# Learning Hypertrees From Shortest Path Queries

**Shaun M. Fallat** SHAUN.FALLAT@UREGINA.CA
**Valerii Maliuk** VALERA.MALUK@GMAIL.COM
**Seyed Ahmad Mojallal** AHMAD_MOJALAL@YAHOO.COM
**Sandra Zilles** SANDRA.ZILLES@UREGINA.CA
*University of Regina, Canada*

**Editors:** Claire Vernade and Daniel Hsu

## Abstract

We consider the problem of learning a labeled hypergraph from a given family of hypergraphs, using shortest path (SP) queries. An SP query specifies two vertices and asks for their distance in the target hypergraph. For various classes $\mathcal{H}$ of hypertrees, we present bounds on the number of queries required to learn an unknown hypertree from $\mathcal{H}$. Matching upper and lower asymptotic bounds are presented for learning hyperpaths and hyperstars, both in the adaptive and in the non-adaptive setting. Moreover, two non-trivial classes of hypertrees are shown to be efficiently learnable from adaptive SP queries, under certain conditions on structural parameters.

**Keywords:** Hypergraph, Hypertree, Shortest path query, Graph learning

## 1. Introduction

Hypergraphs are used for representing dependencies between data objects in various applications. For example, hyperedges are a means of representing chemical interactions in computational biology (Murgas et al., 2022), or of user groups in recommender systems (Wang et al., 2022). Hypergraphs have been deployed in knowledge representation, constraint satisfaction, machine learning, and data analysis; their structural and algorithmic aspects are the focus of numerous studies in graph theory (Berge, 1976; Bretto, 2013).

The focus of this paper is on learning hypergraphs over a known set of vertices, using queries. In particular, a learning algorithm poses a query that is answered by an oracle, thus revealing partial information about an unknown target hypergraph $H^*$ from a given class $\mathcal{H}$ of hypergraphs. The goal of the learning algorithm is to identify all hyperedges in $H^*$ with a small number of queries.

This kind of setting has been studied extensively for learning (non-hyper-)graphs (Alon and Asodi, 2005; Alon et al., 2004; Angluin and Chen, 2008; Reyzin and Srivastava, 2007), where various types of queries have been considered. An edge detection query, for instance, is formulated as a set $V_0$ of vertices; the oracle will provide the information whether or not any two vertices in $V_0$ are adjacent in $H^*$, i.e., whether or not an edge exists among vertices in $V_0$ (Alon and Asodi, 2005). By contrast, an edge counting query for a vertex set $V_0$ asks for the *number* of edges in $H^*$ that connect vertices in $V_0$ (Reyzin and Srivastava, 2007). Betweenness queries, which have been used both for learning and for verifying graphs (Abrahamsen et al., 2016; Janardhanan, 2017), specify three vertices in order to find out whether the first vertex lies on a shortest path between the other two. In a shortest path (SP) query, the learning algorithm selects two vertices $v$ and $w$, and the oracle responds with the distance of $v$ and $w$, i.e., the length of a shortest path between them (Erlebach et al., 2006; Reyzin and Srivastava, 2007; Mathieu and Zhou, 2013; Kannan et al., 2015).

SP queries are particularly natural in bioinformatics applications, for example in the context of learning evolutionary trees, where the distance between two species can be determined through research, thus providing an oracle to the learning algorithm (King et al., 2003; Hein, 1989).

Query learning of hypergraphs has been studied primarily with a focus on edge detection queries (Angluin and Chen, 2006; Abasi, 2018; Abasi et al., 2018; Balkanski et al., 2022). By contrast, to the best of our knowledge, there is no literature on learning hypergraphs with SP queries. Our study addresses the question how many SP queries are needed in order to identify an unknown hypergraph in an underlying class of potential target hypergraphs. Intuitively, the class of all hypergraphs over $n$ vertices does not admit efficient learning algorithms, i.e., the required number of SP queries for learning arbitrary hypergraphs is quadratic in $n$. Therefore, we restrict the topological structure of the hypergraphs under consideration. Specifically, we focus on learning hypertrees—a type of hypergraph expressive enough to be useful for a variety of applications (e.g., biochemical interaction networks), yet simple enough to potentially allow for efficient learning algorithms.[1]

To gain insights into the problem of hypertree learning, we first consider two special cases, namely hyperpaths, which are hypertrees with smallest possible maximum degree, and hyperstars, which are hypertrees with largest possible maximum degree. We show that neither of these simple forms of trees can be learned efficiently if the learning algorithm is non-adaptive, i.e., submits all its queries upfront in a batch. In the adaptive setting, where the answers to previous queries can affect the learner's next queries, it turns out that $O(n)$ SP queries suffice to learn hyperpaths of at least 3 edges, while hyperstars have a query complexity of $\Theta(mn + cn)$. Here $m$ is the number of edges in the hyperstar and $c$ is the size of the intersection of all these edges.

Turning to more complex structures, we provide two adaptive learning algorithms for interesting classes of hypertrees.

The first one is a recursive procedure for learning any binary hypertree of diameter at least 3; binary hypertrees are hypertrees in which each edge overlaps with at most three other edges and each vertex has degree at most 3. This algorithm uses $O(dn)$ SP queries, where $d$ is the unknown diameter of the target hypertree.

The second algorithm is purely iterative, and learns hypertrees of a more uniform structure; specifically, it assumes that all edges have identical size and all edge intersections have identical size. While this type of hypertree may be less useful for real-world applications, our learning algorithm provides insights into which structural properties affect the complexity of hypertree learning.

In the sum, our paper provides the first systematic study of learning hypergraphs (specifically, various classes of hypertrees) from SP queries. For various topological structures, we provide matching upper and lower bounds on the SP query complexity, and we provide efficient learning algorithms for various sub-classes of hypertrees.

## 2. Preliminaries

A natural generalization of graphs, where edges are made up of 2-subsets of vertices, is the notion of a hypergraph. For our purposes, a *hypergraph* $H$ consists of a pair $(V, E)$ of sets, where $V$ is a finite nonempty set (whose elements are called *vertices*) and the edge-set $E \subseteq \mathcal{P}(V)$ where $\mathcal{P}(V)$ is the power set of $V$. We assume that $e \not\subset e'$ for any two $e, e' \in E$. In the special case when every edge in $E$ has the same cardinality we refer to $H$ as *uniform hypergraph*. In particular, if every

1. We consider only hypertrees in which the intersection of three or more edges is either empty or equal to the intersection of any two of these edges.

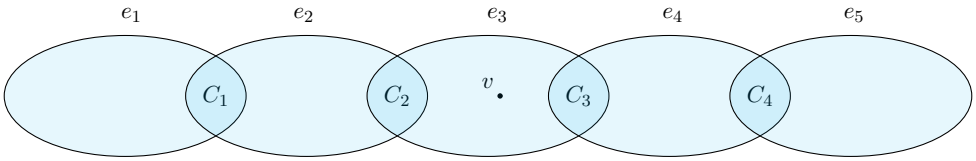


Figure 1: If $|C_i| \leq c$ for all $i$, the depicted hypergraph is a $(\leq c)$-hyperpath with $m = 5$ edges. Assuming $c = |C_i|$ for all $i$, it is a $c$-hyperpath. Vertex $v$ belongs to edge $e_3$, but not to any other edge.

edge $e \in E$ contains precisely $r$ vertices, then $H$ is said to be *$r$-uniform*. Finally, the *order* of a hypergraph $H$ is simply the cardinality of $V$, that is, $|V|$.

Throughout this work, we make use of various terms associated with hypergraphs. For ease of exposition, we now list several such necessary terms. Suppose $H = (V, E)$ is a given hypergraph. Then two vertices $u$ and $v$ are said to be *adjacent* if there is an edge $e$ of $H$ with $\{u, v\} \subseteq e$, and in this case, we say $e$ is *incident* to both $v$ and $u$.

Our main focus is concerned with distance in a hypergraph. Consequently, we carefully highlight the notion of a path in a hypergraph. To this end, a *path* $P$ in a hypergraph $H$ from $v_1$ to $v_{s+1}$ is an alternating sequence $v_1 e_1 \cdots v_s e_s v_{s+1}$ in which $v_1, \ldots, v_{s+1} \in V$ are distinct vertices, $e_1, e_2, \ldots, e_s$ are distinct edges of $H$, and for $i = 1, 2, \ldots, s$, $v_i, v_{i+1} \in e_i$. Further, we say the *length* of $P$ is $s$. Analogous to conventional graphs, a hypergraph $H$ is said to be *connected* if, for any two vertices $u$, $v$ in $H$, there is a path from $u$ to $v$. For any connected hypergraph $H$, we set the *distance*, $d(x, y)$, between two vertices $x$ and $y$ as the minimum length of a path from $x$ to $y$. Similar to graphs, the maximum distance between a vertex $v \in V$ and any other vertex in $V$ is called the *eccentricity* of the vertex $v$ and is denoted by $ecc(v)$. The *radius* (*diameter*, resp.) of the hypergraph $H$ is defined as the minimum (maximum, resp.) eccentricity of any vertex. Finally, the *center* of a hypergraph is the set of all its vertices of minimum eccentricity.

Beyond paths we also consider cycles in a hypergraph. Thus if $v_1 e_1 \cdots v_{s-1} e_{s-1} v_s$ is a path from $v_1$ to $v_s$ and $s > 2$ then $v_1 e_1 \cdots v_s e_s v_{s+1}$, $v_1 = v_{s+1}$, is called a *cycle of length $s$*.

More generally, for any $c \geq 1$, a $c$-hyperpath is a hypergraph with edge set $\{e_1, \ldots, e_m\}$ such that the edges can be ordered so that $|e_i \cap e_{i+1}| = c$ for all $i = 1, \ldots, m-1$, and $e_i \cap e_j = \emptyset$ whenever $|i - j| > 1$. A $c$-hyperstar is a hypergraph with edge set $\{e_1, \ldots, e_m\}$ such that $|\bigcap_{i=1}^m e_i| = c$ and $e_j \cap e_{j'} = \bigcap_{i=1}^m e_i$ for all $j, j'$. When allowing $|e_i \cap e_{i+1}| \leq c$ (or $|\bigcap_{i=1}^m e_i| \leq c$, resp.), we instead use the term $(\leq c)$-hyperpath (or $(\leq c)$-hyperstar, resp., where a hypergraph is a $(\leq c)$-hyperstar iff it is a $c'$-hyperstar for some $c' \leq c$), see Fig. 1 for illustration. These types of hypergraphs are special cases of $c$-hypertrees and $(\leq c)$-hypertrees, defined as follows.

A 1-hypertree is a connected hypergraph without cycles. (The absence of cycles implies that no two edges overlap in more than one vertex.) For $c > 1$, a hypergraph $H'$ is a $c$-hypertree ($(\leq c)$-hypertree, resp.) iff $H'$ results from a 1-hypertree $H$ by adding $c - 1$ vertices (up to $c - 1$ vertices, resp.) to the intersection of any maximal set of intersecting edges in $H$. In such $H'$, for any three edges $e_1$, $e_2$, $e_3$ in $H'$, $e_1 \cap e_2 \cap e_3$ is either empty or equals the intersection between any two of the three edges.

If $H$ is any hypergraph, then the *degree* of an edge $e$ in $H$ is the number of edges in $H$ that intersect $e$. Thus, in a $c$-hyperpath with $m \geq 2$ edges, there are exactly two edges of degree 1; all

other edges have degree 2. In a $c$-hyperstar with $m$ edges, each edge has degree $m-1$. A hypertree is called a *binary* hypertree if its maximum edge degree is at most 3 and no vertex belongs to more than 3 edges.

### 2.1. Learning From SP Queries

We consider the problem of using a smallest possible number of shortest path (SP) queries to learn the edge set of a hidden hypergraph $H^*$ for which the vertex set $V$ is known. In the underlying model, a learning algorithm (called learner) issues SP queries in the form of pairs $(u, v) \in V^2$ to an oracle; the oracle will then respond with the value $d(u, v)$, i.e., the learner is told the length of a shortest path between $u$ and $v$.

Adaptive learning proceeds sequentially; at any point in time, the learner is allowed to make the choice of its next query dependent on the responses to previously asked queries. By contrast, in non-adaptive learning, the learner fixes its entire set of queries in advance.

This paper focuses on learning hypertrees; in particular, it provides upper and lower bounds on the smallest possible number of SP queries needed to learn specific subclasses of hypertrees. The formal proofs of these bounds will require some additional definitions and notation.

**Definition 1** *Given a hypergraph $H$ with vertex set $V$ and edge set $E$, the* query graph $G_H$ *(at any point in time during learning) is defined to be a simple graph with the vertex set $V$ such that any two vertices $u$ and $v$ are adjacent iff the learner has asked the query $(u, v)$ or $(v, u)$ in the past.*

For a hypergraph $H$ and a vertex $v$ of $H$, the term $d(v, \_)$, called the distance sequence of $v$, denotes the string of distance numbers between the vertex $v$ and all other vertices, sorted in increasing order. For example, if $H$ has ten vertices including $v$, and the distances from $v$ to the remaining nine vertices are $1, 1, 1, 2, 2, 3, 3, 3, 4$, then $d(v, \_) = 1^3, 2^2, 3^3, 4^1$.

For any graph and any two vertices $u$ and $v$ in the graph, we define the *joint neighborhood of $u$ and $v$* to be the union of the neighborhood of $u$ and the neighborhood of $v$, minus $\{u, v\}$.

## 3. Learning hyperpaths

Our first observation is that non-adaptive query learning is too weak a setting for efficient learning even of very restricted forms of hyperpaths. In particular, we obtain a quadratic lower bound already for $r$-uniform $c$-hyperpaths; this bound remains quadratic in $n$ when $c = 1$:

**Theorem 2** $\Omega(n^2 - n(r - c + 1))$ *SP queries are needed to non-adaptively learn an $r$-uniform $c$-hyperpath, where $r > 2c$.*

**Proof** Let $H$ be an $r$-uniform $c$-hyperpath, with $r > 2c$, and assume the edges of $H$ are labelled $e_1, \ldots, e_m$. As in Fig. 1, we let $K_i$ ($i = 1, 2, \ldots, m-1$) be the intersection of $e_i$ and $e_{i+1}$. If the query graph constructed by the learner contains two non-adjacent vertices $u$ and $v$ whose joint neighborhood is of size at most $n-r+c-2$, then the oracle may select two $r$-uniform $c$-hyperpaths $H_1$ and $H_2$ such that $u$ and $v$ are in the $e_2$ with one of them in $e_2 \setminus e_1$ and the other in $K_1 = e_1 \cap e_2$. Let $V_1$ be the set of vertices that the learner has placed in the joint neighborhood of the non-adjacent vertices $u$ and $v$. Hence the size of $V_1$ is at most $n-r+c-2$. Then the oracle places all vertices in $V_1$ within the edges $e_2$ to $e_m$, with $c-1$ vertices in $K_1$, $c$ vertices in each $K_i$ for $i \in \{2, \ldots, m-1\}$,

and the remaining vertices are placed in other non-shared sections. Removing the vertices $u$ and $v$ leaves $n-r+c-2$ vertices in the edges $e_i$ where $i \in \{2,\ldots,m\}$. Thus the oracle has sufficient freedom to label the vertices in $V_1$ as desired. Furthermore, the oracle can freely swap the vertices $u$ and $v$; both choices will result in exactly the same answers to the queries asked by the learner. Consequently, the learner, when receiving responses from the oracle, will be unable to determine which of $u$ and $v$ to place in $K_1$ and which to place in $e_2 \setminus e_1$. Therefore, the learner is not able to learn $H$ if the query graph constructed by the learner contains two non-adjacent vertices $u$ and $v$ whose joint neighborhood is of size at most $n-r+c-2$.

Therefore, a successful set of queries must yield a query graph $G$ in which any two vertices $u$ and $v$ are adjacent or otherwise they have a joint neighborhood of size at least $n-r+c-1$. Observe that such a query graph may be disconnected even with one isolated vertex (of course $G$ can not have two isolated vertices by assumption). Hence the query graph $G$ has at least $\Omega(n^2-n(r-c+1))$ edges.

Suppose that $G$ is the query graph induced by $m(G)$ SP queries. Hence the minimum number of edges for such a query graph gives a lower bound for the minimum number of SP queries.

If every vertex has a degree at least $\frac{n-r+c-1}{2}$, then we have

$$2m(G)=\sum_{i=1}^{n} d_i(G) \geq \frac{n\,(n-r+c-1)}{2}, \text{ which implies } m(G) \geq \frac{n^2-nr+nc-n}{4}.$$

Otherwise, there exists a vertex $v$ with degree less than $\frac{n-r+c-1}{2}$. Then applying the conclusion above, the degree of each vertex that is not adjacent to $v$ is strictly greater than $\frac{n-r+c-1}{2}$. Note there are at least $n-\frac{n-r+c-1}{2}$ such vertices that are not adjacent to $v$, so

$$2m(G) \geq \frac{n-r+c-1}{2}\big(n-\frac{n-r+c-1}{2}\big),$$

that is,

$$m(G) \geq \frac{n^2-(r-c+1)^2}{8}.$$

The lower bounds on $m(G)$ imply that the learner must ask $\Omega(n^2-n(r-c+1))$ non-adaptive SP queries to fully identify the hypergraph $H$. ■

By contrast, for the adaptive setting, we will prove below that all $(\leq n)$-hyperpaths can be learned with a linear number of queries, as long as they have at least three edges. For two overlapping edges however, we obtain a lower bound with an additional factor of $c$, where $c$ is the size of the intersection of the two edges. We state this result explicitly, as it will be useful later on.

**Lemma 3** *Let $\mathcal{H}$ be the class of $c$-hyperpaths of order $n$ with exactly 2 edges, in which*

- *one edge ($e_1$) contains a single vertex $v^*$ in addition to a set $C$ of $c=|C|$ center vertices, and*
- *the second edge ($e_2$) contains the remaining $n-c-1$ vertices, in addition to $C$.*

*Then $\Omega(cn)$ SP queries are needed to adaptively learn any member of $\mathcal{H}$.*

**Proof** Note that the distance sequence $d(v, \_)$ for a vertex $v$ satisfies (i) $d(v^*, \_) = 1^c, 2^{n-1-c}$, (ii) $d(v, \_) = 1^{n-2}, 2^1$, if $v \notin C \cup \{v^*\}$, and (iii) $d(v, \_) = 1^{n-1}$, if $v \in C$.

Now suppose there are two distinct vertices $v, v'$ such that a learner $L$ asks at most $c$ queries containing $v$ or $v'$. This corresponds to a query graph in which the joint neighborhood of $v$ and $v'$ has size at most $c$. Consider two hypergraphs $H_1, H_2 \in \mathcal{H}$ for which $d(v, v') = d(v, u) = d(v', u) = 1$ for all $u$ in the joint neighborhood of $v$ and $v'$: in $H_1$, the role of $v^*$ is played by $v$, while $v'$ lies in $C$; in $H_2$, it is the opposite. In both $H_i$, all vertices adjacent to $v^*$ in the query graph are members of $C$.

Hence, if the oracle answers 1 to all queries made by $L$, then $L$ cannot uniquely determine the target labeling of the vertices. This means that a successful learner must ask enough queries so that all pairwise joint neighborhoods in the query graph have size greater than $c$. This requires at least $\frac{1}{2}n(c+1)$ queries. We thus obtain the desired lower bound of $\Omega(cn)$. ■

As a hyperpath of two edges is also a hyperstar, we immediately obtain the following corollary.

**Corollary 4** *$\Omega(cn)$ SP queries are needed to adaptively learn $c$-hyperpaths of order $n$ with exactly 2 edges. Likewise, $\Omega(cn)$ SP queries are needed to adaptively learn $c$-hyperstars of order $n$ with exactly 2 edges.*

As promised earlier, we now show that a linear number of queries suffices for learning hyperpaths of at least 3 edges.

**Theorem 5** *The minimal number of adaptive SP queries for learning any member of the class of all ($\leq n$)-hyperpaths that have $n$ vertices and at least 3 edges is in $\Theta(n)$. In particular, $3n - 6$ queries are sufficient for identifying any member in this class.*

**Proof** The lower bound $\Omega(n)$ is obvious. To verify that $3n - 6$ queries suffice, consider a learner $L$ that asks the following queries:

1. $L$ selects any vertex $v$ and asks all $n - 1$ queries of the form $(v, v')$ for $v' \in V \setminus \{v\}$. Thus, $L$ obtains the distance sequence $d(v, \_)$.
2. $L$ selects any vertex $v_1$ whose distance to $v$ is maximal, and asks all $n - 2$ queries of the form $(v_1, v')$ for $v' \in V \setminus \{v, v_1\}$. Thus, $L$ obtains the distance sequence $d(v_1, \_)$.
3. $L$ selects any vertex $v_2$ whose distance to $v_1$ is maximal (i.e., this distance equals the diameter $d$ of $H^*$), and asks all $n - 3$ queries of the form $(v_2, v')$ for $v' \in V \setminus \{v, v_1, v_2\}$. (If $v$ happens to be a vertex at maximal distance to $v_1$, these $n - 3$ queries are not needed.) Thus, $L$ obtains the distance sequence $d(v_2, \_)$.

Then $L$ has asked at most $3n - 6$ queries, and the information thus obtained is sufficient to identify the target hyperpath. To see this, first note that $v_1$ and $v_2$ are vertices "at opposite ends" of the target hyperpath $H^*$. That means, we can determine the edge set of $H^*$ as $\{e_1, \ldots, e_d\}$, where (i) $v_1 \in e_1 \setminus \{e_2, \ldots, e_d\}$, (ii) $v_2 \in e_d \setminus \{e_1, \ldots, e_{d-1}\}$, and (iii) $e_i$ intersects $e_{i+1}$ for $1 \leq i \leq d - 1$. For every vertex $v'$ in $H^*$ and each $i \in \{1, \ldots, d\}$, we have:

- $v' \in e_i \setminus (e_{i-1} \cup e_{i+1})$ iff $d(v_1, v') = i$ and $d(v_2, v') = d - i + 1$;
- $v' \in e_i \cap e_{i+1}$ iff $d(v_1, v') = i$ and $d(v_2, v') = d - i$;

where we assume $e_0 = e_{d+1} = \emptyset$. Thus, $H^*$ is fully identified. ■

## 4. Learning hyperstars

Hyperpaths can be seen as an extreme case of (degenerate) hypertrees. In the same vein, hyperstars form the opposite extreme. It is thus of interest to contrast our bounds on the SP query complexity of hyperpaths with the corresponding bounds for hyperstars.

In the non-adaptive setting, hyperstars again require a quadratic number of queries, even when focusing on $r$-uniform $c$-hyperstars, when $r$ and $c$ are known to the learner. The proof is very similar to that of Theorem 2 and given in Appendix B.

**Theorem 6** $\Omega(n^2 - 2n(r-c))$ *non-adaptive SP queries are needed to learn an $r$-uniform $c$-hyperstar of order $n$, where $r \geq 3$.*

As in the case of hyperpaths, the ability to adapt queries depending on the responses of the oracle reduces the number of queries needed to learn hyperstars. We make use of a result by Liu and Mukherjee (2022), in order to show that the number of SP queries required for learning any member of the class of all (not necessarily uniform) $(\leq c)$-hyperstars of order $n$ is in $\Theta(mn + cn)$, where $m$ is the number of edges in the target hyperstar. This holds irrespective of whether or not $m$ and $c$ are known to the learner.

Liu and Mukherjee (2022) consider the problem of learning partitions of a graph with membership queries. A hidden graph with $n$ vertices is assumed to have $m$ connected components, and the goal of the learner is to find out the set of vertices in each component. A membership query asks whether any two vertices belong to the same component or not. Liu and Mukherjee (2022) show that the learner (in an adaptive setting) can be forced to make at least $(m-1)n - \binom{m}{2}$ queries, if $m$ is known in advance (their Theorem 1.2), and at least $mn - \binom{m+1}{2}$ queries otherwise (their Theorem 1.3). Moreover, both these lower bounds are tight, as Liu and Mukherjee (2022) demonstrate with Algorithm 1, which was originally proposed by Reyzin and Srivastava (2007).

**Algorithm 1 (Reyzin and Srivastava (2007))** *Let $v_0 \in V$ be an arbitrary vertex. Initialize a set of known vertices as $V^* = \{v_0\}$, and a set of representative vertices as $R = \{v_0\}$. Then repeat the following steps until $V^* = V$.*

1. *For every unknown $v \in V \setminus V^*$ and every representative $r \in R$, ask a query for $(v, r)$ to determine whether $v$ and $r$ belong to the same component.*
2. *As soon as one of these queries, say $(v, \bar{r})$, receives the answer* yes*, then add $v$ to $V^*$; we now know that $u$ and $\bar{r}$ are in the same component.*
3. *Else, i.e., if all these queries receive the answer* no*, then add $v$ to $R$; we now treat $v$ as the representative of a newly detected component.*

**Theorem 7 (Liu and Mukherjee (2022))** *Algorithm 1 learns the $m$ components of a hidden graph with $n$ vertices using at most $mn - \binom{m+1}{2}$ queries if $m$ is not known in advance, and can be capped to use at most $(m-1)n - \binom{m}{2}$ queries, if $m$ is known in advance. Both these upper bounds are tight.*

Learning a $c$-hyperstar is not much different than learning the components of a hidden graph. Suppose $C \subseteq V$ is the intersection of all edges in the target hyperstar. The modified edges $e \setminus C$,

for $e \in E$, then form a partition of $V \setminus C$ into $m$ parts. Learning this partition with SP queries is equivalent to learning the $m$ components of a hidden graph with vertex set $V \setminus C$, using membership queries. This is because an SP query on a hyperstar admits only two possible answers: the answer 1 is equivalent to a *yes* for the corresponding membership query; the answer 2 corresponds to a *no*. This means, we can learn a hyperstar by first learning $C$ and then applying Algorithm 1.

**Theorem 8** *The minimum number of SP queries needed for learning any member of the class of all (not necessarily uniform) $(\leq c)$-hyperstars of order $n$ is in $\Theta(mn + cn)$, where $m$ is the number of edges in the target hyperstar. This holds irrespective of whether or not $c$ and $m$ are known to the learner.*

**Proof** The upper bound $O(mn + cn)$ is obtained by the following learner $L$.

First, $L$ picks a vertex $v$ and asks $n - 1$ SP queries to obtain the distance sequence $d(v, \_)$. If $d(v, \_) = 1^{n-1}$, then $v$ is in the intersection $C$ of all edges of the target hyperstar $H^*$; we then repeat this procedure with a new vertex instead of $v$, until we find the first vertex $v'$ for which $d(v', \_) \neq 1^{n-1}$. In this first stage, $L$ consumes at most $(c + 1)(n - 1) \in O(cn)$ queries.

Second, $L$ picks any two vertices $v', v''$ for which it has confirmed $d(v', v'') = 2$. At least one such pair of vertices has been found in the first stage, if any exists. In particular, $v', v'' \notin C$. Now $L$ needs to pose only $O(n)$ additional queries to obtain both $d(v', \_)$ and $d(v'', \_)$. The vertices at distance 1 from both $v'$ and $v''$ now form the set $C$.

In the sum, so far, $L$ has spent $O(cn)$ queries to identify $C$. As argued before the formulation of Theorem 8, it now suffices for $L$ to invoke Algorithm 1 to learn the remaining parts of each edge in $H^*$, which can be done at an additional cost in $O(mn)$, according to Theorem 7. In total, the number of queries posed by $L$ is in $O(mn + cn)$.

It remains to show a lower bound of $\Omega(mn + cn)$. Clearly, the problem of identifying $H^*$ does not become harder when $C$ is known. Therefore, the lower bound of $\Omega(mn)$ for learning graph components (from Theorem 7) applies to the problem of learning hyperstars as well. A lower bound of $\Omega(cn)$ follows immediately from Corollary 4. In combination, we obtain a lower bound of $\Omega(mn + cn)$. ■

**Remark 9** *For adaptive learning of hyperpaths, we showed that the SP query complexity is in $\Omega(cn)$ when the target hyperpaths may consist of only two edges (see Corollary 4). This complexity is reduced to $\Theta(n)$ when the target graph is guaranteed to have at least three edges (Theorem 5). This raises the question whether the query complexity $\Theta(mn+cn)$ in Theorem 8 can also be reduced to $\Theta(mn)$ if the target hyperstar is known to have at least three edges. The answer to this question is 'no'. To see this, one can modify Lemma 3 to apply to a class of hyperstars with three edges, without any change in the lower bound on the query complexity.*

## 5. Learning hypertrees

In this section, we will provide two upper bounds for learning certain families of hypertrees. Each bound is formulated in terms of various parameters of the target hypertree, corresponding to efficient learning algorithms for various subclasses of hypertrees. Our first bound implies that binary $(\leq c)$-hypertrees can be learned with a number of SP queries that is linear in both $n$ and the diameter $d$, while the second bound shows that $r$-uniform $c$-hypertrees have an SP query complexity linear in $r$, $n$, and (roughly) the number of edges at any fixed distance from a maximally eccentric vertex.

**Lemma 10** *Given any unknown $(\leq n)$-hypertree with $n$ vertices and unknown diameter $d \geq 3$, $2n-3$ adaptive SP queries are sufficient to learn the diameter $d$ as well as to determine a vertex of maximum eccentricity.*

**Proof** An algorithm $L$ for learning the diameter of a hidden $(\leq n)$-hypertree $H = (V, E)$ proceeds as follows. First, $L$ selects an arbitrary vertex $v$ and asks $n-1$ queries to obtain the distance sequence $d(v, \_)$. Since $H$ has at least three edges, this distance sequence is not equal to $1^{n-1}$. Now $L$ chooses a vertex, say $w$, whose distance from $v$ equals the maximum distance number in $d(v, \_)$. Then $L$ asks an additional $n-2$ queries to obtain $d(w, \_)$ (notice that $d(w, v)$ was already obtained before). Clearly, $ecc(w) \geq ecc(u)$ for all $u \in V$. Thus, the diameter of $H$ equals the maximum distance number occurring in $d(w, \_)$. This completes the proof. ■

Our first upper bound for adaptively learning hypertrees concerns binary hypertrees with arbitrary diameter $d \geq 3$ and arbitrary size of edge intersections. In the classical graph setting, where edges contain exactly two vertices, a similar learning problem was studied by Hein (1989), who provided an algorithm for learning binary trees adaptively with $O(n \log(n))$ SP queries. Hein's algorithm, however, does not extend in a straightforward manner to hypertrees. At its core, their method uses a notion called "depth" of a node (not to be confused with the typical notion of depth of a node in a tree), which does not translate immediately to the hypertree setting. We hence designed a new algorithm to prove the following theorem.

**Theorem 11** *Let $\mathcal{H}$ be the class of all binary $(\leq n)$-hypertrees of order $n$ and diameter at least 3. Then $O(nd)$ SP queries suffice to adaptively learn any target hypertree $H \in \mathcal{H}$, where $d$ is the diameter of $H$ and is not known to the learner in advance.*

**Proof** By Lemma 10, since $d \geq 3$, a learner $L$ can first determine a vertex $w$ of maximum eccentricity, as well as the diameter $d$, at the expense of $O(n)$ SP queries. Then, $L$ calls a recursive procedure $P$ with three inputs: (i) the vertex set $V$, (ii) the vertex $w$ and its distance sequence $d(w, \_)$, (iii) the diameter $d$. The recursive procedure $P$ works as follows.

The base case applies when $d \leq 2$. Here the target is a binary $(\leq n)$-hypertree with diameter at most 2, which means it is a hyperstar with at most 3 edges. The intersection of all edges consists of exactly the vertices at distance 1 from both $w$ and $w'$ for some vertex $w'$ at maximal distance from $w$; they are known from calculating $d(w, \_)$, thus identifying a suitable choice for $w'$, and then calculating $d(w', \_)$. Now $P$ invokes Algorithm 1 to learn the partition of the remaining vertices into edges; this can be done with $O(mn) = O(3n) = O(n)$ queries (see Theorem 7). The target is now identified and $P$ terminates.

The general case applies when $d \geq 3$. From the calculation of $d(w, \_)$, we obtain a vertex $v$ with $d(w, v) = d$. Clearly, the (unique) hyperpath $H_{wv}$ in $H$ that connects $w$ and $v$ is a longest hyperpath contained in $H$ as a sub-hypergraph. The procedure $P$ now determines $H_{wv}$ as well as, for each maximal sub-hypertree $H'$ of $H$ that consists of edges not belonging to $H_{wv}$, a vertex set $V_{H'}$. Specifically, $V_{H'}$ is the set of vertices belonging to $H'$ but not to $H_{wv}$. This is done as follows.

First, the procedure $P$ asks $O(n)$ queries to determine $d(v, \_)$. A vertex $u$ belongs to $H_{wv}$ iff $d(w, u) + d(u, v) \in \{d, d+1\}$. The actual placement of vertices in edges along $H_{wv}$ is determined in the same way as in the proof of Theorem 5, solely based on the responses to previous queries.

Now let $t \geq 2$ and consider all vertices $u$ for which $d(w, u) + d(u, v) = d + t$. These vertices are not in $H_{vw}$. If $t$ is even, such a vertex $u$ is in a sub-hypertree that ends in an edge adjacent to two

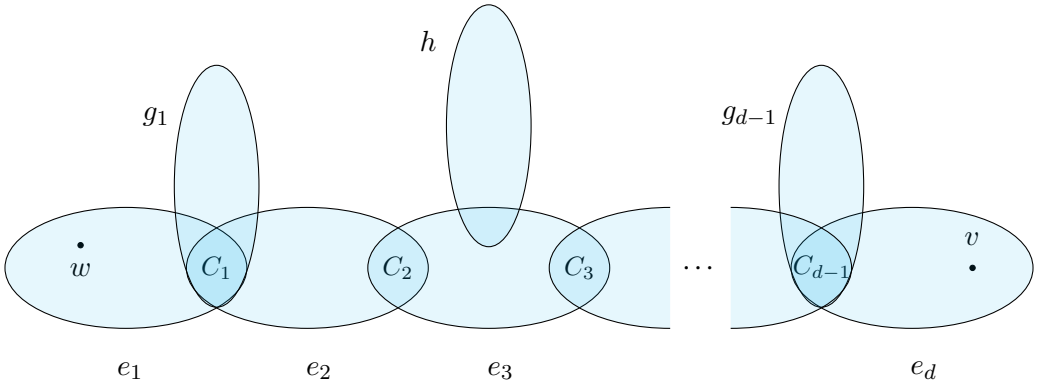


Figure 2: Learning a longest path $H_{wv}$ in a hypertree. Here $H_{wv}$ consists of edges $e_1$ through $e_d$. Edges $g_1$ and $g_{d-1}$ are part of sub-hypertrees of type 1, while $h$ is an edge of a sub-hypertree of type 2.

edges of $H_{wv}$. We call this a sub-hypertree of type 1. For illustration, in Fig. 2, such $u$ could lie, e.g., on a sub-hypertree containing $g_1$ or $g_{d-1}$, but not on a sub-hypertree containing $h$. If $t$ is odd, the vertex $u$ is in a sub-hypertree that ends in an edge adjacent to exactly one edge of $H_{wv}$. We call this a sub-hypertree of type 2. For illustration, in Fig. 2, such $u$ could lie, e.g., on a sub-hypertree containing $h$, but not on a sub-hypertree containing $g_1$ or $g_{d-1}$.

Now, $P$ has identified the vertex sets of all maximal sub-hypertrees in $H$ that are adjacent to $H_{wv}$, except for the intersections of these sub-hypertrees with $H_{wv}$.

Since the degree of each edge is at most 3, the distance values $d(w, u)$ and $d(u, v)$ tell us not only the type of sub-hypertree in which $u$ is located, but also to exactly which edge(s) on $H_{wv}$ this sub-hypertree is adjacent. Next, $P$ determines the intersection(s) of these sub-hypertrees with $H_{wv}$. In Fig. 2, this would mean to identify the intersections of (i) $g_1$, $e_1$, and $e_2$, i.e., $C_1$, (ii) $g_{d-1}$, $e_{d-1}$, and $e_d$, i.e., $C_{d-1}$, and (iii) $h$ and $e_3$.

The intersections of type 1 sub-hypertrees with $H_{wv}$ have already been identified when learning $H_{wv}$, since any non-empty intersection of three edges equals the intersection of two of these edges. In our case, two of these edges belong to $H_{wv}$ and their intersection is known already. In Fig. 2, the intersection of $g_1$, $e_1$, and $e_2$ equals the intersection of $e_1$ and $e_2$, and is known already.

Learning the intersections of type 2 sub-hypertrees with $H_{wv}$ requires additional SP queries. For each type 2 sub-hypertree $H'$, let $e_i$ be the edge on the hyperpath $H_{wv}$ with which the sub-hypertree $H'$ intersects. Further, let $V'$ be the set of vertices in $H'$, except those belonging to $H_{wv}$. (This set was identified before.) Now procedure $P$ picks *any* vertex $u \in V'$ that has the smallest distance to $w$ (and to $v$), i.e., $d(w, u) \leq d(w, u')$ for all $u' \in V'$. Clearly, $u$ belongs to the edge $e'$ of $H'$ that is adjacent to $e_i$. In Fig. 2, $e_i$ would be $e_3$, the sub-hypertree $H'$ would be attached to $e_3$ via the edge $e' = h$, and $u$ would be a member of $h$, but not of $e_3$. Procedure $P$ now asks SP queries of the form $(u, z)$ for each vertex $z \in e_i$. The vertices $z$ for which $d(u, z) = 1$ belong to the intersection of $e_i$ and $e'$; all other queried vertices $z$ fulfill $d(u, z) = 2$ and do not belong to this intersection. This step costs $O(n)$ queries in total.

Now, for each sub-hypertree $H'$ (of type 1 or 2), $P$ chooses a vertex $w'$ in $V'$ that has maximum distance from $w$—such $w'$ must have maximum eccentricity in $H'$. This $w'$ can now be used to compute the diameter $d'$ of $H'$ in $O(n)$ queries, cf. Lemma 10. Procedure $P$ then calls itself recursively with (i) the vertex set $V'$ of $H'$, (ii) the vertex $w'$ and its distance sequence, and (iii) $d'$.

This recursion has depth $O(d)$, because $d' < d$, so that the base case (diameter at most 2) will be reached after fewer than $d$ recursive calls. Since the number of SP queries made at each level of the recursion (across all sub-hypertrees at the same recursive depth) is in $O(n)$, the learner $L$ makes $O(dn)$ queries overall, while learning the target hypertree. ■

Next, we present an upper bound for learning $r$-uniform $c$-hypertrees of unknown diameter $d \geq 3$. By Lemma 10, for such hypergraphs, finding a vertex $w$ with $ecc(w) = d$ requires $O(n)$ SP queries. In an $r$-uniform hypertree, such vertex $w$ satisfies $d(w, \_) = 1^{r-1}, 2^{k_2(r-c)}, \ldots, d^{k_d(r-c)}$ for some positive integer values $k_2, \ldots, k_d$. The value $k_i$ is simply the number of edges whose non-shared portion lies at distance $i$ from $w$ in the target hypertree, see Appendix C.

**Theorem 12** *Let $\mathcal{H}$ be the class of all $r$-uniform $c$-hypertrees of order $n$ and diameter at least 3, where $r$ and $c$ are fixed, and $r > 2c$. Then*

$$m(\frac{5}{2}r - \frac{5}{2}c - 2) - d(r - c - 2) + 2c - 3 + \frac{r-c}{2}\sum_{i=1}^{d} k_i^2 + (r-c)\sum_{i=2}^{d} k_i k_{i-1}$$

*SP queries suffice to adaptively learn any target hypertree $H \in \mathcal{H}$. In this bound, $d$ denotes the diameter of $H$, $k_1 = 1$, and the $k_i$ for $i > 2$ are such that $d(w, \_) = 1^{r-1}, 2^{k_2(r-c)}, \ldots, d^{k_d(r-c)}$ for some vertex $w$ of maximum eccentricity in $H$.[2] This number of queries is in*

$$O(n + r\sum_{i=1}^{d} k_i^2 + r\sum_{i=2}^{d} k_i k_{i-1}) .$$

**Proof** Let $H \in \mathcal{H}$. We first fix some terminology. If $w$ is a vertex of maximum eccentricity in $H$, let $d(w, \_) = 1^{r-1}, 2^{k_2(r-c)}, \ldots, d^{k_d(r-c)}$. If $e$ is an edge in $H$ and $w \notin e$, then there is some $k > 1$ such that exactly $r - c$ vertices in $e$ have distance $k$ from $w$, while the remaining $c$ vertices have distance $k - 1$. The subset $\hat{e}$ of $r - c$ vertices at the larger distance is called the *edge thrust* of $e$ wrt $w$. The distance of an edge thrust $\hat{e}$ to $w$ is simply the distance of any vertex in $\hat{e}$ to $w$. If $e$ contains $w$, then all of $e$ forms its edge thrust; this is the only edge thrust of size $r$.

There exist $k_s$ edge thrusts at distance $s$ from $w$. We denote these by $\hat{e}_{s,i}$ for $i \in [k_s]$. For $s \in \{2, \ldots, d\}$ and $i \in [k_s]$, each edge thrust $\hat{e}_{s,i}$ is a subset of an edge that has $c$ common vertices with at least one edge thrust $\hat{e}_{s-1,i}$ where $i \in [k_{s-1}]$. To learn $H$, we proceed in three phases.

In phase 1, our learner determines a vertex $w$ of maximum eccentricity, as well as the values $d$ and $k_s$ for $2 \leq s \leq d$. By the proof of Lemma 10, this can be done with $O(n)$ SP queries.

In phase 2, the learner determines all edge thrusts $\hat{e}_{s,i}$ for $s \in \{2, \ldots, d\}$ and $i \in [k_s]$, as well as the edge including $w$. If $k_s = 1$, the learner can already determine the edge thrust $\hat{e}_{s,i}$ based on queries from phase 1; $\hat{e}_{s,i}$ consists of all vertices at distance $s$ from $w$. For $k_s > 1$, the learner chooses an arbitrary vertex $v_1$ at distance $s$ of $w$ and then asks SP queries for all pairs $(v_1, v_2)$ for

2. For each target hypertree $H$, the actual value of each individual $k_i$ depends on the choice of $w$, we assume a choice of $w$ that maximizes the stated bound on the number of queries.

all vertices $v_2$ at the same distance $s$ from $w$. Thus the learner uses $k_s(r-c)-2$ queries to identify one edge thrust $\hat{e}_{s,i}$ for some $i \in [k_s]$; in particular, this edge thrust contains the vertex $v_1$. The learner repeats this algorithm on the remaining vertices for all $i \in [k_s]$, to learn all edge thrusts $\hat{e}_{s,i}$ for $i \in [k_s]$. This requires no more than

$$\sum_{i=2}^{k_s}(i(r-c)-2) = (r-c)\sum_{i=2}^{k_s} i - 2(k_s-1) = (r-c)\frac{(k_s+2)(k_s-1)}{2} - 2(k_s-1)$$

SP queries. Let $A = \{i \in [d] \,|\, k_i \geq 2\}$. Our learner asks all these queries for each value of $k_s$, for a total of $\sum_{s\in A}\sum_{i=2}^{k_s}(i(r-c)-2)$ SP queries. As $(r-c)\frac{(k_s+2)(k_s-1)}{2} - 2(k_s-1) = 0$ when $k_s = 1$, the number of queries used in phase 2 is

$$\sum_{s\in A}\sum_{i=2}^{k_s}(i(r-c)-2) = \sum_{s=1}^{d}\sum_{i=2}^{k_s}(i(r-c)-2) = \frac{r-c}{2}\sum_{s=1}^{d}(k_s+2)(k_s-1) - 2\sum_{s=1}^{d}(k_s-1)$$
$$= \frac{r-c}{2}\sum_{s=1}^{d} k_s^2 + \frac{m}{2}(r-c-4) - d(r-c-2)\,.$$

In phase 3, our learner determines which edges overlap, and which vertices lie in their intersection. To learn all vertices in the intersections of edges in $H$, let $u$ be a vertex in an edge thrust $\hat{e}_{d,i}$ where $i \in [k_d]$. The learner asks $k_{d-1}(r-c)$ SP queries to obtain $d(u,z)$ for all vertices $z$ in edge thrusts $\hat{e}_{d-1,i}$ for $i \in [k_{d-1}]$. From queries resulting in the answer 1, the learner learns the intersection of the edge that includes $\hat{e}_{d,i}$ with the edge thrust $\hat{e}_{d-1,j}$, for some $j \in [k_{d-1}]$.

The learner repeats this procedure for one vertex each per edge thrust $\hat{e}_{d,i}$ for $i \in [k_d]$, to determine distances to all vertices in $\hat{e}_{d-1,i}$ for $i \in [k_{d-1}]$. Thus, it uses $(r-c)\,k_d\,k_{d-1}$ SP queries to determine intersections of edges including edge thrusts $\hat{e}_{d,i}$ and $\hat{e}_{d-1,j}$ for $i \in [k_d]$ and $j \in [k_{d-1}]$. In total, to learn all edge intersections in $H$, it suffices to ask $(r-c)\sum_{i=2}^{d} k_i k_{i-1}$ SP queries.

Applying Lemma 10, and using the fact that $n = m(r-c)+c$, the total number of SP queries asked by our learner is thus at most

$$2n - 3 + \frac{r-c}{2}\sum_{i=1}^{d} k_i^2 + \frac{m}{2}(r-c-4) - d(r-c-2) + (r-c)\sum_{i=2}^{d} k_i k_{i-1}$$
$$= m(\frac{5}{2}r - \frac{5}{2}c - 2) - d(r-c-2) + 2c - 3 + \frac{r-c}{2}\sum_{i=1}^{d} k_i^2 + (r-c)\sum_{i=2}^{d} k_i k_{i-1}\,.$$

■

**Remark 13** *Let $b = \max\{k_i \,|\, i \in [d]\}$. We have $\sum_{i=1}^{d} k_i = m$ and*

$$r\sum_{i=1}^{d} k_i^2 + r\sum_{i=2}^{d} k_i k_{i-1} \leq br\sum_{i=1}^{d} k_i + rb\sum_{i=2}^{d} k_i < brm + rbm = 2brm.$$

*Theorem 12 then implies that the SP query complexity of learning $r$-uniform $c$-hypertrees is in $O(rbm) = O(bn)$. Hence, if $b$ is a constant (as, e.g., in the special case of learning $r$-uniform $c$-hyperpaths), then the upper bound is linear in the number of vertices.*

## 6. Conclusions

Hypertrees are objects of interest for many applications. This paper analyzed their learnability from SP queries, which are a natural form of query previously studied in the context of learning graphs, but not in the context of hypergraphs. There are multiple insights to be gained from our algorithms and bounds.

For example, there is a strong connection between learning hyperstars and learning graph partitions, which was studied by Liu and Mukherjee (2022). Since the hyperstars we consider are just finite set systems in which the intersection of any two sets equals the intersection of all sets, this connection is certainly not surprising, but it still turned out useful for our subsequent analysis.

Of potential value are also our algorithms for learning binary hypertrees and for learning $r$-uniform $c$-hypertrees. Firstly, the algorithm for binary hypertrees can possibly be extended to handle more complex structures, for instance hypertrees of larger degree, where any edge along the path $H_{wv}$ can host both a sub-hypertree of type 1 and a sub-hypertree of type 2. A more detailed analysis of this type of structure is left for future work. Our algorithm for learning uniform hypertrees nicely reveals the effect of the parameters $k_i$, which are related to the number of edges at a certain distance away from a vertex of maximum eccentricity. Insights into the effect of these parameters may be helpful in the design of learning algorithms for more complex classes of hypergraphs, possibly also using different kinds of queries.

In this vein, we hope that our work lays the foundation for future studies on learning interesting classes of hypergraphs from various types of queries.

## Acknowledgments

The authors would like to thank the anonymous referees for their detailed comments, as well as Kamyar Khodamoradi for helpful discussions. Shaun Fallat was supported in part by NSERC Discovery Research Grant, Application No.: RGPIN–2019–03934. Sandra Zilles was supported in part by NSERC Discovery Research Grant, Application No.: RGPIN–2017–05336, by an NSERC Canada Research Chair, and by a Canada CIFAR AI Chair held at the Alberta Machine Intelligence Institute (Amii).

## References

Hasan Abasi. Error-tolerant non-adaptive learning of a hidden hypergraph. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 3:1–3:15, 2018.

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.

Noga Alon, Richard Beigel, Simon Kasif, Steven Rudich, and Benny Sudakov. Learning a hidden matching. *SIAM Journal on Computing*, 33(2):487–501, 2004.

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 Annual Conference on Learning Theory (COLT)*, pages 419–420, 2022.

Claude Berge. *Graphs and Hypergraphs*. North-Holland Mathematical Library, 1976.

Alain Bretto. *Hypergraph Theory – An Introduction*. Springer, 2013.

Thomas Erlebach, Alexander Hall, Michael Hoffmann, and Matús Mihalák. Network discovery and verification with distance queries. In *Proceedings of the 6th Italian Conference on Algorithms and Complexity (CIAC)*, volume 3998 of *Lecture Notes in Computer Science*, pages 69–80. Springer, 2006.

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)*, volume 9134 of *Lecture Notes in Computer Science*, pages 773–784. Springer, 2015.

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

Xizhi Liu and Sayan Mukherjee. Tight query complexity bounds for learning graph partitions. In *Proceedings of the 35th Annual Conference on Learning Theory (COLT)*, volume 178, pages 167–181, 2022.

Claire Mathieu and Hang Zhou. Graph reconstruction via distance oracles. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 7965 of *Lecture Notes in Computer Science*, pages 733–744. Springer, 2013.

Kevin A. Murgas, Emil Saucan, and Romeil Sandhu. Hypergraph geometry reflects higher-order dynamics in protein interaction networks. *Scientific Reports*, 12:20879, 2022.

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)*, volume 4754 of *Lecture Notes in Computer Science*, pages 285–297. Springer, 2007.

Table 1: Table of Notation

| | |
|---|---|
| SP | shortest path |
| $d$ | diameter |
| $n$ | number of vertices |
| $m$ | number of edges |
| $r$ | number of vertices per edge in a uniform hypergraph |
| $c$ | number of common vertices between two overlapping edges |
| $d(v, w)$ | distance between vertices $v$ and $w$ |
| $d(v, \_)$ | distance sequence of vertex $v$ (sequence of distances between $v$ and all $n-1$ other vertices) |
| $e_i$ | the edge with label $i$ in a hypergraph |

Zhihui Wang, Jianrui Chen, Fernando E. Rosas, and Tingting Zhu. A hypergraph-based framework for personalized recommendations via user preference and dynamics clustering. *Expert Systems with Applications*, 204:117552, 2022.

## Appendix A. Overview of Notation

Table 1 gives an overview of the notation used throughout the paper.

## Appendix B. Proof of Theorem 6

**Theorem 6** $\Omega(n^2 - 2n(r-c))$ *non-adaptive SP queries are needed to learn an* $r$*-uniform* $c$*-hyperstar of order* $n$*, where* $r \geq 3$*.*

**Proof** Let $S$ be a $r$-uniform $c$-hyperstar of order $n$, where $r \geq 3$. If the query graph constructed by the learner contains two non-adjacent vertices $u$ and $v$ whose joint neighborhood is of size at most $n - 2r + 2c$, then the oracle can chose two hyperstars in which $u$ is in one edge and and $v$ is in another. Let $V_1$ be the set of vertices that the learner identifies in the joint neighborhood of $u$ and $v$. The size of $V_1$ is at most $n - 2r + 2c$ and contains the vertices in the shared section of $S$. In this case the oracle places all vertices in $V_1$ in edges from the third to the final edge.

Hence the oracle has sufficient freedom to place the vertices in $V_1$ as desired, and therefore can freely swap $u$ and $v$ in the first and second edges. Now, the learner will be unable to decide which of $u$ and $v$ to place in the first and second edges. Thus the learner is not able to fully identify the hidden hypergraph as required.

Therefore, a successful set of queries must yield a query graph $G$ in which any two vertices $u$ and $v$ are adjacent or have a joint neighborhood of size at least $n - 2r + 2c + 1$. Hence the query graph $G$ has at least $\Omega(n^2 - 2n(r-c))$ edges.

Suppose that $G$ is the query graph induced by $m(G)$ non-adaptive SP queries. Observe that the minimum number of edges for such a query graph gives a lower bound for the minimum number of SP queries.

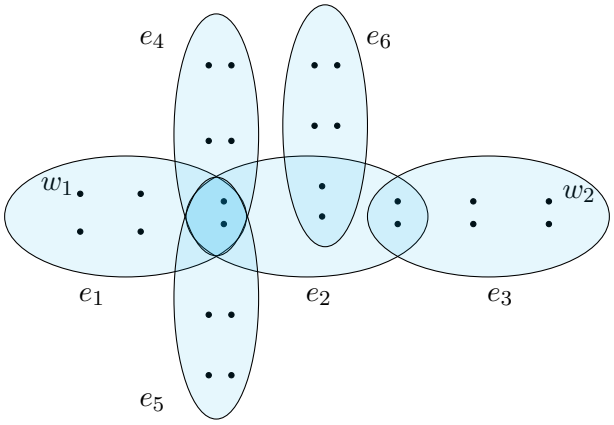


Figure 3: A 6-uniform 2-hypertree $H$

If every vertex has a degree at least $\frac{n-2r+2c}{2}$, then

$$2m(G) = \sum_{i=1}^{n} d_i(G) \geq \frac{n\,(n-2r+2c)}{2}, \quad \text{which implies}$$

$$m(G) \geq \frac{n^2 - 2nr + 2nc}{4}. \tag{1}$$

Otherwise, there exists a vertex $v$ whose degree is less than $\frac{n-2r+2c}{2}$. Then by our conclusion above, the degree of each vertex that is not adjacent to $v$ is strictly greater than $\frac{n-2r+2c}{2}$. There are at least $n - \frac{n-2r+2c}{2}$ such vertices that are not adjacent to $v$ , so

$$2m(G) \geq \frac{n-2r+2c}{2}\,\big(n - \frac{n-2r+2c}{2}\big),$$

that is,

$$m(G) \geq \frac{n^2 - (2r-2c)^2}{8}. \tag{2}$$

Inequalities (1) and (2) imply that the learner must ask $\Omega(n^2 - 2n(r-c))$ non-adaptive SP queries to fully identify $S$. ■

## Appendix C. Illustration of Parameters $k_i$ for $r$-Uniform $c$-Hypertrees, Theorem 12

Figure 3 shows an example of a 6-uniform 2-hypertree $H$. The diameter of $H$ is $d = 3$, and the two vertices $w_1$ and $w_2$ are of (maximum) eccentricity 3. Using the notation from Theorem 12, for $w_1$ we have $d(w_1, \_) = 1^{r-1},\ 2^{k_2(r-c)},\ \ldots, d^{k_d(r-c)} = 1^5, 2^{12}, 3^8$, which yields $k_2 = 3,\ k_3 = 2$. For $w_2$, we have $d(w_2, \_) = 1^{r-1},\ 2^{k_2(r-c)},\ \ldots, d^{k_d(r-c)} = 1^5, 2^4, 3^{16}$, which yields $k_2 = 1,\ k_3 = 4$.