Mycroft. Tight cycles in hypergraphs. *Electronic Notes in Discrete Mathematics*, 49:675–682, 2015. 7, 34, 35

[AKPP18] Peter Allen, Christoph Koch, Olaf Parczyk, and Yury Person. Finding tight hamilton cycles in random hypergraphs faster. In Michael A. Bender, Martin Farach-Colton, and Miguel A. Mosteiro, editors, *LATIN 2018: Theoretical Informatics - 13th Latin American Symposium, Buenos Aires, Argentina, April 16-19, 2018, Proceedings*, volume 10807 of *Lecture Notes in Computer Science*, pages 28–36. Springer, 2018. 7

[AWW14] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2014. 2

[BBB19] Enric Boix-Adserà, Matthew Brennan, and Guy Bresler. The average-case complexity of counting cliques in erdős-rényi hypergraphs. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1256–1280. IEEE Computer Society, 2019. 1, 6, 38

[BCM22] Karl Bringmann, Nofar Carmeli, and Stefan Mengel. Tight fine-grained bounds for direct access on join queries. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '22, page 427–436, New York, NY, USA, 2022. Association for Computing Machinery. 6

[BRSV17] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Average-case fine-grained hardness. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 483–496. ACM, 2017. 1, 6

[BRSV18] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Proofs of work from worst-case assumptions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 789–819. Springer, 2018. 6

[BT17] Arturs Backurs and Christos Tzamos. Improving viterbi is hard: Better runtimes imply faster clique algorithms. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 311–321. PMLR, 2017. 2

[CK21] Nofar Carmeli and Markus Kröll. On the enumeration complexity of unions of conjunctive queries. *ACM Trans. Database Syst.*, 46(2), may 2021. 1, 6

[CS23] Nofar Carmeli and Luc Segoufin. Conjunctive queries with self-joins, towards a fine-grained enumeration complexity analysis. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '23, page 277–289, New York, NY, USA, 2023. Association for Computing Machinery. 6

[CTG$^+$23] Nofar Carmeli, Nikolaos Tziavelis, Wolfgang Gatterbauer, Benny Kimelfeld, and Mirek Riedewald. Tractable orders for direct access to ranked answers of conjunctive queries. *ACM Trans. Database Syst.*, 48(1):1:1–1:45, 2023. 6

[DLW20] Mina Dalirrooyfard, Andrea Lincoln, and Virginia Vassilevska Williams. New techniques for proving fine-grained average-case hardness. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Virtual, November 16 – 19, 2020*. IEEE Computer Society, 2020. 1, 6, 9, 26, 27, 28, 30

[Gol20] Oded Goldreich. On counting \$t\$-cliques mod 2. *Electron. Colloquium Comput. Complex.*, TR20-104, 2020. 6

[HM19] Hao Huang and Jie Ma. On tight cycles in hypergraphs. *SIAM Journal on Discrete Mathematics*, 33(1):230–237, 2019. 7

[LWW18] Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1236–1252. SIAM, 2018. 1, 2, 4, 7, 8, 9, 11, 16, 20, 23

[NPRW23] Hung Ngo, Kirk Pruhs, Atri Rudra, and Virginia Vassilevska Williams. Fine-grained complexity, logic, and query evaluation program. Simons Institute for the Theory of Computing, Workshop Schedule, September 2023. Logic and Algorithms in Database Theory and AI. 7

[WWS23] Yisu Remy Wang, Max Willsey, and Dan Suciu. Free join: Unifying worst-case optimal and traditional joins. *Proc. ACM Manag. Data*, 1(2):150:1–150:23, 2023. 6

[WXXZ23] Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. *CoRR*, abs/2307.07970, 2023. 4

A Discussion

We start with a proof from the main body of the paper and then continue onto general discussion.

A.1 Proof of Recursion Lemma

We provide the proof of Lemma 6.9.

Lemma A.1. *Let G be a labeled k-partite hypergraph with n nodes per partition.*

Say we are given the counts of the number of subgraphs H in all hypergraphs in S_G.

Additionally, say we are given the counts of all labeled subgraphs of H with $x \in [0,v]$ vertices and $y \in [0,e]$ hyperedges.

Let L be a labeled subgraph of H with v vertices and $e+1$ edges.

Using all of these counts, we can count the number of not-necessarily induced subgraphs L in G in time $O(k! \cdot 2^{2^k} + b^{2^k})$.

Proof. Let H have k vertices, and e_H hyperedges. Let L be given as a list of vertices v with labels corresponding to partitions $i_1, \ldots i_v$, and $e+1$ hyperedges between partitions $i_{a_1}, \ldots, i_{a_c}$. Let S_E be the set of all such sets $\{i_{a_1}, \ldots, i_{a_c}\}$. $\overline{S}_E$, then, will be the set of all sets of partitions not in S_E.

We bring our attention back to S_G. Consider the subset of instances in S_G where the hyperedges between partitions in S_E are labeled 1. These are essentially all graphs where the edges between partitions relevant to L are in G. Call this subset $S_G[L]$.

Take the counts of the number of H that appear in all graphs in $S_G[L]$, and call this count $c_{S_G[L]}$. This counts the number of H that appear if the graph G were to have all possible hyperedges between partitions in $\overline{S}_E$, weighted by how many edges in S_E that subgraph uses. If H appears in G where ℓ of its edges are in the $\overline{S}_E$ partitions then it is counted $b^{2^k-1-e-1-\ell}$ times.

Given that L is a labeled subgraph of H, at least one labeling of H will share all $e+1$ hyperedges and v vertices of L. There may be many valid labelings for the hyperedges and vertices not in L.

We want to count all H that have labelings that match the edges in L, and not those that only share some. Here is where we will use the additional counts of small graphs that we are given. For subgraphs that only partially match up with L, let's define their overlap to be L'. L' must have v vertices, and at most e hyperedges. We want to somehow identify the counts of all subgraphs that overlap with any possible L' without having edges in $L-L'$ and remove them from our previous count of $c_{S_G[L]}$.

Define $G_{L,L'}$ to be a hypergraph on k vertices where all edges in $L-L'$ are excluded, and all other hyperedges are included. We define $c_{G_{L,L'}}$ to be the count of the number of subgraphs H that exist in this graph that use all edges in L'. We can do this with brute force using $O(k!)$ time. As we will see later, we will need to do this computation on every possible subgraph of L, of which there are $O(2^{2^k})$.

Let L' have $e_{L'}$ edges, and $c_{L'}$ be the count of labeled L' that exist in G. The count of all subgraphs H which overlap exactly L' that are counted in $c_{S_G[L]}$ is:

$$c_{L'} \cdot n^{k-v} \cdot c_{G_{L,L'}} \cdot b^{2^k-1-e-e_H+e_{L'}}$$

Let's try to break down this value. For every $\hat{L}$ that is counted by $c_{L'}$, we want to count the number of ways to construct H around this. We first have to select the remaining vertices for our H. There are n^{k-v} choices for this. Now that we have a particular selection of k vertices, $c_{G_{L,L'}}$ tells us the number of ways that we can construct H without using hyperedges in $L-L'$. For each one of these constructions of H, $\hat{H}$, we need to count how many times it appears in $c_{S_{G[L]}}$. $\hat{H}$ will only appear in the count in instances of the graph when the edges in $\hat{H}$ match the label that is selected for the partitions. However, for the edges that are not in $\hat{H}$, $\hat{H}$ will be counted regardless of the label selected. So $\hat{H}$ will be counted $b^{2^k-1-e-e_H+e_{L'}}$ times.

We note that this justification works any $L' \subseteq L$. Furthermore, since every possible $\hat{H}$ has to overlap with *some* subgraph of L (keeping in mind that the empty graph is also a subgraph), we can account every single instance represented in $c_{S_G[L]}$ in the following equation:

$$\sum_{L' \subseteq L} c_{L'} \cdot n^{k-v} \cdot c_{G_{L,L'}} \cdot b^{2^k-1-e-e_H+e_{L'}} = c_{S_G[L]}$$

We now have every variable in this equation except for c_L, and we can now solve for it. This computation takes $O(k! \cdot 2^{2^k} + b^{2^k})$ to do, as desired. □

A.2 Why Enumerating and Deciding is Often Easy on Average

To build intuition we will start with easy graph cases. Then we will explain why enumerating or deciding small graph structures is always easy in random graphs. Then we will explain why a similar result holds true for databases.

First, to build intuition, consider the case of triangle. Counting triangles is equivalently hard, up to log factors, in the worst-case and average-case [BBB19]. However, any set of 3 nodes in an Erdős-Rényi graph have a $1/8$ chance of being a triangle. This makes enumerating and deciding if a clique exists easy. To enumerate we can start by taking linear time iterating through possible cliques to build up a backlog. Then after every $O(1)$ possible cliques that have been checked enumerate a new clique. With high probability there will be a saved clique to return. For deciding the existence of a clique one can simply return 'yes' and be correct with probability at least $1-(1-1/8)^{n/3}$.

A.2.1 Hypergraphs

The core ideas of both cases apply to all small graphs. If a subgraph (or subhypergraph) has a constant number of nodes k then any one particular set of k nodes will have a constant probability of being the relevant graph structure. There are at most 2^k edges and hyperedges in the subgraph and the chance that each of those edges exists or doesn't in the requested direction is 2^{-2^k}. While this is an ugly looking relationship, it is constant if k is constant (note if there are fewer edge constraints the relationship is much less negative). Now, given that the graph structure has a constant probability of existing we can say the probability of our subhypergraph existing in an Erdős-Rényi graph is at least

$$1-(1-2^{-2^k})^{n/k}.$$

If k is constant this is $1-2^{-\Theta(n)}$. Enumerating these graph structures will similarly be easy with high probability. The expected number of these graph structures is high and enumerating possible subhypergraphs and returning when you find a subhypergraph which meets the condition is sufficient.

A.2.2 Databases

What about in databases? Well, similarly we have a case where we have a small structure we are looking for. In this case, rows that correspond to a query of interest. Every row exists iid with probability $1/2$, so once again, given a choice of elements the relevant queried rows exist with constant probability. This will, once again, make deciding very easy (simply return yes and you will be correct with high probability). When enumerating we can once again take the approach of taking linear time to build up a large number of instances and then after checking $O(1)$ possible query entries return an answer. If we check $\text{LARGE\_CONSTANT} \cdot 2^{2^{|\text{SIZE\_OF\_QUERY}|}}$ entries each time the probability that we ever run out of instances to return is low.