Online and Offline Learning of Orderly Hypergraphs Using Queries

Shaun M. Fallat SHAUN.FALLAT@UREGINA.CA
Department of Mathematics and Statistics, University of Regina, Canada

Kamyar Khodamoradi KAMYAR.KHODAMORADI@UREGINA.CA
Department of Computer Science, University of Regina, Canada

David G. Kirkpatrick KIRK@CS.UBC.CA
Department of Computer Science, University of British Columbia, Canada

Valerii Maliuk VALERA.MALUK@GMAIL.COM
Department of Computer Science, University of Regina, Canada

Seyed Ahmad Mojallal AHMAD_MOJALAL@YAHOO.COM
Department of Mathematics, Simon Fraser University, Canada

Sandra Zilles SANDRA.ZILLES@UREGINA.CA
Department of Computer Science, University of Regina, Canada; Amii, Canada

Editors: Matus Telgarsky and Jonathan Ullman

Abstract

In the context of learning hypergraphs with shortest-path queries (SP-queries), we present the first provably optimal online algorithm for learning a broad and natural class of hypertrees which we call *orderly hypertrees*. Our online algorithm can be transformed into a provably optimal offline algorithm. Orderly hypertrees can be positioned within the Fagin hierarchy of hypergraph acyclicity (studied in database theory), and strictly encompass the broadest class in this hierarchy that is learnable with subquadratic SP-query complexity. Our results also motivate the study of a new type of query, called dependency query (D-query), which is weaker than an SP-query. Positive and negative results on D-queries shed light on the structural properties of classes of hypertrees for which efficient learning requires the full information provided by SP-queries.

Keywords: Learning hypergraphs, shortest-path queries, online algorithms, hypergraph acyclicity

1. Introduction

Due to their various applications, hypergraphs have been the focus of numerous learning-theoretic studies in the past two decades. Typical learning settings are rooted in similar studies for conventional graphs (hypergraphs with edges of size 2), and often involve models in which a learner interacts with an information source (oracle) by asking queries about the unknown target hypergraph. The settings differ in the type of query the learner can ask, in whether the learner is deterministic or randomized, and whether it learns adaptively or non-adaptively Abrahamsen et al. (2016); Hein (1989); Janardhanan (2017); Kannan et al. (2015); King et al. (2003); Reyzin and Srivastava (2007).

A line of research on conventional graphs focused on learning trees with so-called shortest-path queries (SP-queries), allowing the learner to select two vertices whose distance in the target graph will be revealed. Hein (1989) showed that phylogenetic trees of bounded degree can be learned with $O(n \log n)$ SP-queries, where n denotes the number of vertices in the target tree. Their method was proven optimal through a matching lower bound (King et al., 2003). Recently, Bastide and Groenland (2025) presented an entirely new algorithm, proving that Hein's bound generalizes to the

class of all trees (not just phylogenetic trees of bounded degree); specifically, they obtained a bound of $O(n\Delta \log_\Delta n)$ for the number of SP-queries needed to learn any tree on n vertices, where Δ is the degree of the target tree and not known to the learner in advance. They provide a matching lower bound even for the expected number of SP-queries needed by a randomized algorithm.

The main question pursued in our paper is whether interesting classes of *hyper*trees can be learned with $o(n^2)$ SP-queries (in the worst case over all hypertrees in the class). This question is not only natural to ask but also of relevance to studies in database theory, computational biology, and other application domains in which hypergraphs (and specifically hypertrees) are used for modeling relationships between entities. To the best of our knowledge, the only existing work on learning hypertrees with SP-queries is a recent paper by Fallat et al. (2024); they provide two algorithms, each of which yields subquadratic query complexity only for a very restricted subclass of hypertrees. While Fallat et al. proved that learning hyperstars (hypertrees of diameter two) already requires $\Omega(n^2)$ SP-queries in the worst case, their algorithms suggest that there is hope for efficient learning of a large subclass of hypertrees of diameter at least three.

The definition of hypertree itself is not trivial. In database theory, an entire hierarchy of notions of hypergraph acyclicity has been proposed (Fagin, 1983), dubbed α-, β-, and γ-acyclicity, as well as Berge-acyclicity (in increasing order of restriction). We demonstrate that the class of γ-acyclic hypertrees, even when restricted to those of diameter at least three, has an SP-query complexity of $\Omega(n^2)$. By contrast, we define a new class of hypertrees, called orderly hypertrees, and show that this class (i) fits into Fagin's hierarchy between γ-acyclic and Berge-acyclic hypergraphs, and (ii) allows for efficient SP-query learning, when restricting to hypergraphs of diameter at least three.

Specifically, we show that Hein's method for learning phylogenetic trees can be generalized to an algorithm learning any orderly hypertree H of diameter at least three with $O(n\Delta \log_\Delta m)$ SP-queries, where Δ is the maximum number of edges any single edge in H can intersect, and m is the number of edges in H (both unknown to the learner). To do so, we represent an orderly hypergraph in a unique way as a special bipartite colored graph, which we call *skeleton graph*. Notably, orderly hypertrees are exactly those hypergraphs whose skeletons are trees. Our algorithm learns the skeleton tree of the target, and thus the target hypertree itself. The lower bound technique applied by Bastide and Groenland (2025) for conventional trees carries over to show that our algorithm is optimal. Moreover, it substantially improves on the two methods presented by Fallat et al. (2024). On the technical side, we make use of a new tree separator argument, similar in style to arguments previously used in the context of learning conventional trees (King et al., 2003; Bastide and Groenland, 2025), yet requiring novel insights in order to handle hypertrees.

Notably, our algorithm can be formulated as an *online* algorithm that inserts vertices one by one into an initially empty hypergraph, incrementally constructing the unique subhypergraph consistent with the queries posed so far. This makes it amenable to applications in which entities (vertices) are revealed in a stream. Moreover, our online algorithm is provably optimal in terms of worst-case query complexity among all possible online algorithms for the same task.

The distance information provided by SP-queries is crucial for efficient learning of orderly hypertrees: We introduce a weaker form of query, called *dependency query* (D-query), which asks whether there is an edge containing both the selected vertices. We show that the class of orderly hypertrees of diameter at least three cannot be learned with $o(n^2)$ D-queries, even if the learner can specify a vertex set of arbitrary size and ask whether it contains any two dependent vertices. To contrast this result, we provide non-trivial classes of (possibly cyclic) hypergraphs such that any target in this class can be learned with $O(mn)$ D-queries. In doing so, we show how mixing D-

queries with a small number of edge detection queries, which were previously studied in the context of hypergraphs (Angluin and Chen, 2006; Abasi et al., 2018; Balkanski et al., 2022), can be helpful.

Part of the content of this paper is presented in a modified and extended form in (Fallat et al., 2025).

2. Preliminaries

A hypergraph $H = (V, E)$ consists of a finite vertex set V of size n and an edge set E of size m; elements of E are subsets of V of cardinality ≥ 2. A *path* P in H from v_1 to v_t is an alternating sequence $v_1 e_1 \cdots e_{t-1} v_t$ in which $v_1, \ldots, v_{t-1} \in V$ are distinct vertices, $e_1, \ldots, e_{t-1} \in E$ are distinct edges, and for $1 \leq i < t$, $v_i, v_{i+1} \in e_i$. The *length* of P is $t-1$. If $v_1 = v_t$ then P is a *cycle*. Further, the *distance* $d_H(v, w)$ between two vertices v and w is the minimum length of a path from v to w, which is ∞ if no such path exists. Hypergraph H is *connected* if, for any two vertices u, v in H, there is a path from u to v. Given H, the eccentricity of a vertex $v \in V$ is given by $\max_{w \in V} d_H(v, w)$. The *edge degree* of an edge e in H is the number of edges $e' \neq e$ in H for which $e \cap e' \neq \emptyset$. A *private vertex* of an edge e in H is a vertex in e that does not belong to any edge $e' \neq e$ in H. We denote by $P_H(e)$ the set of all *private vertices* of edge e. The *line graph* of hypergraph H, denoted $L(H)$, is the graph with vertex set E and edge set $\{(e, e') \mid e, e' \in E \ \& \ e \cap e' \neq \emptyset\}$. If $U \subseteq V$, we denote by $H[U]$ the subhypergraph of H induced on U: $H[U] = \{e \cap U \mid e \in H\} \backslash \{\emptyset\}$.

Definition 1 *A connected hypergraph H is a hypertree if its line graph $L(H)$ is chordal and H satisfies the* Helly property*, i.e., given a subset $S \subseteq E$, if every two edges in S have a nonempty intersection, then S has a nonempty intersection.*

The focus of this paper is on learning classes of hypergraphs with queries. In this context, a learner for a class $\mathcal{H}$ of hypergraphs is an algorithm that works iteratively in rounds. In each round, it asks a query about an unknown target hypergraph $H = (V, E) \in \mathcal{H}$ and receives the correct answer from an oracle. The learning process stops successfully once the target H is the only hypergraph in $\mathcal{H}$ for which all queries have been answered correctly. The learner is said to *learn* $\mathcal{H}$ if it successfully identifies every target $H \in \mathcal{H}$ in this fashion, see, e.g., (Beerliova et al., 2006).

Unless stated otherwise, we assume that the learner knows the vertex set V, but has to identify the edge set E (as a set of subsets of V). It does not know any parameters of the target H (such as the number m of edges, the diameter d, etc.) unless pre-specified by $\mathcal{H}$. The efficiency of the learner is assessed in terms of the number of rounds of the learning process (i.e., the number of queries asked) in the worst case considered over all possible target hypergraphs in $\mathcal{H}$. Learning in our setup is *adaptive* in that each query may depend on the queries and responses processed in previous rounds. Our main result is an efficient algorithm for learning a natural class of hypertrees (to be defined below) from *shortest path queries* (SP-queries, for short). An SP-query consists of a pair $(v, w) \in V^2$ of vertices and is answered with $d_H(v, w)$, where H is the target hypergraph.

Definition 2 *A class $\mathcal{H}$ is SP-*learnable*, if there exists a learner that learns $\mathcal{H}$ using SP-queries. $\mathcal{H}$ is* hard to learn with SP-queries*, if every learner that learns $\mathcal{H}$ with SP-queries uses $\Omega(n^2)$ queries in the worst case over all targets $H \in \mathcal{H}$, where n is the number of vertices in H.*

Remark 3 *$\mathcal{H}$ is SP-learnable iff $\mathcal{H}$ contains no two distinct members H, H' such that $d_H = d_{H'}$.*

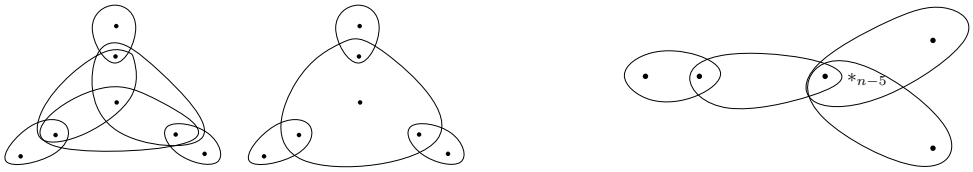

Figure 1: (Left) Two hypertrees with identical distances between corresponding vertices; vertices displayed in the same position in the left and right trees are assumed to be identical. (Right) Any class $\mathcal{H}$ containing all hypertrees isomorphic to this hypertree, with diameter 3, is hard to learn with SP-queries. (Here $*_{n-5}$ denotes a cluster of $n-5$ vertices).

Figure 1(left) displays two hypertrees that induce the same distance function; any class containing them both is hence not SP-learnable. SP-learnability is also impossible when $\mathcal{H}$ has two members H, H', where H has nested edges (i.e., at least two edges, one of which is strictly contained in the other) and H' results from H by removing at least one edge that is contained in another edge. We therefore assume throughout this paper that a hypergraph does not have nested edges.

Remark 4 *Fallat et al. (2024) showed that the class of hyperstars (hypertrees of diameter two) is hard to learn with SP-queries. However, even SP-learnable classes of hypertrees of diameter at least three can be hard to learn with SP-queries: Figure 1(right) shows such a hypertree of diameter three. Any class $\mathcal{H}$ containing all isomorphic hypertrees of this structure is hard to learn with SP-queries. An adversary can force a learner to query quadratically pairs among the $n-3$ rightmost depicted vertices; the argument here is identical to that in the proof of Lemma 27 in Appendix A.*

3. Orderly Hypergraphs

The leftmost and rightmost hypertrees in Figure 1 have in common that three of their edges have a non-empty intersection while two of the three have a strictly larger intersection. It turns out that this can make SP-learnability difficult. We hence propose the study of what we call *orderly* hypertrees.[1]

Definition 5 *A hypergraph H is intersection-orderly, or* orderly *for short, iff, for any two distinct edges $e_1, e_2 \in E$ with $S := e_1 \cap e_2 \neq \emptyset$, and any edge $e \in E$, we have either $S \subset e$ or $S \cap e = \emptyset$.*

Below we will show that orderly hypertrees of diameter at least three can be learned efficiently from SP-queries. This result is interesting from a learning-theoretic perspective, since both (i) the class of orderly hyperforests (collections of orderly hypertrees) and (ii) the class of orderly hypertrees of diameter two are hard to learn with SP-queries.[2] More importantly though, this result is significant from an application point of view since the notion of orderly hypertree fits nicely into the Fagin hierarchy (cf. Fagin (1983)) of acyclic hypergraphs, which is familiar and well-studied in relational database theory. According to Fagin, a hypergraph H is

- *α-acyclic* if and only if the empty hypergraph can be obtained from H by repeatedly removing isolated vertices, private vertices, and edges that are subsets of other edges.

1. Orderly hypertrees were previously also considered by Fallat et al. (2024), without explicitly naming them.
2. For (i), note that distinguishing between (a) a set of isolated vertices and (b) a set of isolated vertices plus an edge of size two requires $\Omega(n^2)$ SP-queries. Claim (ii) was proven by Fallat et al. (2024).

- β-*acyclic* if and only if all subsets of H are α-acyclic.
- γ-*acyclic* if and only if H is beta-acyclic and does not contain pairwise distinct vertices x, y, z such that $\{\{x, y\}, \{y, z\}, \{x, y, z\}\} \subseteq H[\{x, y, z\}]$.
- *Berge-acyclic* if the associated (bipartite) incidence graph $G = \{\{x, e\} \mid x \in e,\ e \in H\}$ is acyclic. Equivalently, H is β-acyclic and does not contain pairwise distinct vertices x, y, z such that $\{\{x, y\}, \{x, y, z\}\} \subseteq H[\{x, y, z\}]$.

It turns out that the class of orderly hypertrees lies between that of γ-acyclic and that of Berge-acyclic hypergraphs (see Appendix A for a proof):

Claim 6 *A hypergraph H is an orderly hypertree if and only if H is β-acyclic and does not contain pairwise distinct vertices x, y, z such that $\{\{x, y\}, \{y\}, \{x, y, z\}\} \subseteq H[\{x, y, z\}]$.*

Note that hypertrees obeying the structure shown in Figure 1 are γ-acyclic. Thus, by Remark 4, the class of γ-acyclic hypertrees is hard to learn with SP-queries. Orderly hypertrees (of diameter at least three) therefore strictly contain the structurally richest class in the acyclicity hierarchy that can be learned with $o(n^2)$ SP-queries in the worst case, as we will demonstrate below.

4. Skeletons of Hypergraphs

Orderliness of a hypergraph $H = (V, E)$ implies that V can be partitioned into disjoint subsets each of which is either the set of private vertices of an edge or the intersection of two edges.

Proposition 7 *For an orderly hypergraph $H = (V, E)$ the sets in $\bigcap_E = \{e \cap e' \mid e, e' \in E,\ e \neq e',\ e \cap e' \neq \emptyset\}$ together with the sets $P_H(e)$, for $e \in E$, form a partition of V. Any two vertices v and v' in the same part are* equivalent *in the sense that if $ve_1 \ldots e_{t-1}x$ is a path in H then $v'e_1 \ldots e_{t-1}x$ is a path in H (so the distances from v and v' to any third vertex x are identical).*

The proof is straightforward and given in Appendix A.

This property will be exploited by our learning algorithm for orderly hypertrees in Section 5. To this end, we introduce a helpful representation of a hypergraph as a (conventional) bipartite graph.

Definition 8 *Let $H(V, E)$ be any hypergraph. The skeleton graph of H, denoted $S(H)$, is defined as the bipartite graph with the following properties.*

- *Nodes in one part are black and in the other part are colored (either blue or red);*
- *Black nodes correspond to the elements of E, the edges of H. Blue nodes correspond to the non-empty sets of private vertices $P_H(e)$, $e \in E$. Red nodes correspond to the elements of $\bigcap_E$, the non-empty intersections of distinct edges in E; and*
- *Edges of $S(H)$ join (a) blue nodes to their corresponding (black) edge, and (b) red nodes to all (black) edges whose common intersection is the set associated with that red node.*

Figure 2 illustrates an orderly hypertree and its corresponding skeleton graph. Hypergraphs can be uniquely reconstructed from their skeleton graphs and vice versa, so that the problem of learning one is equivalent to the problem of learning the other (see Appendix A):

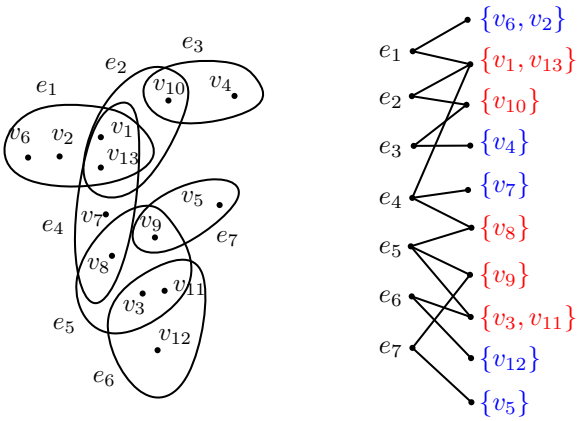

Figure 2: Orderly hypertree (left) and its skeleton graph (right)

Lemma 9 *Let $H=(V,E)$, $H'=(V,E')$ be orderly hypergraphs. If $S(H) = S(H')$ then $H = H'$.*

Note that the color of nodes is implicit in the skeleton structure: blue nodes have degree one, and red nodes have degree at least two. For a given orderly hypergraph H, denote by $[v]$ the colored node of $S(H)$ that contains vertex v.

If H is a hypergraph, a path P joining two nodes in its skeleton graph $S(H)$ has length $\lambda(P)$ given by the number of black nodes in P. Accordingly, if H is an orderly hypertree and $v_1e_1v_2\cdots e_{t-1}v_t$ is a path from vertex v_1 to vertex v_t in H, then $P = [v_1]e_1[v_2]\cdots e_{t-1}[v_t]$ and $P' = [v_1]e_1[v_2]\cdots e_{t-1}$ are paths in $S(H)$, where $\lambda(P) = \lambda(P') = t-1$.

Remark 10 *If H is an orderly hypertree there is a unique path in $S(H)$ joining a specified pair of nodes in $S(H)$. If P joins $[u]$ and $[v]$ in $S(H)$ then $\lambda(P) = d_H(u,v)$.*

Hence, the skeleton of an orderly hypertree is a tree. In fact, by Proposition 7 and Remark 10:

Claim 11 *Let $H = (V,E)$ be a hypergraph. Then the following three statements are equivalent. (i) H is an orderly hypertree. (ii) H is a hypertree and the colored nodes of $S(H)$ form a partition of V. (iii) $S(H)$ is a tree.*

Remark 12 *Let H be an orderly hypertree. Since the distances between all pairs of nodes in $S(H)$ uniquely determine the tree $S(H)$, it follows that—unlike the situation for general hypertrees—the distances between all pairs of vertices in an orderly hypertree H uniquely determine H.*

Thus, given the skeleton graphs of two orderly hypertrees, their isomorphism can be tested in time proportional to the sum of their edge degrees (the size of the skeleton graphs). In particular, orderly hypertree isomorphism can be tested in time $O(n\Delta \log_\Delta m)$, assuming reconstruction can be efficiently implemented. Similarly, other properties of a given orderly hypertree, such as its diameter (the length of the longest path) or centroid, can be computed from this representation in time proportional to the sum of their edge degrees.

5. Learning of Orderly Hypertrees With SP-Queries

The literature on learning (conventional) graphs provides at least two approaches demonstrating that trees can be learned with $O(n \log_\Delta n)$ SP-queries. The first approach is due to Hein (1989), who proved the weaker claim that phylogenetic trees with bounded degree can be learned with $O(n \log_2 n)$ SP-queries. Bastide and Groenland (2025) obtained the bound $O(n \log_\Delta n)$ with a different method, and for general trees. We will establish below that orderly hypertrees of diameter at least three can be learned with $O(n \log_\Delta m)$ SP-queries. To obtain this result, we would have had the option to attempt a generalization of Bastide and Groenland's approach to hypertrees or the alternative option of generalizing Hein's approach. While Bastide and Groenland's method appears simpler, Hein's method has the advantage that it is designed to insert vertices incrementally into an initially empty tree. We will show that this method can be generalized to an *online* learning algorithm for orderly hypertrees. In doing so, we show indirectly that Hein's method for conventional trees in fact gives an $O(n \log_\Delta n)$ bound for general trees (not just phylogenetic trees).

Online Learning of Hypergraphs. Let V^* be some universal set of vertices, and $\mathcal{H}$ a class of connected hypergraphs $H = (V, E)$, where $V \subseteq V^*$. In this setting, for any target hypergraph $H = (V, E) \in \mathcal{H}$, the set V (or even its size n) is *not* known to the learner. Instead, it is presented as a sequence. With each successive vertex v_{next}, the learner poses a set of distance queries between v_{next} and previously presented vertices, to an oracle. The learner is said to identify H in an online fashion if, for every $i \in [1 : n]$, every hypergraph $H' \in \mathcal{H}$ that is consistent with the distances obtained from the queries associated with the first i vertices satisfies $H'[v_1, \dots v_i] = H[v_1, \dots v_i]$. In particular, after all n vertices have been presented, the distance profile is unique to H, among hypergraphs on V within $\mathcal{H}$. Note that the cost of (i.e., the number of queries asked by) a learning algorithm A when identifying a target hypergraph H in a class $\mathcal{H}$ now depends on the sequence in which the vertices in V are presented to A. In general, the cost of online algorithms can be substantially higher than the cost of offline algorithms. For a fair competitive analysis, one needs to consider the cost with respect to other algorithms that deal with the same presentation.

Fallat et al. (2024) showed that the class of hypergraphs consisting of only two intersecting edges is hard to learn offline with SP-queries. Given any individual vertex $v \in e_1 \cap e_2$ in the target hypergraph $H = (V, \{e_1, e_2\})$, the learner needs to know a pair $(v_1, v_2) \in P_H(e_1) \times P_H(e_2)$ in order to determine that v is not private. As long as no such pair is known, all distances observed will be 1, which leaves open the possibility of all the vertices used in queries so far belonging to a single edge. Thus, *every* online learner for the class of orderly hypertrees (even if constrained to have diameter at least three) must incur a worst-case cost quadratic in the number h of vertices presented before two vertices from two distinct edges have occurred in the presentation.

The algorithm we present below consumes $O(h^2 + i\Delta \log_\Delta m)$ queries on any vertex sequence of length i, which turns out to be asymptotically optimal.

5.1. Induced Sub-Skeletons of Orderly Hypertrees

Our main result is an algorithm that learns orderly hypertrees from SP-queries in an online fashion. For any sequence $(v_1, \dots, v_i)$ processed by the algorithm, it produces a skeleton graph that is consistent with the *sub-skeleton induced by the target hypertree* on the vertex set $\{v_1, \dots, v_i\}$—a notion we first need to define formally. Let $H^* = (V^*, E^*)$ be an orderly hypertree, and let $d^*(u, v)$

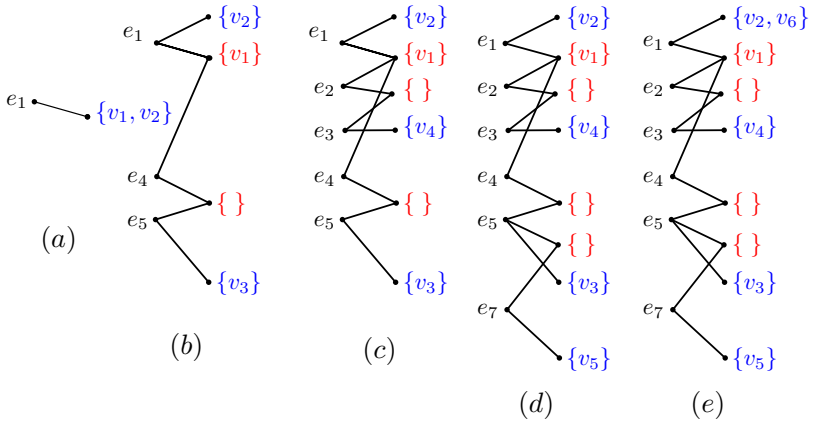

Figure 3: Sub-skeletons of the orderly hypertree from Figure 2 induced on vertex sets $\{v_1, v_2\}$ (a), $\{v_1, v_2, v_3\}$ (b), $\{v_1, v_2, v_3, v_4\}$ (c), $\{v_1, \ldots, v_5\}$ (d) and $\{v_1, \ldots, v_6\}$ (e).

denote the length of a shortest path joining u and v in H^*. For any $V \subseteq V^*$, denote by $d^*(V, V)$ the set $\{d^*(u, v) \mid u, v \in V\}$. The elements of $d^*(V, V)$ define a unique substructure of $S(H^*)$:

Definition 13 *The* sub-skeleton *of H^* induced on V is formed from $S(H^*)$ by (i) replacing all colored nodes by their intersection with V, and then (ii) choosing the smallest subtree that contains all of the resulting non-empty colored nodes.*

The length of the path joining two colored nodes in a sub-skeleton, whether they are occupied or not, uniquely determines the hypertree distance between vertices that ultimately occupy those nodes. This allows our learning algorithm to infer full distance information from limited distance queries. Our online algorithm now proceeds in two phases. In Phase 1, it determines whether all vertices presented so far belong to a single edge. As soon as this is no longer true, Phase 2 starts.

5.2. Phase 1: while there is a one-edge sub-skeleton consistent with all SP-queries so far

Suppose that vertices are indexed by their position in the insertion sequence. Addition of a new vertex v_{next}, $\text{next} > 1$ involves querying its distance from all vertices v_j, $j < \text{next}$. While $d^*(v_j, v_{\text{next}}) = 1$, for $j < \text{next}$, v_{next} is added to the single blue node in the skeleton containing all of the vertices v_j, $j < \text{next}$ (private vertices of a single edge). The phase ends when it is discovered that $d^*(v_j, v_{\text{next}}) > 1$, for some v_j, $j < \text{next}$. The single blue node is identified as the attachment node N_A and the skeleton is updated as described in Phase 2 below.

For the hypertree illustrated in Figure 2, Phase 1 ends with the insertion of vertex v_3. Figure 3 (b) illustrates the sub-skeleton at the transition from Phase 1 to Phase 2.

5.3. Phase 2: the current sub-skeleton has diameter at least two

The change in the current sub-skeleton resulting from the insertion of a new vertex v_{next} is either (i) the expansion of a colored sub-skeleton node, or (ii) the appendage of chain of new sub-skeleton nodes at some attachment point in the current sub-skeleton. Figure 3 (c) (d), and (e) illustrate the sub-skeleton after the insertion of the first four, five and six vertices. Addition of a vertex v_{next} in

this phase updates the current sub-skeleton by (i) determining the point of update/attachment (using maximal path separators), and (ii) updating the sub-skeleton at this attachment node, denoted N_A.

Determining the Point of Update/Attachment The location of the node N_A is determined by identifying a sequence of paths in the current sub-skeleton. The paths in question all have the skeleton node $[v_1]$ as one endpoint and a blue node (that we denote by $[v_\perp]$) as the other endpoint.

For each such path P, we need to determine the node on P, denoted $N_\vdash$, where either (i) vertex v_{next} should be inserted, or (ii) the path from node $[v_1]$ to node v_{next} departs from P. For this we use SP-queries to determine the distance from $[v_\perp]$ to both v_1 and v_{next}. These distances (together with the distance from v_1 to v_{next}) determine both the location of node $N_\vdash$ (and ultimately node N_A). Note that earlier tree-reconstruction methods use the same observation (cf. (Hein, 1989)).

Lemma 14 *The pairwise distances (in H^*) between any three vertices, $u, v,$ and w uniquely determine the sub-skeleton induced on these vertices. (See Appendix A for details.)*

Consider the current sub-skeleton, expanded to include v_{next} and rooted at $[v_1]$. Removal of the edges along the path P from $[v_0]$ to $[v_\perp]$ yields a collection of disjoint rooted subtrees. By Lemma 14, we can determine node $N_\vdash$, the root of the subtree in this collection that contains $[v_{\text{next}}]$.

Let $k = d^*(v_1, v_{\text{next}}) + d^*(v_\perp, v_{\text{next}}) - d^*(v_1, v_\perp)$. There are two cases:

(k is even) In this case, v_{next} belongs to the subtree rooted at the colored node $N_\vdash$ at distance $d^*(v_1, v_{\text{next}})$ from $[v_1]$ on the path between $[v_1]$ to $[v_\perp]$.

(k is odd) In this case, v_{next} belongs to the subtree rooted at the black node $N_\vdash$ at distance $d^*(v_1, v_{\text{next}})$ from $[v_1]$ on the path between $[v_1]$ to $[v_z]$.

The process continues, by choosing $[v_\perp]$ to be a leaf node in the subtree rooted at $N_\vdash$, until this subtree is reduced to a single node, which is the desired update/attachment node N_A.

The choice of node $[v_\perp]$ is critical in realizing the desired bound on the number of SP-queries. By generalizing an observation made by Hein (1989), one can show that a greedy choice results in a bound of $\Omega(n\sqrt{m})$ SP-queries; see Appendix A for details. The choice (and its associated path) that leads to the efficient (measured in terms of worst-case number of resulting SP-queries) location of the update/attachment point N_A is abstracted as a two player game on $S(H)$, detailed below.

Skeleton Update Lemma 14 dictates how to update a sub-skeleton at N_A. There are two cases:
($k = 0$) In this case the new vertex v_{next} expands the colored attachment node N_A.
($k > 0$) In this case a new node $[v_{\text{next}}]$ is created, linked by a length k alternating chain of black and colored nodes to the attachment node N_A.

If the attachment node N_A is blue (with adjacent black node N_B) then k must be even. In this case, N_A becomes red and the vertices currently associated with that node that have distance $k/2+1$ from v_{next} are split off into a new blue node adjacent to N_B. This splitting can occur at most Δ times for any vertex, since each split adds one to the degree of the edge associated with N_B. Thus the total number of SP-queries needed to perform the splits in the first i insertions is $O(i\Delta)$.

In our example, the insertions of both v_2 and v_4 lead to even length attachments at $[v_1]$ (Figure 3(center left)). Following this, inserting v_5 leads to an odd length attachment at the black node e_5 (Figure 3(center right)). Next, inserting v_6 leads to an expansion of node $[v_2]$ (Figure 3(right)).

5.4. Algorithm Analysis: Correctness and Complexity

The correctness of the update process follows from Lemma 14. The process of locating the update/attachment point in the current skeleton terminates since the size of the subtree rooted at $N_\vdash$

decreases with each successive path P. To obtain an SP-query complexity of $O(\Delta \log_\Delta m)$ for each insertion step, the method for selecting node $[v_\perp]$ (and the associated path) is crucial.

Definition 15 *Let $T = (V, E)$ be a rooted tree. The* path depth game *on T is played in rounds by two players, $\pi_{\min}$ and $\pi_{\max}$. In each round, (i) π_{min} first selects a path P in T and deletes all edges on P from E (leaving V unchanged), and then (ii) π_{max} selects one of the thus exposed rooted subtrees T' of T and sets $T := T'$. The game ends when T consists of a single vertex.*

We define the path depth *of T to be the number of rounds after which this game ends, assuming that $\pi_{\min}$ aims to minimize the number of rounds and $\pi_{\max}$ aims to maximize it.*

Imagine that the path game is played on the current sub-skeleton rooted at node $[v_1]$. The choice of a path P by $\pi_{\min}$ translates directly into a choice of node $[v_\perp]$: simply extend P to any leaf node. The choice of exposed subtree by $\pi_{\min}$ reflects the choice of an adversary forcing the worst-case behaviour of our algorithm. It follows that the number of SP-queries required, in the worst case, to locate the update/attachment node N_A in any insertion step is at worst three times the path depth of the current sub-skeleton. We show that this depth value is in $\Theta(\Delta \log_\Delta m)$, where Δ denotes the maximum vertex degree in the target skeleton $S(H^*)$ (and thus the maximum edge degree in H^*).

Theorem 16 *Consider the class of all rooted trees $T = (V, E)$ with ℓ leaves and maximum vertex degree at most Δ. The worst-case path depth of such trees is in $\Theta(\Delta \log_\Delta \ell)$.*

Proof Let $k = \lceil \log_\Delta \ell \rceil$. To prove the lower bound, suppose Δ is odd. We show that $\Omega(k\Delta)$ rounds are required for the path depth game with a complete rooted tree T_k of uniform out-degree Δ and depth k (which has Δ^i nodes on level i). Let $R_{i,j}$ denote a rooted tree, whose root has $2i + 1$ children, each of these the root of a subtree T_j. Note that $R_{(\Delta-1)/2,k-1} = T_k$. Faced with $R_{i,j}$, where $i > 0$ and $j \geq 0$, $P_{\min}$'s choice of a path contains nodes in a most two of the $2i + 1$ principal subtrees. This will leave an untouched subtree $R_{i-1,j}$, in the case $i > 1$, or $R_{(\Delta-1)/2,j-1}$, if $i = 1$ and $j > 0$. It follows by induction on i and j that $\pi_{\max}$ can force $i + j((\Delta - 1)/2)$ rounds. So, starting with $R_{(\Delta-1)/2,k-1}$, the player $\pi_{\max}$ can force at least $k(\Delta - 1)/2$ rounds.

For the upper bound, let T be any rooted tree with ℓ leaves and maximum vertex degree $\leq \Delta$. We will call any vertex in T i-light, if it roots a subtree that has at most Δ^{i-1} leaves. All other vertices are called i-heavy. A i-light (i-heavy) subtree is a subtree rooted in a i-light (i-heavy) vertex. The term *i-fringe-heavy vertex* refers to any i-heavy vertex all of whose children are i-light.

The game is played by $\pi_{\min}$ in k stages. Stage i, $i = k, k - 1, \ldots 1$ starts with a $i + 1$-light tree and, following $O(\Delta)$ rounds, forces $\pi_{\max}$ to select a i-light tree. Hence, after k stages, consisting of $O(k\Delta)$ rounds in total, the game ends with a subtree consisting of a single vertex.

Consider the i-th stage. For any path $(r, u_1, \ldots, u_s, v)$ from r to a i-fringe-heavy vertex v, all vertices $u_1, \ldots, u_s$ are i-heavy, but not i-fringe-heavy. Thus, T has $\leq \Delta$ i-fringe-heavy vertices. (Otherwise T would have $> \Delta \cdot \Delta^{i-1}$ leaves, contradicting our assumption that the stage begins with an $i + 1$-light tree.) Also, every i-heavy vertex lies on a path from r to one of the at most Δ i-fringe-heavy vertices. $\pi_{\min}$ starts by picking any path P from r to any i-fringe-heavy vertex v. $\pi_{\max}$ can now choose between any of the subtrees rooted at vertices on P. Since any such choice results in a subtree with one fewer i-fringe-heavy vertex in its interior, it follows that after at most Δ rounds $\pi_{\max}$ is forced to choose a subtree T that contains no i-heavy vertices in its interior.

At this point, either T is i-light, which ends the stage, or all of the at most Δ principle subtrees of T are i-light. In the latter case, $\pi_{\min}$ can select a path consisting of only a single edge incident to

v, splitting off a light subtree with each such selection. After no more than Δ such selections, $\pi_{\max}$ is forced to return a i-light tree, ending the stage. Thus the i-th stage ends after $\leq 2\Delta$ rounds. ■

To sum up, we obtain Theorem 17 for an algorithm without prior knowledge of V, n, E, m, Δ.

Theorem 17 *The class of all n-vertex orderly hypertrees of diameter at least three can be learned, in an online fashion, using $O(h^2 + i\Delta \log_\Delta m)$ SP-queries in total for the first i vertex insertions, where m (resp., Δ) is the (unknown) number of edges (resp., maximum edge degree) of the target hypertree, and h is the length of the prefix of the first i insertions that consists of vertices that all belong to a common edge.*

5.5. Offline Learning of Orderly Hypertrees

Given that the target hypertree has diameter at least three, an offline learner with access to V can ask a linear number of queries of the form (v_1, v_2), (v_1, v_3), (v_1, v_4), until a vertex at distance greater than 1 from v_1 is found (some such vertex must exist). The learner can then proceed exactly as the online learner does in Phase 2, and identify the target graph without the h^2 overhead. This yields the following result, which is realized by an algorithm that knows (only) V in advance.

Theorem 18 *The class of all n-vertex orderly hypertrees of diameter at least three can be learned, in an offline fashion, using $O(n\Delta \log_\Delta m)$ SP-queries, where m (resp., Δ) is the (unknown) number of edges (resp., maximum edge degree) of the target hypertree.*

Recall that dropping the condition on the diameter would give us a lower bound of $\Omega(n^2)$, since orderly hyperstars are hard to learn. Note, however, that connectivity is not essential for learning to have sub-quadratic cost. The additional cost in dealing with a hyperforest with $\leq c$ hypertrees is $O(cn)$ queries, since c queries will suffice to determine which of the hypertrees a new vertex belongs to; the remainder of the insertion procedure works as described above.

5.6. Asymptotic Optimality of Our Algorithm

For learning conventional trees with n vertices (and $n-1$ edges), Bastide and Groenland (2025) proved an $\Omega(n\Delta \log_\Delta n)$ lower bound on the expected number of SP-queries used by any randomized offline algorithm, where Δ is an upper bound in the degree of the target tree. A straightforward application of the same tools they use yields an $\Omega(n\Delta \log_\Delta m)$ lower bound on the number of SP-queries used by any deterministic offline algorithm for learning orderly hypertrees of diameter at least three. It follows that our offline algorithm for learning orderly hypertrees is asymptotically optimal. Moreover, since the $\Theta(h^2)$ overhead in the online setting is unavoidable, our online algorithm is also asymptotically optimal. In other words, the bounds in Theorems 17 and 18 are tight.

6. Learning Hypergraphs with Dependency Queries

Our algorithm uses distance information in order to insert vertices into sub-skeletons. This raises the question whether efficient learning is possible with a weaker form of queries—asking only whether the two given vertices are *dependent*, i.e., belong to one edge. Generalizing this notion, the learner selects a set $F \subseteq V$ of any size, and will be told whether F contains at least two dependent vertices.

Definition 19 *Let $H = (V, E)$ be a hypergraph. A dependency query (D-query), posed by a learner with unknown target H, consists of a set $F \subseteq V$ which the learner passes to the oracle. The oracle responds 1 if there is any edge in E including at least two vertices in F, and 0 otherwise. If $q \leq n$, then the term D_q-query refers to any D-query in which the set F is of cardinality at most q.*

Clearly, D_2-queries are no stronger than SP-queries. However, D_n-queries turn out to be incomparable to SP-queries, in terms of how much information they provide to a learner. Comparing these types of queries, we use the term *X-query complexity* of a class $\mathcal{H}$ of hypergraphs to refer to the worst-case number of X-queries a learner has to make in order to learn any member of $\mathcal{H}$.

Proposition 20 *There are classes of hypergraphs for which (i) the D_2-query and D_n-query complexity are asymptotically larger than the SP-query complexity, (ii) the SP-query and D_2-query complexity are asymptotically larger than the D_n-query complexity. In particular, claim (i) is witnessed by the class of orderly hypertrees of diameter at least three. (See Appendix A for a proof.)*

Thus, our algorithm for learning orderly hypertrees of diameter at least 3 with $O(\Delta n \log_\Delta m)$ SP-queries cannot be replaced by an efficient algorithm using D_2-queries, or even D_n-queries.

We now show that replacing SP-queries with D_2-queries gives us an upper bound $O(mn)$, given that the target hypergraph is known to have private vertices in every edge. This result even holds beyond hypertrees; its proof is given in Appendix A.

Theorem 21 *Let $\mathcal{H}$ be the class of all connected orderly hypergraphs of order n and diameter at least 3, with each edge containing at least one private vertex. The worst-case number of D_2-queries needed to learn any target hypergraph $H \in \mathcal{H}$ is in $O(mn)$, where m is the number of edges in H. Dropping any of the four premises in isolation would yield quadratic D_2-query complexity.*

The final statement in Theorem 21 shows that D-queries are rather ineffective unless we impose strong constraints on the hypergraphs to be learned. These constraints can be substantially reduced if the learner can initially use a small amount of queries of a different type:

Note that D_n-queries are one possible way of generalizing *edge detection* queries (ED-queries) from graphs to hypergraphs. For both conventional graphs and hypergraphs, an ED-query is determined by a subset $F \subseteq V$ of the vertex set; the oracle responds with 1 if the set F contains an edge, and with 0 otherwise (Alon and Asodi, 2005; Grebinski and Kucherov, 1997; Abasi and Bshouty, 2019; Angluin and Chen, 2008, 2006; Abasi et al., 2018; Balkanski et al., 2022). On conventional graphs, D_n-queries and ED-queries are equivalent.

Interestingly, asking n ED-queries before switching to D_2-queries, results in the same bound as in Theorem 21, yet for a much larger class of hypergraphs. The condition on connectedness can then be dropped, and private vertices are required only in 3-cycles contained as induced sub-hypergraphs:

Definition 22 *A vicious 3-cycle is a hypergraph consisting of three edges e_1, e_2 and e_3 such that (i) $e_i \cap e_j \neq \emptyset$, and $e_i \cap e_j \cap e_k = \emptyset$, where $i, j, k \in \{1, 2, 3\}$ are distinct, and (ii) at least one of the three edges has no private vertex.*

Theorem 23 *Let $\mathcal{H}$ be the class of all (not necessarily connected) orderly hypergraphs whose components have diameter ≥ 3, and contain no vicious 3-cycles as induced sub-hypergraphs. Then there is an algorithm that learns every $H \in \mathcal{H}$ using n ED-queries and $O(nm)$ D_2-queries.*

The proof of this theorem uses a nice hitting set argument, and is detailed in Appendix A.

7. Conclusions

Our focus was on query learning of a broad class of hypertrees which we call orderly hypertrees. Their positioning in Fagin's hierarchy makes them potentially relevant to database theory, and our results showed that they are, compared to members of Fagin's hierarchy, the broadest class of hypertrees for which efficient SP-query learning is possible. We provided a provably optimal *online* SP-query algorithm for learning a orderly hypertrees of diameter at least three, which can be transformed into an optimal offline algorithm for the same class. Our results also motivated the definition and study of D-queries, potentially opening up new lines of research on novel query types.

Our SP-query algorithm for learning orderly hypertrees is built on a method proposed by Hein (1989) for conventional trees. An alternative approach, followed by generalizing the method by Bastide and Groenland (2025), would not yield an online learning algorithm, but it may potentially have other advantages such as applying to a broader class of hypertrees, allowing general edge weights, or yielding improved bounds for specific subclasses of orderly hypertrees. It would also open up the possibility of adopting related query modes as described by King et al. (2003), such as bounded queries and ε-approximate queries. We leave this approach for future consideration.

Acknowledgments

S. Fallat was supported in part by an NSERC Discovery Grant, application no. RGPIN-2019-03934.
D. Kirkpatrick was supported through the NSERC Discovery Grants program, under grant no. 22R83583.
K. Khodamoradi was supported through the NSERC Discovery Grants program, application no. RGPIN-2024-06360.
S. Zilles was supported through a Canada CIFAR AI Chair at the Alberta Machine Intelligence Institute (Amii), through an NSERC Canada Research Chair, through the New Frontiers in Research Fund under grant no. NFRFE-2023-00109 and through the NSERC Discovery Grants program under application no. RGPIN-2017-05336.

References

Hasan Abasi and Nader H. Bshouty. On learning graphs with edge-detecting queries. In *Proceedings of the International Conference on Algorithmic Learning Theory (ALT)*, pages 3–30, 2019.

Hasan Abasi, Nader H. Bshouty, and Hanna Mazzawi. Non-adaptive learning of a hidden hypergraph. *Theoretical Computer Science*, 716:15–27, 2018.

Mikkel Abrahamsen, Greg Bodwin, Eva Rotenberg, and Morten Stöckel. Graph reconstruction with a betweenness oracle. In *Proceedings of the 33rd Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 5:1–5:14, 2016.

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

Dana Angluin and Jiang Chen. Learning a hidden hypergraph. *Journal of Machine Learning Research*, 7:2215–2236, 2006.

Dana Angluin and Jiang Chen. Learning a hidden graph using $O(\log n)$ queries per edge. *Journal of Computer and System Sciences*, 74(4):546–556, 2008.

Eric Balkanski, Oussama Hanguir, and Shatian Wang. Learning low degree hypergraphs. In *Proceedings of the 35th Conference on Learning Theory (COLT)*, pages 419–420, 2022.

Paul Bastide and Carla Groenland. Tight distance query reconstruction for trees and graphs without long induced cycles. *Random Structures and Algorithms*, 66(4), 2025.

Zuzana Beerliova, Felix Eberhard, Thomas Erlebach, Alexander Hall, Michael Hoffmann, Matús Mihalák, and L. Shankar Ram. Network discovery and verification. *IEEE Journal on Selected Areas in Communications*, 24(12):2168–2181, 2006.

Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM (JACM)*, 30(3):514–550, 1983. doi: 10.1145/2402.322390.

Shaun Fallat, Valerii Maliuk, Seyed Ahmad Mojallal, and Sandra Zilles. Learning hypertrees from shortest path queries. In *Proceedings of the 35th International Conference on Algorithmic Learning Theory (ALT)*, pages 1–16, 2024.

Shaun M. Fallat, Kamyar Khodamoradi, David G. Kirkpatrick, Valerii Maliuk, Seyed Ahmad Mojallal, and Sandra Zilles. Distance-based learning of hypertrees. arXiv:2511.22014, 2025.

Vladimir Grebinski and Gregory Kucherov. Optimal query bounds for reconstructing a hamiltonian cycle in complete graphs. In *Proceedings of the 5th Israel Symposium on Theory of Computing and Systems (ISTCS)*, pages 166–173, 1997.

Jotun J. Hein. An optimal algorithm to reconstruct trees from additive distance data. *Bulletin of Mathematical Biology*, 51(5):597–603, 1989.

Mano Vikash Janardhanan. Graph verification with a betweenness oracle. In *Proceedings of the International Conference on Algorithmic Learning Theory (ALT)*, pages 238–249, 2017.

Sampath Kannan, Claire Mathieu, and Hang Zhou. Near-linear query complexity for graph inference. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 773–784, 2015.

Valerie King, Li Zhang, and Yunhong Zhou. On the complexity of distance-based evolutionary tree reconstruction. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 444–453, 2003.

Lev Reyzin and Nikhil Srivastava. Learning and verifying graphs using queries with a focus on edge counting. In *Proceedings of the 18th International Conference on Algorithmic Learning Theory (ALT)*, pages 285–297. Springer, 2007.

Appendix A. Deferred proofs

Claim 6 *A hypergraph H is an orderly hypertree if and only if H is β-acyclic and does not contain pairwise distinct vertices x, y, z such that $\{\{x, y\}, \{y\}, \{x, y, z\}\} \subseteq H[\{x, y, z\}]$.*

Proof "$\Rightarrow$" Let H be an orderly hypertree.

To show that H is β-acyclic, we need to show that every subset $H' \subseteq H$ of hyperedges can be transformed to the empty hypergraph by repeatedly removing isolated vertices, private vertices, and edges contained in other edges. So let $H' \subseteq H$ and let H'' be the hypergraph resulting from H' after the iterated removal of isolated and private vertices as well as contained edges.

Suppose H'' is not empty. Then every edge e in H'' intersects at least two distinct other edges e_1, e_2 in H''; moreover, for every edge e in H'', there are vertices v_1, v_2 in H'' and edges e_1, e_2 in H'' such that $v_1 \in e \cap e_1 \backslash e_2$ and $v_2 \in e \cap e_2 \backslash e_1$. Orderliness then implies that $e \cap e_1$ and $e \cap e_2$ are disjoint. Since this holds for every edge e in H'', and the latter has finitely many edges, the line graph of H'' has (a) a cycle of length three or (b) a chordless cycle of length at least four. (a) contradicts the Helly property in combination with orderliness. (b) is not possible, since H'' is a hypertree. Therefore, H'' is empty, and H is β-acyclic.

It remains to show that H does not contain three distinct vertices x, y, z such that $\{\{x, y\}, \{y\}, \{x, y, z\}\} \subseteq H[\{x, y, z\}]$. If such x, y, z existed, then H would have pairwise distinct edges e_1, e_2, e_3 with $e_1 \cap \{x, y, z\} = \{x, y\}$, $e_2 \cap \{x, y, z\} = \{y\}$, and $e_3 \cap \{x, y, z\} = \{x, y, z\}$. Let $S = e_1 \cap e_3$. Since $\{x, y\} \subseteq S$, we have $S \neq \emptyset$. Now $e_2 \cap S \neq \emptyset$, but e_2 does not contain S. This contradicts the premise that H is orderly. Consequently, H does not contain three distinct vertices x, y, z such that $\{\{x, y\}, \{y\}, \{x, y, z\}\} \subseteq H[\{x, y, z\}]$.

"$\Leftarrow$" Let H be a β-acyclic hypergraph that does not contain three distinct vertices x, y, z with $\{\{x, y\}, \{y\}, \{x, y, z\}\} \subseteq H[\{x, y, z\}]$. We need to show that $L(H)$ is chordal, H has the Helly property, and H is orderly.

To show that $L(H)$ is chordal, suppose by way of contradiction that $L(H)$ has a chordless cycle of length at least four. The set of vertices of such cycle are a set of edges in H that form a subhypergraph of H that is not α-acyclic. Thus H is not β-acyclic—a contradiction. Therefore, $L(H)$ is chordal.

To verify that H is orderly, let e_1 and e_2 be two edges in H with non-empty intersection. By way of contradiction, suppose there is an edge $e \in E \backslash \{e_1, e_2\}$ such that e contains some element $y \in e_1 \cap e_2$, but $e \not\supseteq e_1 \cap e_2$. Thus, let $x \in e_1 \cap e_2 \backslash e$. Since we limit ourselves to hypergraphs without nested edges, there are vertices z, z' such that $z \in e_1 \backslash e_2$ and $z' \in e_2 \backslash e_1$. By the premise, $\{\{x, y\}, \{y\}, \{x, y, z\}\} \not\subseteq H[\{x, y, z\}]$ and $\{\{x, y\}, \{y\}, \{x, y, z'\}\} \not\subseteq H[\{x, y, z'\}]$. Therefore, e contains both z and z'. Thus $\{e_1, e_2, e\}[\{x, y, z, z'\}] = \{\{x, y, z\}, \{x, y, z'\}, \{y, z, z'\}\}$. In particular, $\{e_1, e_2, e\}$ is not α-acyclic, and thus H is not β-acyclic—a contradiction. Hence H is orderly.

The Helly property can be verified as follows. Suppose $S := \{e_1, \ldots, e_k\}$ is a minimal subset of E contradicting the Helly property, i.e., any two edges in S intersect, $e_1 \cap \cdots \cap e_{k-1} \neq \emptyset$, but $e_1 \cap \cdots \cap e_k = \emptyset$. Since H is orderly, $e_1 \cap \cdots \cap e_{k-1} = e_i \cap e_j$ for any two distinct $i, i \in \{1, \ldots, k-1\}$. Thus, $e_1, \ldots, e_{k-1}$ form an orderly hyperstar, and e_k intersects each edge of that hyperstar but not the intersection of the hyoperstar edges. After removing all isolated and private vertices from $e_1, \ldots, e_k$, none of the remaining edges are nested, so that the empty hypergraph cannot be obtained by removal of contained edges. In other words, S is a subhypergraph of H that is not α-acyclic.

Thus H is not β-acyclic, which contradicts the premises. Consequently, H has the Helly property. ■

Proposition 7 *For an orderly hypergraph $H = (V, E)$ the sets in $\bigcap_E = \{e \cap e' \mid e, e' \in E,\ e \neq e' \ \&\ e \cap e' \neq \emptyset\}$ together with the sets $P_H(e)$, for $e \in E$, form a partition of the vertices in V. Furthermore, any two vertices v and v' in the same part are* equivalent *in the sense that if $ve_1 \ldots e_{t-1}x$ is a path from v to x in H then $v'e_1 \ldots e_{t-1}x$ is a path from v' to x in H (so, in particular, the distances from v and v' to any third vertex x are identical).*

Proof Clearly, $\bigcap_E \cup \bigcup_{e \in E} P_H(e) = V$. Moreover, given $e \in E$, no vertex $v \in P_H(e)$ is contained in $P_H(e')$ for any $e' \in E \backslash \{e\}$ or in $e' \cap e''$ for any $e', e'' \in E$, $e' \neq e''$. Now suppose a vertex $v \in V$ belongs to two distinct sets $e_1 \cap e_1' \in \bigcap_E$, $e_2 \cap e_2' \in \bigcap_E$, where $e_i, e_i' \in E$. Then the intersection $e_1 \cap e_1' \cap e_2 \cap e_2'$ contains v, but $e_1 \cap e_1' \neq e_2 \cap e_2'$, which immediately violates the definition of orderliness. ■

Lemma 9 *Let $H = (V, E)$, $H' = (V, E')$ be orderly hypergraphs. If $S(H) = S(H')$ then $H = H'$.*

Proof The black nodes of $S(H)$ correspond to the edges of H. The vertices forming the edge corresponding to a given black node are exactly the vertices associated with the colored nodes adjacent to that black node. ■

Lemma 14 *The pairwise distances (in H^*) between any three vertices, $u, v,$ and w uniquely determine the sub-skeleton induced on these vertices.*

Proof By Remark 10, there is a unique path from $[u]$ to $[v]$ (resp., $[u]$ to $[w]$ and $[v]$ to $[w]$) in $S(H^*)$. Since the length of this path is $d_{H^*}(u, v)$ (resp., $d_{H^*}(u, w)$ and $d_{H^*}(v, w)$)), the sub-skeleton induced on $\{u, v, w\}$ is uniquely determined. ■

We remarked in the main body that the choice of node $[v_\perp]$ is critical in realizing the desired bound on the number of SP-queries. A greedy choice can be inefficient:

Proposition 24 *The online learning algorithm desribed in Section 5 chooses a sequence of paths all of which have the skeleton node $[v_1]$ as one endpoint and a blue node $[v_\perp]$ as the other endpoint. Suppose $[v_\perp]$ is always chosen to be the most distant vertex from $[v_1]$ in the subtree containing the vertex v_{next}. Then, to locate an attachment node requires $\Omega(\sqrt{\ell})$ SP-queries in the worst case, where ℓ is the number of leaves in the current sub-skeleton.*

Proof Our argument is a generalization of a similar claim made by Hein (1989) about their algorithm for conventional trees.

Suppose the current sub-skeleton is a binary tree of the form shown in Figure 4, where $[v_1]$ is the node in the lower left corner. Furthermore, suppose that v_{next} has to be inserted at the green leaf. If $[v_\perp]$ is always chosen to be the most distant vertex from $[v_1]$ in the subtree containing the vertex v_{next}, then the leaf labeled i in the figure is a candidate to be chosen as $[v_\perp]$ in the i-th step. Scaling up this example, it follows that $\Theta(\sqrt{\ell})$ SP-queries could be used in the worst case. ■

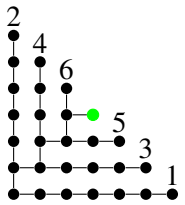

Figure 4

Proposition 20 *There are classes of hypergraphs for which (i) the D_2-query and D_n-query complexity are asymptotically larger than the SP-query complexity, (ii) the SP-query and D_2-query complexity are asymptotically larger than the D_n-query complexity. In particular, claim (i) is witnessed by the class of orderly hypertrees of diameter at least three.*

This proposition is an immediate consequence of the following two lemmas.

Lemma 25 *Let $\mathcal{H}$ be the class of all hyperpaths (which are, in particular, orderly hypertrees) of diameter at least three. Learning a target hypergraph in $\mathcal{H}$ requires $\Omega(n^2)$ D_n-queries ($\Omega(n^2)$ D_2-queries, resp.) in the worst case. By contrast, the worst-case number of SP-queries required to learn any target hypergraph in $\mathcal{H}$ is in $\Theta(n)$.*

Proof The SP-query complexity was proven by Fallat et al. (2024)).

The quadratic lower bound on the D_n-query complexity can be derived as follows. For any $n \geq 4$, consider a hyperpath P_n (Figure 5) of length three with two edges e_1 and e_2 of cardinality $\frac{n}{2}$ and one edge e of cardinality 2 with no private vertices.

For learning such a hyperpath, dependency queries with more than two vertices are useless. Each set of size greater than 2 has at least two vertices in the same edge, so that any D_q-query with $q > 2$ will always receive the answer 1. Hence, consider a learner asking only D_2-queries to identify P_n. Clearly, against an adversarial oracle, such learner must ask one query for each set $\{v_1, v_2\}$ where $v_i \in e_i$. This yields the claimed bound of $\Omega(n^2)$ for D_n-queries. ■

Lemma 26 *Let $\mathcal{H}$ be the class of all hypergraphs consisting of one edge of size 2 and $n-2$ isolated vertices. Learning a target hypergraph in $\mathcal{H}$ requires $\Omega(n^2)$ SP-queries ($\Omega(n^2)$ D_2-queries, resp.) in the worst case. By contrast, the worst-case number of D_n-queries required to learn any target hypergraph in $\mathcal{H}$ is in $\Theta(\log n)$.*

Proof By a standard information-theoretic argument, a lower bound of $\Omega(\log n)$ on the D_n-query complexity is obtained. The matching upper bound follows from a result by Angluin and Chen (2008), who showed that a graph with edges of size at most 2 can be learned with $O(\log n)$ edge detection queries per edge. The definition of edge detection queries coincides with the definition of D-queries when the underlying hypergraphs are conventional graphs. Since every hypergraph in $\mathcal{H}$ is in fact a graph, this upper bound translates to D_n-queries.

Next, consider any learner asking only SP-queries. If the learner asks at most $\binom{n}{2} - 2$ queries, there exist at least two sets $\{v_1, v_2\}$ and $\{v_3, v_4\}$ of size 2 each, with $\{v_1, v_2\} \neq \{v_3, v_4\}$, neither of which is posed as a query. Now there are at least two distinct hypergraphs in $\mathcal{H}$ for which all

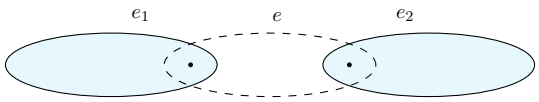

Figure 5: Hyperpath P_n from the proof of Proposition 20.

posed queries would have been answered ∞ by the oracle (one containing the edge $\{v_1, v_2\}$, and one containing the edge $\{v_3, v_4\}$). Thus, in the worst case, the learner must ask $\binom{n}{2} - 1$ SP-queries to identify its target. Since D_2-queries provide no more information than SP-queries, the same lower bound applies to them. ■

Theorem 21 ***(Part 1.)*** *Let $\mathcal{H}$ be the class of all connected orderly hypergraphs of order n and diameter at least 3, with each edge containing at least one private vertex. The worst-case number of D_2-queries needed to learn any target hypergraph $H \in \mathcal{H}$ is in $O(mn)$, where m is the number of edges in H.*

Proof The claim is witnessed by the following algorithm for learning any $H \in \mathcal{H}$:

(1) Initialize $i = 0$, $V^* = V$ and repeat the following until $V^* = \emptyset$:

(a) Select a vertex $v_i \in V^*$ and ask a total of $n-1$ D_2-queries of the form $\{v_i, v'\}$ – one for each vertex $v' \in V$ with $v' \neq v_i$.

(b) Let $V_i \subseteq V$ be the set of vertices v' for which such query is answered 1. (Note that V_i is the union of all edges in H containing v_i.) Set $V^* := V^* \backslash V_i$ and $i := i + 1$.

Since H is orderly, the sub-hypergraph induced by V_i is an orderly hyperstar (possibly just a single edge). For each edge e in H, since e has at least one private vertex, there must exist an i such that $e \subseteq V_i$.

(2) Since H is connected with diameter at least 3, each set V_i intersects with at least one set V_j, $j \neq i$. Thus, for each set V_i:

(a) Select a vertex $w_i \in V_i$ that belongs to some set V_j, $j \neq i$. For each vertex $w \in V_i$, ask one D_2-query of the form $\{w_i, w\}$.

(b) If all queries are answered 1, then V_i is a single edge. Otherwise, the vertices in V_i form a hyperstar H_i within H; moreover, the vertex w_i has maximum eccentricity in H_i. (Note that the intersections of V_i with any V_j are known at this point.)
We can now learn H_i with at most $O(n_i m_i)$ D_2-queries, where $n_i = |V_i|$ is the number of vertices in H_i, and m_i is the number of edges in H_i. To do so, note that the edge e in H_i that contains w_i is already determined by the queries in (2a). The other edges of H_i are learned by picking any vertex $u \in V_i \backslash e$ and asking D_2-queries of the form $\{u, w\}$ for each vertex $w \in V_i$. Continuing this process, each time picking a vertex $u \in V_i$ that does not belong to the previously learned edges of H_i, all edges of H_i are learned. The total number of D_2 queries in this step is in $O(\sum_i m_i n_i)$ which can be bounded by $O(n \sum_i m_i) = O(mn)$.

Hence the total number of D_2-queries consumed in order to learn H is in $O(mn)$. ■

Theorem 21 ***(Part 2.)*** *Dropping any of the four premises of Theorem 21 (Part 1) in isolation would yield quadratic D_2-query complexity.*

Most of the claims needed to prove this remark have already been established. Firstly, dropping the condition on the diameter yields the class of all orderly hyperstars, which is known to have an SP-query (and thus D_2-query) complexity of $\Theta(n^2)$. Secondly, learning arbitrary connected orderly hypergraphs of diameter ≥ 3 also needs $\Omega(n^2)$ D_2-queries—even $\Omega(n^2)$ D_n-queries—if the condition on private vertices is dropped. This is witnessed by hyperpaths of diameter ≥ 3 (see Lemma 25). Not necessarily connected orderly hypergraphs of diameter at least three are hard to learn with D_2-queries, even if each edge has private vertices; this is witnessed by Lemma 26. Finally, forgoing orderliness yields:

Lemma 27 *Let $\mathcal{H}$ be the class of all connected hypergraphs of diameter at least 3 with each edge containing at least one private vertex. Learning a hypergraph $H \in \mathcal{H}$ requires $\Omega(n^2)$ D_n-queries in the worst case, even for fixed $m \geq 4$.*

Proof For any $n \geq 7$, consider the hypergraph H_n in Figure 6. Each edge in H_n has exactly one private vertex, $|e_1 \cap e_j| = 1$ for $j \in \{0, 2, 3\}$, and the intersection of e_2 and e_3 consists of $n-5 \geq 2$ vertices, one of which belongs to e_1.[3]

Consider any sequence S of D_n-queries. The answers obtained in this sequence S may or may not determine some vertices as members of $e_0 \cup e_1$. Suppose there are two distinct vertices v, w that are not determined as belonging to $e_0 \cup e_1$. This is possible as long as S has at most $n-5$ queries containing exactly one of the two vertices v and w.

Now there are two distinct hypergraphs H, H' in $\mathcal{H}$, both isomorphic to H_n, for which the query sequence S would receive identical sequences of answers from the oracle. In other words, S would not be able to distinguish H from H'.

H and H' differ only in the placement of v and w. H has w as a private vertex in e_2, while v lies in $(e_2 \cap e_3) \backslash e_1$; in H', it is the opposite.

Clearly, any query containing either (i) both v and w or (ii) neither of them will be answered the same both for H as target and for H' as target. The same is true for any query containing either v or w, together with any vertex that S determines to belong to $e_0 \cup e_1$. If S has at most $n-5$ queries containing exactly one of v and w, it has at most $n-5$ queries containing exactly one of v and w together with vertices that S does *not* determine as members of $e_0 \cup e_1$. We can now assume that all queries of the latter type in S are answered with 1, in particular, that the private vertex of edge e_3 is not among the vertices queried together with v or w. Hence H and H' cannot be distinguished through such a query sequence S.

This implies that, to learn H_n with a sequence S of D_n-queries, S must have, for any two distinct vertices v, w outside $e_0 \cup e_1$, more than $n-5$ distinct queries containing exactly one of v and w. Since there are $\Omega(n^2)$ choices for such v, w, this proves the claimed lower bound of $\Omega(n^2)$ D_n-queries. ■

Theorem 23 *Let $\mathcal{H}$ be the class of all (not necessarily connected) orderly hypergraphs whose components have diameter ≥ 3, and contain no vicious 3-cycles as induced sub-hypergraphs. Then there is an algorithm that learns every $H \in \mathcal{H}$ using n ED-queries and $O(nm)$ D_2-queries.*

3. The same structure was used in the Figure 1; we replicate the figure here and add annotation, for ease of reading.

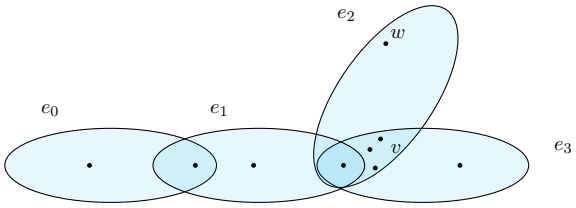

Figure 6: The hypergraph H_n from the proof of Lemma 27.

This theorem is an immediate consequence of the following three lemmas. Jointly, they clearly establish Theorem 23, as desired.

These lemmas make use of the notion of hitting set. A hitting set in a hypergraph $H = (V, E)$ is a set $T \subseteq V$ such that for all edges $e \in E(H)$, $T \cap e \neq \emptyset$. A hitting set T of H is called minimal if no proper subset of T is a hitting set.

Lemma 28 *Given a hitting set T for an unknown hypergraph H, a learning algorithm can infer a minimal hitting set $T' \subseteq T$ for H with $|T|$ ED-queries. In particular, there is an algorithm that learns a minimal hitting set for any unknown hypergraph over n vertices, by asking at most n ED-queries.*

Proof Let $T = \{v_1, \ldots, v_t\}$, $t = |T|$. The claim is witnessed by the following learning procedure.

1. Set $i = 1$ and $T' = T$.
2. Set $T_i := T' \backslash \{v_i\}$ and ask an *ED*-query for $V \backslash T_i$.
3. If the answer is 0, set $T' := T' \backslash \{v_i\}$, $i := i + 1$. If $i \leq t$ then go to step (2), otherwise, if $i > t$ then stop and return T'.
4. If the answer is 1, set $i := i + 1$. If $i \leq t$ then go to step (2), otherwise, if $i > t$ then stop and return T'.

Clearly, $T' \subseteq T$, and T' is a minimal hitting set for H.

Since the full vertex set V is always a hitting set of size n for H, one can learn minimal hitting sets using at most n *ED*-queries. ■

Lemma 29 *Any minimal hitting set for a hypergraph H with $\leq m$ edges has size at most m.*

Proof First we show that any subset of a minimal hitting set for H is a minimal hitting set for the sub-hypergraph induced by it. To this end, let T be a minimal hitting set for H, and let $T' \subset T$. If T' were not a minimal hitting set for the sub-hypergraph H' it induces, then there would be a vertex $v \in T'$ such that $T' \backslash \{v\}$ is a hitting set for H'. But then $T \backslash \{v\} = T' \backslash \{v\} \cup (T \backslash T')$ is a hitting set for H, which contradicts the minimality of T.

Second, note that any minimal hitting set T for H contains a vertex v such that there is an edge e that contains v but no other vertex in T.

These two properties together imply that a minimal hitting set has size no larger than m. ■

Lemma 30 *Let $\mathcal{H}$ be the class of all (not necessarily connected) orderly hypergraphs H over n vertices such that each component of H has diameter at least 3, and H contains no vicious 3-cycles as induced sub-hypergraphs. There is an algorithm which, given a size t hitting set for any target hypergraph $H \in \mathcal{H}$, can learn H using at most $n(m+2t-1)$ D_2-queries, where m is the number of edges in H.*

Proof Suppose that $T = \{v_1, \ldots, v_t\}$ is a hitting set of the target hypergraph $H^* \in \mathcal{H}$. For each $i \in [t]$, we define $V_i := \{v \in V \mid d(v, v_i) = 1\}$, and call the sub-hypergraph $C_i = (V_i, E_i)$ induced by V_i the *cluster* of v_i. Note that C_i is a hyperstar, $V = \bigcup_{i=1}^t V_i$ and $E = \bigcup_{i=1}^t E_i$.

For each cluster C_i, the vertex set V_i can be identified with at most n D_2-queries, so that no more than nt D_2-queries suffice to learn $V_1, \ldots, V_n$. (The strategy for learning V_i here is the same as Step 1 in the proof of Theorem 21.)

Next, we describe how to learn all edges in each cluster C_i. Since the diameter of each component is at least 3, V_i intersects at least one set V_j for $j \neq i$. The learner asks a D_2-query of the form $\{v_i, v_j\}$.

Case 1. The answer to the query $\{v_i, v_j\}$ is 1. Then the intersection $V_{i,j} := V_i \cap V_j$ contains a whole edge $e_{i,j}$ including both v_i and v_j. To learn all edges of the clusters C_i and C_j, the learner selects an arbitrary vertex $w_1 \in V_i \backslash V_{i,j}$. Since the hypergraph H has no vicious 3-cycles as induced sub-hypergraphs, the set $V_i \backslash V_{i,j}$ is nonempty.

The learner then asks D_2-queries of the form $\{w_1, v\}$ for all $v \in V_i$. In this way, it learns an edge e_1 in C_i. If $V_i \backslash (V_{i,j} \cup e_1)$ is nonempty, the learner picks an arbitrary vertex $w_2 \in V_i \backslash (V_{i,j} \cup e_1)$ and continues this process to learn all edges in C_i except for $e_{i,j}$. Analogously, it also learns all edges in C_j except for $e_{i,j}$. Finally, it identifies $e_{i,j}$ as $V_{i,j} \backslash K$, where K is the intersection of all learned edges in C_i and C_j.

Case 2. The answer to the query $\{v_i, v_j\}$ is 0. Then $V_i \cap V_j$ does not contain an entire edge. The learner selects an arbitrary vertex $u_1 \in V_i \cap V_j$ and asks D_2-queries of the form $\{u_1, v\}$ for all $v \in V_i$, allowing it to identify an edge e_1 in C_i. Then if $V_i \backslash e_1$ is nonempty, the learner chooses an arbitrary vertex u_2 in $V_i \backslash e_1$ and continues this process to learn all edges in C_i.

Applying the above process, all edges of the cluster C_i are learned by asking at most $|E_i| \cdot |V_i|$ D_2-queries. Hence the total number of D_2-queries to learn H^* is bounded from above by

$$tn + \sum_{i=1}^{t} |E_i| \cdot |V_i| \leq nt + \max_{1 \leq i \leq t} |V_i| \sum_{j=1}^{t} |E_i| \leq nt + \max_{1 \leq i \leq t} |V_i|(m + t - 1) \leq n(m + 2t - 1)\,.$$

■