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

Cheng-Hao Fu
Boston University, MA, USA

Andrea Lincoln
Boston University, MA, USA

Rene Reyes
Boston University, MA, USA

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 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.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Database theory; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Hypergraphs, hypercycles, fine-grained complexity, average-case complexity, databases

Digital Object Identifier 10.4230/LIPIcs.ICALP.2025.81

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2504.18640` [13]

Funding *Rene Reyes*: Supported by the National Science Foundation under Grant No.2209194.

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 [16]. On the practical side, a recent line of work in database theory (initiated by Carmeli and Kröll [9]) 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. [5] has already been applied to hypergraph problems by Boix-Adserà et al. [7], 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 Dalirrooyfard et.al [12] 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 [16]).

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. [1, 4])). *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 [16]). *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, [16] 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 [16] 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

We now outline results for weighted hypercycle, depicted in Figure 1. For details, see the full version. 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 3.** *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 $k \in [u+1, 2u-1]$.*

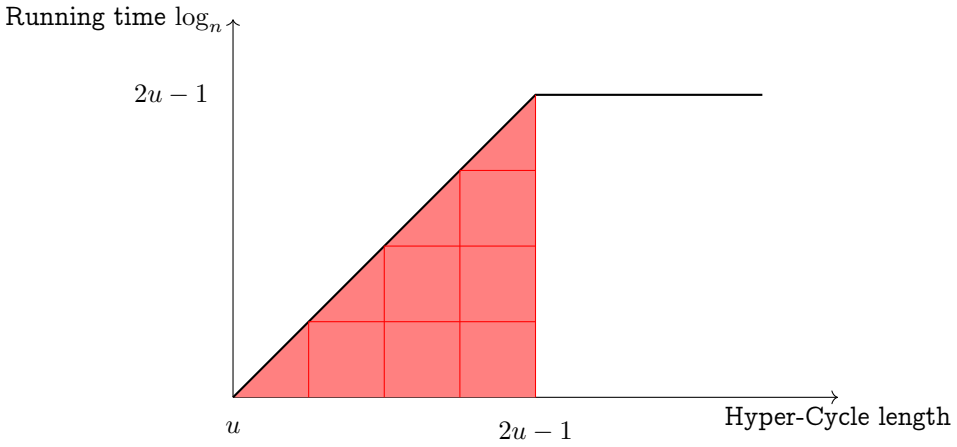

Figure 1 The running time of minimum weight hyper-cycle in a weighted u-uniform hyper-graph. The exponent of the running time is indicated by the black line and the lower bound is matching, as indicated by the hatched red area. Note the running time is $O(n^u)$ for a u length hyper-cycle and $O(n^{2u-1})$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

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 [16] 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 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 fast matrix multiplication to obtain an algorithmic speedup. This algorithm runs in time $\tilde{O}(n^{k-3+\omega})$ where ω is the matrix multiplication constant. Then, when $k \geq 2u - 1$, we can reduce the problem to a more general reachability problem and once again use fast matrix multiplication to obtain an algorithm which runs in time $\tilde{O}(n^{2u-1-(3-\omega)})$. 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 3.1.

▶ **Theorem 4.** *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 5.** *There exists a time-$\tilde{O}(n^{k-3+\omega})$ algorithm for finding k-hypercycles in any n-node u-uniform hypergraph when $k > \gamma_3^{-1}(u)$.*

Finally, when $k \geq 2u - 1$, we get an algorithm with runtime independent of k that also exhibits savings from fast matrix multiplication.

▶ **Theorem 6.** *There exists a time-$\tilde{O}(k^k(n + m + n^{2u-1-(3-\omega)}))$ algorithm for finding k-hypercycles in any n-node u-uniform 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 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

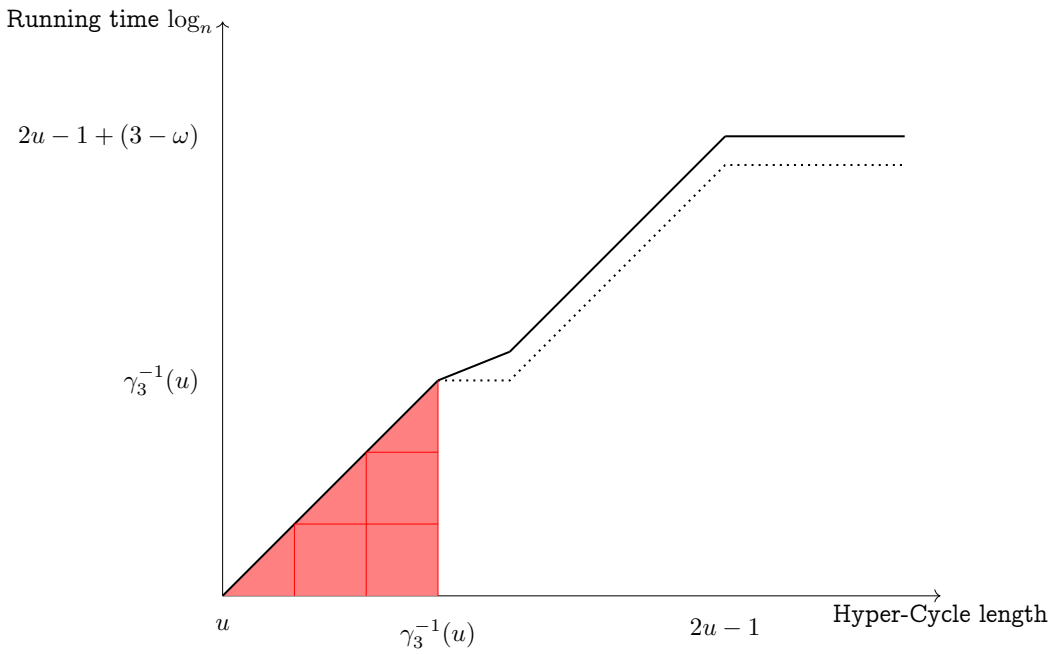

Figure 2 The running time of *unweighted* hyper-cycle in a u-uniform hyper-graph. The exponent of the running time is indicated by the black line. The lower bound is matching for the hatched red area from cycles of length u to $\gamma_3^{-1}(u)$ and the running time is $O(n^k)$ for a k-hypercycle. Then 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 ($\omega < 2.3716$ [19]). 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.

that aren't color-coded we use a new form of inclusion exclusion. Prior work introduced inclusion-edgesculsion [16]. 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 7** (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 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 subhypergraphs. 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 the full version [13]. A table in a database is called a relation. A relation, R_i, is a set of tuples. For example, a relation on $(user\_id, user\_name)$

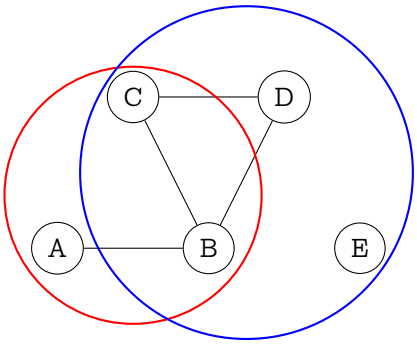

Figure 3 An example of a small subgraph with hyperedges of multiple different sizes. This graph contains hyperedges $\{A, B\}, \{B, C\}, \{B, D\}, \{C, D\}, \{A, B, C\}$, and $\{B, C, D, E\}$. We depict the 3-width edge as a red circle, the 4-width edge as a blue circle, and the 2-width edges as black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

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 the full version [13] 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, which can be found in the full version [13] of the paper. We also explain why *small* enumeration queries will be easy.

1.4 Context and Prior Works

There has been a lot of work on average-case fine-grained complexity. Ball et al. [5] showed that starting from popular fine-grained hypotheses, evaluating certain functions could be shown to be hard on average. Later Ball et al. [6] showed that you can build a fine-grained proof of work using these functions. [7] 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 [14]. Dalirrooyfard et al. [12] 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 [9] use fine-grained hypotheses to show that answering unions of conjunctive queries are hard. Carmeli and Segoufin [10] explore the fine-grained complexity of enumeration queries with self joins. Bringman et al. [8] 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 [11] 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 [18] 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[17]. 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. [2] 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 [15] 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. [3] 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 [16] 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 present open problems in Section 5.

There is a full version of this paper [13], which contains details of proofs, extra discussion, as well as the following omitted sections.

In the full version we: give the tight upper and lower bounds for minimum hypercycle, give the worst-case to average-case reduction for counting subhypergraphs, and show the average-case hardness for (self-join-free) counting queries.

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 8.** *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 9.** *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*

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

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

▶ **Definition 10.** *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.*

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 11.** *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 12** (Theorem 3.1 from [16]). *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 the full version [13].

▶ **Definition 13** (From [12]). *Let $H = (V_H, E_H)$ be a k-node graph with $V_H = \{x_1, \ldots, x_k\}$.*

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

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 theorem from [16] to show hardness for our hypercycle problems. For intuition about this reduction see the full version [13].

3.1 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 12 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 14.** *Recall the function $\gamma_u(k) = k - \lceil \frac{k}{u} \rceil + 1$ from Theorem 12. 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 4.** *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-to-hypercycle reduction from [16] 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.

3.1.1 Understanding the Hyperclique to Hypercycle Reduction

We begin by showing monotonicity of γ.

▶ **Lemma 15.** *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 $\gamma_c(k+1) \geq (k+1) - (\lceil \frac{k}{c} \rceil + 1) + 1 = \gamma_c(k)$. ◀

▶ **Corollary 16.** *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 17** (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 $V_i, V_{i+1 \mod k}, ..., V_{i+u-1 \mod k}$.*

While this seems like a highly restrictive definition, [16, 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 can show that such hypercycles cannot exist when $k \not\equiv 0 \mod u$.

▶ **Lemma 18.** *Suppose there exists a k-hypercycle $v_0, ..., v_{k-1}$ in a k-circle layered hypergraph with uniformity u that does not use exactly one node from each partition. Then, $k \equiv 0 \mod u$.*

For the proof of this lemma, see the full version [13].

3.1.2 Increasing Uniformity Preserves Hardness

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

▶ **Lemma 19.** *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 18, 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. ◀

3.1.3 Getting the Tight Lower Bound

We finally prove that short hypercycles require brute force:

▶ **Theorem 4.** *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 12 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 $k \leq \gamma_3^{-1}(u)$, Lemma 15 tells us that $\gamma_3(k) \leq u$. In the case these are equal, we already have the desired lower bound.

If the inequality is strict, we know from Lemma 19 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 $k \leq \gamma_3^{-1}(u) < 2u$ 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. ◀

From our worst-case to average-case reduction found in the full version of the paper [13], we can conclude that the average-case version of counting (u,k)-hypercycles also requires brute-force:

▶ **Corollary 20.** *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 4 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 Theorem 7 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 $k \geq 2u-1$. In this section we provide details for the second algorithm, details for the first can be found in the full version [13].

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 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 [16]. 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.

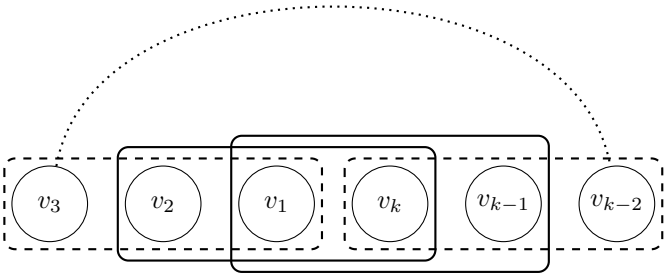

Figure 4 Example with uniformity $u = 3$ to demonstrate why you need $u - 1$ nodes. These are the four hyperedges which involve only the vertices $v_1, v_2, v_2, v_k, v_{k-1}$, and v_{k-2}. The two thick hyperedges include at least one vertex from v_1, v_2, v_3 and at least one vertex from v_k, v_{k-1}, 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 end of the hypercycle, we will need the last $u - 1$ edges to contain all of these vertices. See Figure 4 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.

4.1.1 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 21** ((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 $\text{CLR}_{k-1}(L, R)$ matrix and outputs another reach-ability matrix, $\text{CLR}_k(L, R)$, 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.

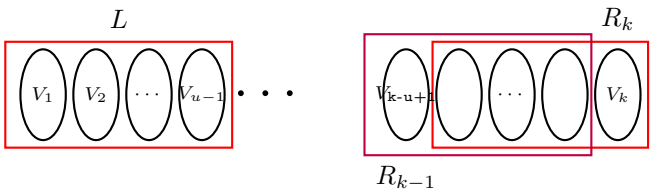

Figure 5 The (u, k)-ECLR problem.

▶ **Definition 22** ((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 $CLR_{k-1}(L, R_{k-1})$ 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 $CLR_k(L, R)$. This matrix is binary if unweighted and is over field of the weights if weighted. See Figure 5 for a visual of this problem.

4.1.2 Hypercycle Algorithm from Reachability Algorithm

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

▶ **Lemma 23.** *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 22.

Note that given the matrix u-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-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-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-CLR$_{i-1}(L,R_{i-1})$ to get u-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 24.** *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 $k\geq 2u-1$.*

4.1.3 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.

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

▶ **Lemma 25.** *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 26.** *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 24.** *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 23 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 26, we know we can set $T_1(n) = O(n^{2u-1-(3-\omega)})$. From Lemma 25 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 6.** *There exists a time-$\tilde{O}(k^k(n + m + n^{2u-1-(3-\omega)}))$ algorithm for finding k-hypercycles in any n-node u-uniform hypergraph G when $k \geq 2u - 1$.*

Proof. Using the color-coding approach from [16, 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 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.

5.1 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 $n^{\lambda_3^{-1}(u)-\epsilon}$ for some $\epsilon > 0$ 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?

5.2 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 [2]. 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 $\Omega(n^{u-\epsilon})$ hyperedges for some constant $\epsilon > 0$.
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" ([2]) 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?

5.3 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

1 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. doi:10.1007/978-3-662-43948-7_4.

2 Peter Allen, Julia Böttcher, Oliver Cooley, and Richard Mycroft. Tight cycles in hypergraphs. *Electronic Notes in Discrete Mathematics*, 49:675–682, 2015. doi:10.1016/J.ENDM.2015.06.091.

3 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. doi:10.1007/978-3-319-77404-6_3.

4 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. URL: http://proceedings.mlr.press/v70/backurs17a.html.

5 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. doi:10.1145/3055399.3055466.

6 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. doi:10.1007/978-3-319-96884-1_26.

7 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. doi:10.1109/FOCS.2019.00078.

8 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, pages 427–436, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3517804.3526234.

9 Nofar Carmeli and Markus Kröll. On the enumeration complexity of unions of conjunctive queries. *ACM Trans. Database Syst.*, 46(2), May 2021. doi:10.1145/3450263.

10 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, pages 277–289, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3584372.3588667.

11 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. doi:10.1145/3578517.

12 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.

13 Cheng-Hao Fu, Andrea Lincoln, and Rene Reyes. Worst-case and average-case hardness of hypercycle and database problems, 2025. arXiv:2504.18640.

14 Oded Goldreich. On counting t-cliques mod 2. *Electron. Colloquium Comput. Complex.*, TR20-104, 2020. URL: https://eccc.weizmann.ac.il/report/2020/104.

15 Hao Huang and Jie Ma. On tight cycles in hypergraphs. *SIAM Journal on Discrete Mathematics*, 33(1):230–237, 2019. doi:10.1137/18M117741X.

16 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. doi:10.1137/1.9781611975031.80.

17 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. URL: https://simons.berkeley.edu/workshops/fine-grained-complexity-logic-query-evaluation/schedule.

18 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. doi:10.1145/3589295.

19 Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. *CoRR*, abs/2307.07970, 2023. doi:10.48550/arXiv.2307.07970.