Journal of Machine Learning Research 7 (2006) 2215-2236 Submitted 12/05; Revised 8/06; Published 10/06

Learning a Hidden Hypergraph

Dana Angluin ANGLUIN@CS.YALE.EDU
Department of Computer Science
Yale University
P.O. Box 208285
New Haven, CT 06520, USA

Jiang Chen CRIVER@CS.COLUMBIA.EDU
Center for Computational Learning Systems
Columbia University
475 Riverside Drive
850 Interchurch MC 7717
New York, NY 10115, USA

Editor: Manfred Warmuth

Abstract

We consider the problem of learning a hypergraph using edge-detecting queries. In this model, the learner may query whether a set of vertices induces an edge of the hidden hypergraph or not. We show that an r-uniform hypergraph with m edges and n vertices is learnable with $O(2^{4r}m \cdot poly(r, \log n))$ queries with high probability. The queries can be made in $O(\min(2^r(\log m + r)^2, (\log m + r)^3))$ rounds. We also give an algorithm that learns an almost uniform hypergraph of dimension r using $O(2^{O((1+\frac{\Delta}{2})r)} \cdot m^{1+\frac{\Delta}{2}} \cdot poly(\log n))$ queries with high probability, where Δ is the difference between the maximum and the minimum edge sizes. This upper bound matches our lower bound of $\Omega((\frac{m}{1+\frac{\Delta}{2}})^{1+\frac{\Delta}{2}})$ for this class of hypergraphs in terms of dependence on m. The queries can also be made in $O((1+\Delta) \cdot \min(2^r(\log m + r)^2, (\log m + r)^3))$ rounds.

Keywords: query learning, hypergraph, multiple round algorithm, sampling, chemical reaction network

1. Introduction

A hypergraph $H = (V, E)$ is given by a set of vertices V and a set of edges E, which is a subset of the power set of V ($E \subseteq 2^V$). The dimension of a hypergraph H is the cardinality of the largest set in E. H is said to be r-uniform if E contains only sets of size r. In this paper, we are interested in learning a hidden hypergraph using *edge-detecting* queries of the following form

$$Q_H(S) : \textit{does } S \textit{ include at least one edge of } H\textit{?}$$

where $S \subseteq V$. The query $Q_H(S)$ is answered 1 or 0, indicating whether S contains all vertices of at least one edge of H or not. We abbreviate $Q_H(S)$ to $Q(S)$ whenever the choice of H is clear from the context. This type of query may be motivated by the following scenarios. We are given a set of chemicals, in which some groups of chemicals react and others don't. When multiple chemicals are combined in one test tube, a reaction is detectable if and only if at least one group of chemicals in the tube react.

Considerable effort, for example, Grebinski and Kucherov (1998), Beigel et al. (2001), Alon et al. (2004), Angluin and Chen (2004), and Alon and Asodi (2005), has been devoted to the case when the underlying reaction network is a graph, that is, chemicals react in pairs. Among them, Grebinski and Kucherov (1998), Beigel et al. (2001) and Alon et al. (2004) study the case when the underlying networks are Hamiltonian cycles or matchings, which have specific applications to genome sequencing. In this application, DNA sequences are aligned according to the reactions that involve the two ends of pairs of DNA sequences in certain experimental settings. The reaction graph can be characterized as either a Hamiltonian cycle or path (if you consider each DNA sequence as a vertex) or a matching (if you consider each end of a DNA sequence as a vertex). Implementations of some of these algorithms are in practical use. Grebinski and Kucherov (2000) also study a somewhat different and interesting query model, which they call the *additive model*, where instead of giving a 1 or 0 answer, a query tells you the total number of edges contained in a certain vertex set.

Angluin and Chen (2004) generalize the problem of learning with edge-detecting queries to general reaction graphs and show that general graphs are efficiently learnable. In this work, we consider a more general problem when the chemicals react in groups of size more than two, that is, the underlying reaction network is a hypergraph. In Angluin and Chen (2004), they give an adaptive algorithm which takes $O(\log n)$ queries per edge, where n is the number of vertices. This is nearly optimal as we can easily show using an information-theoretic argument. For the problem of learning hypergraphs of bounded dimension and a given number of edges, a similar information-theoretic argument gives a lower bound that is linear in the number of edges. However, the lower bound is not achievable. It is shown in Angluin and Chen (2004) that $\Omega((2m/r)^{r/2})$ edge-detecting queries are required to learn a general hypergraph of dimension r with m edges. In the heart of the construction of Angluin and Chen (2004), edges of size 2 are deliberately arranged to hide an edge of size r. The discrepancy in sizes of different coexisting edges is the main barrier for the learner. However, this lower bound does not preclude efficient algorithms for classes of hypergraphs whose edges sizes are close. In particular, the question whether there is a learning algorithm for uniform hypergraphs using a number of queries that is linear in the number of edges is still left open, which is the main subject of this paper.

In this paper, we are able to answer this question affirmatively. Let n be the number of vertices and m be the number of edges in the hypergraph. We show that an r-uniform hypergraph is learnable with $O(2^{4r}m \cdot poly(r, \log n, \log\frac{1}{\delta}))$ queries with probability at least $1-\delta$.

We also obtain results for learning the class of hypergraphs that is almost uniform. Formally speaking,

Definition 1 *A hypergraph is (r,Δ)-uniform, where $\Delta < r$, if its dimension is r and the difference between its maximum and minimum edge sizes is Δ, or equivalently, the maximum and the minimum edge sizes are r and $r-\Delta$ respectively.*

The class of hypergraphs used in the construction of the lower bound in Angluin and Chen (2004) is in fact $(r, r-2)$-uniform. Therefore, they show that $\Omega((2m/r)^{r/2})$ edge-detecting queries are required to learn a $(r, r-2)$-uniform hypergraph. Based on this result, we show by a simple reduction that $\Omega((\frac{m}{1+\frac{\Delta}{2}})^{1+\frac{\Delta}{2}})$ queries are required to learn the class of (r,Δ)-uniform hypergraphs. On the other hand, we extend the algorithm that learns uniform hypergraphs to learning the class of

(r,Δ)-uniform hypergraphs with $O(2^{O((1+\frac{\Delta}{2})r)} \cdot m^{1+\frac{\Delta}{2}} \cdot poly(\log n, \log \frac{1}{\delta}))$ queries with probability at least $1-\delta$. The upper bound and lower bound have the same dependence on m.

Another important issue studied in the literature is the parallelism of algorithms. Since the queries are motivated by an experiment design scenario, it is desirable that experiments can be conducted in parallel. Alon et al. (2004) and Alon and Asodi (2005) give lower and upper bounds for 1-round algorithms for certain types of graphs. Beigel et al. (2001) describe an 8-round algorithm for learning a matching. Angluin and Chen (2004) give a 5-round algorithm for learning a general graph. In this paper, we show that in our algorithm for r-uniform hypergraphs, queries can be made in $O(\min(2^r(\log m + r)^2, (\log m + r)^3))$ rounds, and in our algorithm for (r,Δ)-uniform hypergraphs, queries can be made in $O((1+\Delta)\cdot \min(2^r(\log m + r)^2, (\log m + r)^3))$ rounds.

In the paper, we also introduce an interesting combinatorial object, which we call an *independent covering family*. Basically, an independent covering family of a hypergraph is a collection of independent sets that cover all non-edges. An interesting observation is that the set of negative queries of any algorithm that learns a hypergraph drawn from a class of hypergraphs that is closed under the operation of adding an edge is an independent covering family of that hypergraph. Note both the class of r-uniform hypergraphs and the class of (r,Δ)-uniform hypergraphs are closed under the operation of adding an edge. This implies that the query complexity of learning such a hypergraph is bounded below by the minimum size of its independent covering families. In the opposite direction, we give subroutines to find one arbitrary edge from a hypergraph. With the help of the subroutines, we show that if we can construct small-sized independent covering families for some class of hypergraphs, we are able to obtain an efficient learning algorithm for it. In this paper, we give a randomized construction of an independent covering family of size $O(r2^{2r}m\log n)$ for r-uniform hypergraphs with m edges. This yields a learning algorithm using a number of queries that is quadratic in m, which is further improved to give an algorithm using a number of queries that is linear in m.

As mentioned in Angluin and Chen (2004) and some other papers, the hypergraph learning problem may also be viewed as the problem of learning a monotone disjunctive normal form (DNF) boolean formula using membership queries only. Each vertex of H is represented by a variable and each edge by a term containing all variables associated with the vertices of the edge. A membership query assigns 1 or 0 to each variable, and is answered 1 if the assignment satisfies at least one term, and 0 otherwise, that is, if the set of vertices corresponding to the variables assigned 1 contains all vertices of at least one edge of H. An r-uniform hypergraph corresponds to a monotone r-DNF. An (r,Δ)-uniform hypergraph corresponds to a monotone DNF whose terms are of sizes in the range of $[r-\Delta, r]$. Thus, our results apply also to learning the corresponding classes of monotone DNF formulas using membership queries.

The paper is organized as follows. In Section 3, we formally define the concept of an independent covering family and give a randomized construction of independent covering families for general r-uniform hypergraphs. In Section 4, we show how to efficiently find an arbitrary edge in a hypergraph and give a simple learning algorithm using a number of queries that is quadratic in the number of edges. In Section 5, we give an algorithm that learns r-uniform hypergraphs using a number of queries that is linear in the number of edges. Then we derive a lower bound for almost uniform hypergraphs in Section 6. Finally, we show how to learn the class of (r,Δ)-uniform hypergraphs in Section 7.

2. Preliminaries

Let $H = (V, E)$ be a hypergraph. In this paper, we assume that edges do not contain each other, as there is no way to detect the existence of edges that contain other edges using edge-detecting queries. A subset of V is an *independent set* of H if it contains no edge of H. We use the term *non-edge* to denote any set that is a candidate edge in some class of hypergraphs but is not an edge in the target hypergraph. For example, in an r-uniform hypergraph, any r-set that is not an edge is a non-edge. In an (r,Δ)-uniform hypergraph, any set of size in the range of $[r-\Delta, r]$ that is not an edge is a non-edge. The *degree* of a set $\chi \subseteq V$ in a hypergraph H denoted as $d_H(\chi)$ is the number of edges of H that contain χ. In particular, $d_H(\emptyset) = |E|$ is the number of all edges in H.

Throughout the paper, we omit the ceiling and floor signs whenever they are not crucial.

3. An Independent Covering Family

Definition 2 An independent covering family *of a hypergraph H is a collection of independent sets of H such that every non-edge not containing an edge is contained in one of these independent sets.*

When H is a uniform hypergraph, the above only requires that every non-edge is contained in one of the independent sets in the independent covering family. An example is shown below.

Example 1 *Let $V = [1,6]$. Let $H = (V, \{\{1,2,3\},\{4,5,6\},\{2,4,5\}\})$ be a 3-uniform hypergraph.*

$$\mathcal{F} = \{\{1,2,4,6\},\{1,2,5,6\},\{1,3,4,5\},\{1,3,4,6\},\{2,3,4,6\},\{2,3,5,6\}\}$$

is an independent covering family of H. As we can easily verify, all sets in $\mathcal{F}$ are independent sets, and every triple except $\{1,2,3\},\{4,5,6\},\{2,4,5\}$ is contained in some set in $\mathcal{F}$.

The concept of independent covering families is central in this paper. This can be appreciated from two aspects.

First, we observe that if the target hypergraph is drawn from a class of hypergraphs that is closed under the operation of adding an edge (e.g., the class of all r-uniform hypergraphs), the set of negative queries of any algorithm that learns it is an independent covering family of this hypergraph. This is because if there is a non-edge not contained in any of the sets on which these negative queries are made, we will not be able to distinguish between the target hypergraph and the hypergraph with this non-edge being an extra edge. Therefore, the minimum size of independent covering families bounds the query complexity from below. Furthermore, any learning algorithm gives a construction of an independent covering family of the target hypergraph. Therefore, in order to learn the hypergraph, we have to be able to construct an independent covering family for it.

Second, although the task of constructing an independent covering family seems substantially easier than that of learning, since the hypergraph is known in the construction task, we show that efficient construction of small-sized independent covering families yields an efficient learning algorithm. In Section 4, we will show how to find an arbitrary edge out of a hypergraph of dimension r using $O(r \log n)$ queries. Imagine a simple algorithm in which at each iteration we maintain a sub-hypergraph of the target hypergraph which contains edges that we have found, and construct an independent covering family for it and ask queries on all the sets in the family. If there is a set whose query is answered positively, we can find at least one edge out of this set. The edge must be a new edge as the set is an independent set of the sub-hypergraph that we have found. We repeat this process until we have collected all the edges in the target hypergraph, in which case the independent covering family we construct is a proof of this fact. Suppose that we can construct an independent covering family of size at most $f(m)$ for any hypergraph with at most m edges drawn from certain class of hypergraphs. The above algorithm learns this class of hypergraphs using only $O(f(m) \cdot m \cdot r \log n)$ queries.

In the rest of this section, we give a randomized construction of a linear-sized (linear in the number of edges) independent covering family of an r-uniform hypergraph which succeeds with probability at least 1/2. By the standard probabilistic argument, the construction proves the existence of an independent covering family of size linear in the number of edges for any uniform hypergraph. This construction leads to a quadratic algorithm described in Section 4, and is also a central part of our main algorithm given in Section 5.

Our main theorem in this section is as follows.

Theorem 3 *Any r-uniform hypergraph with m edges has an independent covering family of size $O(r2^{2r} m \log n)$.*

Before giving the construction, we introduce some notation and definitions. We call a vertex set $\chi \subseteq V$ *relevant* if it is contained in at least one edge in the hypergraph. Similarly, a vertex is *relevant* if it is contained in at least one edge in the hypergraph. Let $p_H(\chi) = 1/(2^{r+1} d_H(\chi))^{1/(r-|\chi|)}$, where χ is a relevant vertex set. We will call $p_H(\chi)$ the *discovery probability* of χ, as this is a probability that will help in discovering edges containing χ in our learning algorithms.

Definition 4 *A* (χ, p)-sample *is a random set of vertices that contains χ and contains each other vertex independently with probability p.*

We will abbreviate (χ, p)-sample as χ-sample when the choice of p is clear or not important in the context.

In the construction, we draw $(\chi, p_H(\chi))$-samples independently for each relevant set χ. Each $(\chi, p_H(\chi))$-sample deals only with non-edges that contain χ. Let us take a look at the probability that a $(\chi, p_H(\chi))$-sample P_χ covers some non-edge $z \supseteq \chi$ while excluding all edges. Due to our choice of $p_H(\chi)$,

$$Pr[z \subseteq P_\chi] = p_H(\chi)^{r-|\chi|} = \frac{1}{2^{r+1} d_H(\chi)}.$$

Therefore, if we draw $2^{r+1} d_H(\chi)$ many χ-samples, the probability that z is contained in at least one χ-sample is $\Omega(1)$. However, such a χ-sample is not necessarily an independent set. Especially when z contains a high degree subset χ', it is likely that such a χ-sample contains an edges that contains χ'. But since we will also draw $(\chi', p_H(\chi'))$-samples, it is reasonable to hope that a $(\chi', p_H(\chi'))$-sample has better chance of success in dealing with z. In fact, in our construction, we show that the set of χ-samples, where $\chi \subseteq z$ has the minimum discovery probability among all relevant subsets of z, has an independent set that contains z with probability at least $1/2$.

A construction of an independent covering family is given in Algorithm 1, which succeeds with probability at least 1/2 as shown by Lemma 5.

Lemma 5 *$\mathcal{F}_H$ (constructed in Algorithm 1) contains an independent covering family of H with probability at least 1/2.*

Algorithm 1 Construction of an independent covering family

```
1: $\mathcal{F}_H \leftarrow$ a set containing $4(\ln 2 + r\ln n)\cdot 2^r d_H(\chi)$ $(\chi, p_H(\chi))$-samples drawn independently for every relevant set $\chi$.
2: Output the independent sets contained in $\mathcal{F}_H$ as an independent covering family.
```

Proof Suppose z is a non-edge and χ is a subset of z with the minimum discovery probability. Let P_χ be a χ-sample. As argued before,

$$Pr[z \subseteq P_\chi] = \frac{1}{2^{r+1} d_H(\chi)}.$$

Since χ has the minimum discovery probability, the degree of any subset $\chi' \subseteq z$ is at most $1/(2^{r+1} p_H(\chi)^{r-|\chi'|})$. By the union bound,

$$\begin{aligned} Pr[P_\chi \text{ is independent} | z \subseteq P_\chi] &\geq 1 - \sum_{\chi' \subseteq z} d_H(\chi') p_H(\chi)^{r-|\chi'|} \\ &\geq 1 - \sum_{\chi' \subseteq z} \frac{1}{2^{r+1} p_H(\chi)^{r-|\chi'|}} p_H(\chi)^{r-|\chi'|} \\ &= 1/2. \end{aligned}$$

The probability that a χ-sample contains z and is independent is at least $1/(2^{r+2} d_H(\chi))$. Therefore, the probability that such a χ-sample exists in $\mathcal{F}_H$ is at least

$$\begin{aligned} & 1 - (1 - \frac{1}{2^{r+2} d_H(\chi)})^{4(\ln 2 + r\ln n)\cdot 2^r d_H(\chi)} \\ \geq & 1 - e^{-(r\ln n + \ln 2)} \\ = & 1 - \frac{1}{2n^{-r}}. \end{aligned}$$

Thus, the probability that every non-edge is contained in some negative sample in $\mathcal{F}_H$ is at least $1 - \binom{n}{r}/(2n^r) \geq 1/2$. ■

Theorem 3 is then established by the fact that the size of $\mathcal{F}_H$ is bounded by $\sum_\chi 4(\ln 2 + r\ln n)\cdot 2^r d_H(\chi) = O(r 2^{2r} m \log n)$.

4. A Simple Quadratic Algorithm

In this section, we first give an algorithm that finds an arbitrary edge in a hypergraph of dimension r using only $r\log n$ edge-detecting queries. The algorithm is adaptive and takes $r\log n$ rounds. The success probability in the construction of independent covering families in the previous section can be easily improved by drawing more samples. Using the high-probability version of the construction, we obtain an algorithm using a number of queries that is quadratic in m that learns an r-uniform hypergraph with m edges with high probability. Although the first algorithm for finding one edge is deterministic and simple, the round complexity $r\log n$ might be too high when n is much larger than m. We then improve the round complexity to $O(\log m + r)$ using only $O(\log m \log n)$ more queries. The improved algorithm is randomized and succeeds with high probability.

4.1 Find One Edge

We start with a simpler task, finding just one relevant vertex in the hypergraph. The algorithm FIND-ONE-VERTEX is shown in Algorithm 2.

Algorithm 2 FIND-ONE-VERTEX

```
1: $S \leftarrow V, A \leftarrow V$.
2: while $|A| > 1$ do
3:     Divide $A$ arbitrarily into $A_0$ and $A_1$, such that $|A_0| = \lceil |A|/2 \rceil$, $|A_1| = \lfloor |A|/2 \rfloor$.
4:     if $Q(S \backslash A_0) = 0$ then
5:         $A \leftarrow A_0$.
6:     else
7:         $A \leftarrow A_1$, $S \leftarrow S \backslash A_0$.
8:     end if
9: end while
10: Output the element in $A$.
```

Lemma 6 *FIND-ONE-VERTEX finds one relevant vertex in a non-empty hypergraph with n vertices using at most* $\log n$ *edge-detecting queries.*

Proof First we show that the following equalities hold for each iteration (see Figure 1).

$$Q(S) = 1, Q(S \backslash A) = 0.$$

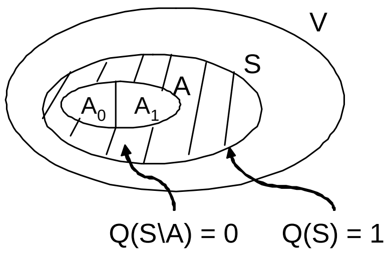

Figure 1: An illustration of FIND-ONE-VERTEX

These equalities guarantee that A contains at least one relevant vertex. Since we assume that the hypergraph is non-empty, the above equalities clearly hold for our initial assignment of S and A. Let's assume $Q(S) = 1$ and $Q(S \backslash A) = 0$ at the beginning of an iteration. There are two cases:

1. $Q(S \backslash A_0) = 0$, clearly the equalities hold for S and A_0.

2. $Q(S \backslash A_0) = 1$, since $Q((S \backslash A_0) \backslash A_1) = Q(S \backslash (A_0 \cup A_1)) = Q(S \backslash A) = 0$, the equalities hold for $S \backslash A_0$ and A_1.

Since the size of A halves at each iteration, after at most $\log n$ iterations, A has exactly one relevant vertex. The algorithm takes at most $\log n$ edge-detecting queries in total, as it makes one query in each iteration. ■

Using FIND-ONE-VERTEX as a subroutine, FIND-ONE-EDGE (Algorithm 3) is a recursive algorithm that finds one edge from a non-empty hypergraph, which is not necessarily uniform. Note knowledge of r is not required in FIND-ONE-EDGE. It is included in the description of the algorithm for the purpose of explanation.

Algorithm 3 FIND-ONE-EDGE

```
 1: Let $r > 0$ be the dimension of the hypergraph.
 2: Call FIND-ONE-VERTEX and let $v$ be the found vertex.
 3: Make a query on $\{v\}$.
 4: **if** the query is answered 1 **then**
 5:    Output $\{v\}$.
 6: **else**
 7:    FIND-ONE-VERTEX also computes a set $S$ such that $Q(S) = 1$ and $Q(S \backslash \{v\}) = 0$. That is,
       $S$ contains only edges incident with $v$.
 8:    Call FIND-ONE-EDGE on the hypergraph induced on $S$ with the vertex $v$ removed. The
       hypergraph is of dimension at most $r - 1$. Let $e$ be the found edge.
 9:    Output the edge $e \cup \{v\}$.
10: **end if**
```

Edge-detecting queries for recursive calls of FIND-ONE-EDGE can be simulated recursively. To make an edge-detecting query for a next-level recursive call of FIND-ONE-EDGE, we just need to make an edge-detecting query at the current level on the union of a subset of S and $\{v\}$. In fact, each time, we make edge-detecting queries on the union of a subset of S and the set of vertices already found.

In FIND-ONE-EDGE, because S contains only edges incident with v, $e \cup \{v\}$ is an edge in the hypergraph. This establishes its correctness. The following lemma shows that it uses only $r \log n$ queries.

Lemma 7 *FIND-ONE-EDGE finds one edge in a non-empty hypergraph of dimension r with n vertices using $r \log n$ edge-detecting queries.*

Proof When $r = 1$, the problem is exactly that of finding one relevant vertex and hence solvable using $\log n$ queries. It is evident that if FIND-ONE-EDGE uses $(r-1)\log n$ queries for a hypergraph with dimension $r - 1$. then it only uses $(r-1)\log n + \log n = r \log n$ queries for a hypergraph with dimension r. ■

4.2 A Quadratic Algorithm

With the help of FIND-ONE-EDGE, we give the first learning algorithm for r-uniform hypergraphs. A sketch of the algorithm has been described in Section 3. Let $H = (V, E)$ be the sub-hypergraph the algorithm has found so far. Algorithm 4 learns a uniform hypergraph with probability at least $1 - \delta$. We will specify δ' later.

In the algorithm we draw $4(\ln(\frac{1}{\delta'}) + r \ln n) \cdot 2^r d_H(\chi)$ χ-samples. Using essentially the same argument as in Section 3, we can guarantee that $\mathcal{F}_H$ contains an independent covering family with

Algorithm 4 The quadratic algorithm

```
1: $e \leftarrow$ FIND-ONE-EDGE(V). $E \leftarrow \{e\}$.
2: repeat
3:     $\mathcal{F}_H \leftarrow 4(\ln\frac{1}{\delta'} + r\ln n) \cdot 2^r d_H(\chi)$ $(\chi, p_H(\chi))$-samples drawn independently for every relevant
       set $\chi$ in $H$.
4:     Make queries on sets of $\mathcal{F}_H$ that are independent in $H$.
5:     Call FIND-ONE-EDGE on one positive sample if there exists any. Let $e$ be the edge found.
       $E \leftarrow E \cup \{e\}$.
6: until no new edge found
```

probability at least $1-\delta'$. Algorithm 4 finds one new edge at each iteration because $\mathcal{F}_H$ is an independent covering family of the already found sub-hypergraph H. Thus, it ends after at most m iterations. If we we choose $\delta' = \delta/m$, it will succeed with probability at least $1-\delta$. As knowledge of m is not assumed, we will choose $\delta' = \delta/\binom{n}{r} \leq \delta/m$. The query complexity will be $O(2^{2r}m^2 \cdot poly(r, \log n) \cdot \log\frac{1}{\delta})$, which is quadratic in m.

4.3 An Improved FIND-ONE-EDGE

Despite the simplicity of FIND-ONE-EDGE, its queries have to be made in $r\log n$ rounds. When irrelevant vertices abound, that is, when n is much larger than m, we would like to arrange queries in a smaller number of rounds. In the following, we use a technique developed in Damaschke (1998) (for learning monotone boolean functions) to find one edge from a non-empty hypergraph with high probability using only $O(\log m + r)$ rounds and $O((\log m + r)\log n)$ queries. However, the algorithm is more involved.

The new algorithm is also based on FIND-ONE-VERTEX. The process of FIND-ONE-VERTEX can be viewed as a binary decision tree. At each internal node, a set A is split and a decision on which branch to follow is made based on query results. The FIND-ONE-VERTEX algorithm does not restrict how we split the set A as long as we divide it into halves. In the new algorithm, we will pre-determine the way A's will be divided at the very beginning of the algorithm.

Let us index each vertex by a distinct binary number $b_1 b_2 \ldots b_{\log n}$. Each split is based on a certain bit. We say that we split a set A according to its i^{th} ($i \in [1, \log n]$) bit, we will divide A into two sets, one containing vertices whose i^{th} bits are 0 and the other containing vertices whose i^{th} bits are 1. We will denote these two sets $A|_{b_i=0}$ and $A|_{b_i=1}$ respectively. If we split $A|_{b_i=0}$ further according to the j^{th} bit, we get another two sets $(A|_{b_i=0})|_{b_j=0}$ and $(A|_{b_i=0})|_{b_j=1}$. We will abbreviate these two sets as $A|_{b_i=0,b_j=0}$ and $A|_{b_i=0,b_j=1}$. In general, let s be a *partial assignment* that assigns some bits to 0 or 1, we use $A|_s$ to denote the set of vertices in A that match the assignments of bits in s.

Using this notation and our splitting scheme, at each iteration of FIND-ONE-VERTEX, A is equal to $V|_s$ for some partial assignment s, and A_0 and A_1 are equal to $A|_{b_i=0}$ and $A|_{b_i=1}$ if we split A according to the i^{th} bit. One of the key ideas in Damaschke (1998) is that because the splits are pre-determined, and the queries are monotone in terms of subset relation, we can make queries on pre-determined splits to make predictions. The idea will be made clear in the rest of the section. PARA-FIND-ONE-VERTEX (Algorithm 5) improves the round complexity of FIND-ONE-VERTEX.

Algorithm 5 PARA-FIND-ONE-VERTEX

```
1: $S \leftarrow V, A \leftarrow V, I \leftarrow [1, \log n]$.
2: while $|A| > 1$ do
3:   $\forall i \in I$, make queries on $(S \backslash A) \cup A|_{b_i=0}$ and $(S \backslash A) \cup A|_{b_i=1}$.
4:   Let $R_i = (Q((S \backslash A) \cup A|_{b_i=0}), Q((S \backslash A) \cup A|_{b_i=1}))$ be the query results for $i \in I$.
5:   case 1: $\exists i \in I$ such that $R_i = (0,0)$
6:     $A \leftarrow A|_{b_i=0}, I \leftarrow I \backslash \{i\}$.
7:   case 2: $\exists i \in I$ such that $R_i = (1,1)$
8:     Choose $a$ from $\{0,1\}$ uniformly at random.
9:     $A \leftarrow A|_{b_i=a}$, $S \leftarrow (S \backslash A) \cup A|_{b_i=a}, I \leftarrow I \backslash \{i\}$.
10:  case 3: $\forall i \in I$, $R_i = (1,0)$ or $R_i = (0,1)$
11:    Swap the indices of vertices so that $R_i = (1,0)$ for every $i \in I$. (If $R_i = (0,1)$, we flip the
       $i^{th}$ bit of all the indices, that is, for every vertex, if the $i^{th}$ bit of its index is 0, we set the $i^{th}$
       bit to 1 and vice versa.)
12:    $\forall i \in I$, let $A^i = A|_{\forall j \in I, j \leq i, b_j=0}$ and make a query on $S^i = (S \backslash A) \cup A^i$.
13:    Let $i^* = \min\{i | Q(S^i) = 0\}$ if it exists and the largest index in $I$ otherwise. Let $j^* =$
       $\max\{j | j < i^*, j \in I\}$.
14:    $I \leftarrow \{i | i > i^*, i \in I\}$.
15:    if all queries are answered 1 then
16:      $A \leftarrow A^{i^*}, S \leftarrow S^{i^*}$ ($i^*$ is the largest index in $I$ in this case).
17:    else
18:      $A \leftarrow A^{i^*}, S \leftarrow S^{j^*}$.
19:    end if
20: end while
21: Output the element in $A$.
```

In PARA-FIND-ONE-VERTEX, the equalities $Q(S) = 1, Q(S \backslash A) = 0$ are also preserved at all times, which establishes the correctness. We first make queries on $(S \backslash A) \cup A|_{b_i=0}$ $(= S \backslash A|_{b_i=1})$ and $(S \backslash A) \cup A|_{b_i=1}$ $(= S \backslash A|_{b_i=0})$ for every i. There are 3 possible query outcomes.

case 1: If there exists i such that $R_i = (0,0)$, that is, both queries are answered 0, all edges contained in S are split between $A|_{b_i=0}$ and $A|_{b_i=1}$, that is, the intersections of each edge with these two sets are a partition of the edge. We call this case an *edge-splitting event*. The iterations at which an edge-splitting event happens are *edge-splitting iterations*. Since we then set A to be $A|_{b_i=0}$, the intersection of A with any edge contained in S becomes strictly smaller. Because we will only shrink A in other cases, the intersections will never increase. Thus, there are at most $r-1$ edge-splitting iterations as each edge is of size at most r.

case 2: If there exists i such that $R_i = (1,1)$, that is, both queries are answered 1, we can set S to be either of the two sets $(S \backslash A) \cup A|_{b_i=0}$ and $(S \backslash A) \cup A|_{b_i=1}$ as they both contain edges, and set A to be $A|_{b_i=0}$ or $A|_{b_i=1}$ respectively. The equalities $Q(S) = 1, Q(S \backslash A) = 0$ are preserved in this case. However, we would like to choose whichever of the two sets $(S \backslash A) \cup A|_{b_i=0}$ and $(S \backslash A) \cup A|_{b_i=1}$ contains fewer edges. Because they do not share a common edge as their intersection $S \backslash A$ does not contain an edge, the sum of the numbers of edges contained in these two sets is at most the number of edges contained in S. If we choose the set with fewer edges, we will cut the number of edges contained in S in half. With a random choice, this happens with probability $1/2$. We will call this case an *edge-separating event* and call the corresponding iteration an *edge-separating iteration*.

case 3: If neither of the two events happens, we need to deal with the third case where $\forall i \in I$, one of the queries is answered 0 and the other is answered 1. In this case, for convenience of exposition, we will flip the indices of all vertices, so that $R_i = (1,0)$ for every $i \in I$. Thus, $\forall i \in I, Q((S \backslash A) \cup A|_{b_i=0}) = 1$. In this case, we can set A to $A|_{b_i=0}$ for some $i \in I$, and the equalities $Q(S) = 1, Q(S \backslash A) = 0$ are preserved. However, this won't help us to reduce the round complexity as it only cuts A in half.

Consider the next split. We shall divide $A|_{b_i=0}$ further into $A|_{b_i=0,b_j=0}$ and $A|_{b_i=0,b_j=1}$ for some $j \in I, j \neq i$. Since we already know that $Q((S \backslash A) \cup A|_{b_j=1}) = 0$, the fact that $A|_{b_i=0,b_j=1}$ is a subset of $A|_{b_j=1}$ implies $Q((S \backslash A) \cup A|_{b_i=0,b_j=1}) = 0$. Therefore, we only need to know $Q((S \backslash A) \cup A|_{b_i=0,b_j=0})$.

(a) If it is 1, we can set A to be $A|_{b_i=0,b_j=0}$ and continue.

(b) Otherwise, it is 0, an edge-splitting event takes place.

In PARA-FIND-ONE-VERTEX, we choose the indices we use to split A in the increasing order of indices in I and make queries on $S^i = (S \backslash A) \cup A^i$ for every $i \in I$ all in parallel (recall that $A^i = A|_{\forall j \in I, j \leq i, b_j=0}$). If all queries are answered 1, i^* is the largest index in I and A^{i^*} is a singleton set containing a relevant vertex. Otherwise, we get an edge-splitting event, since $S^{j^*} = (S \backslash A) \cup A^{j^*}$ contains edges, but both $(S \backslash A) \cup A^{j^*}|_{b_{i^*}=0}$ and $(S \backslash A) \cup A^{j^*}|_{b_{i^*}=1}$ don't (note that j^* is the index right before i^* in the increasing order of indices in I and $A^{i^*} = A^{j^*}|_{b_{i^*}=0}$). In this case, it can be verified that our updates to A and S in the third case preserve the equalities $Q(S) = 1, Q(S \backslash A) = 0$.

By the above analysis, the first case and the third case both result in an edge-splitting event or succeed in finding a relevant vertex. There are at most r such iterations. The second case results in an edge-separating event, in which with probability $1/2$ we will cut the number of edges contained in S in half. We can show that in expectation there are $\log m$ edge-separating events. Therefore, there are $\log m + r$ iterations in expectation. At each iteration, we use at most $3 \log n$ queries which are made in at most 2 rounds. Therefore,

Lemma 8 *In expectation, PARA-FIND-ONE-VERTEX finds one relevant vertex using $O((\log m + r) \log n)$ queries, and the queries can be made in $2(\log m + r)$ rounds.*

PARA-FIND-ONE-VERTEX can work with FIND-ONE-EDGE to find an edge using expected $O(r(\log m + r) \log n)$ queries in expected $2r(\log m + r)$ rounds. In fact, we can improve the round complexity further to $2(\log m + r)$ based on two observations, both of which use the fact that in the whole process we only shrink S.

The first observation is that edges removed from S in the edge-separating events will not be considered again. Therefore, the $\log m$ bounds not only the expected number of edge-separating iterations of PARA-FIND-ONE-VERTEX, but also that of the whole process.

The second observation is that the edge-splitting events can be remembered and reused when we try to find the next relevant vertex. Since we only shrink S, the bits that split all edges in S will continue to do so. Let I^* be the set of edge-splitting indices. In the new vertex finding process, instead of starting with $A = V = S$ (recall in a recursive call of FIND-ONE-EDGE, we look for a relevant edge contained in the S. Therefore, in the recursive call, V is equal to the S we obtain in the previous call), we start with $A = S|_{i \in I^*, b_i = 0}$. Note that the equalities $Q(S) = 1, Q(S \backslash A) = 0$ are preserved. This helps us to bound the number of edge-splitting iterations by $r - 1$ for the whole process.

Thus, we have the following lemma.

Lemma 9 *There is an algorithm that finds an edge in a non-empty hypergraph using expected* $O((\log m + r)\log n)$ *edge-detecting queries. Moreover, the queries can be made in expected* $2(\log m + r)$ *rounds.*

Since the algorithm terminates in expected $\log m + r$ iterations, according to Markov's Inequality, with probability at least $1/2$, the algorithm terminates in $2(\log m + r)$ iterations. We convert it to one that succeeds with high probability by running $\log \frac{1}{\delta}$ copies, each of which has its own independent random choices. All copies are synchronized at each iteration and the algorithm ends when one of them succeeds. This leads to an algorithm that succeeds with high probability. We will refer to this algorithm as PARA-FIND-ONE-EDGE.

Corollary 10 *With probability at least* $1 - \delta$, *PARA-FIND-ONE-EDGE finds an edge using* $O((\log m + r)\log n \log \frac{1}{\delta})$ *edge-detecting queries, and the queries can be made in* $4(\log m + r)$ *rounds.*

5. A Linear-Query Algorithm

Reconstructing an independent covering family at the discovery of every new edge is indeed wasteful. In this section we show how to modify the quadratic algorithm to obtain an algorithm using a number of queries that is linear in the number of edges. Our algorithm is optimal in terms of the dependence on m. Moreover, the queries can be made in $O(\min(2^r(\log m + r)^2, (\log m + r)^3))$ rounds.

Before we begin to describe our algorithm, we introduce some notation and make some definitions. First we reduce the discovery probabilities. Let

$$p_H(\chi) = 1/(2^{r+|\chi|+2} d_H(\chi))^{1/(r-|\chi|)},$$

where χ is a relevant vertex set. Let the *best discovery probability* of χ be the minimum discovery probability among all its subsets. That is,

$$p^*_H(\chi) = \min_{\chi' \subseteq \chi} p_H(\chi').$$

Definition 11 *Let* $\rho_\chi(p)$ *be the probability that a* (χ, p)*-sample is positive, where* χ *is a relevant vertex set.*

Remark 12 $\rho_\chi(p)$ *is continuous and monotonically increasing.*

Angluin and Chen (2004) contains a proof of this fact.

Definition 13 *Let* $p_\chi = \min\left\{p | \rho_\chi(p) = 1/2^{r+1}\right\}$ *be the* threshold probability *of a relevant vertex set* χ.

Remark 14 *Due to the fact that $\rho_\chi(0) = 0$, $\rho_\chi(1) = 1$ and that $\rho_\chi(p)$ is continuous and monotonically increasing, the threshold probability uniquely exists.*

Note that both threshold probabilities and discovery probabilities reflect the degree of set χ or the degrees of its subsets. The difference is that discovery probabilities reflect degrees in the hypergraph we have found, while threshold probabilities reflect degrees in the target hypergraph. Threshold probabilities are only used in analysis.

5.1 Overview Of The Algorithm

An “obvious” improvement to the quadratic algorithm is that instead of calling FIND-ONE-EDGE on one positive sample at each iteration, we can call it on all positive samples. It is plausible that this will yield more edges. However, there is no guarantee that different calls to FIND-ONE-EDGE will output different edges. For instance, calls to FIND-ONE-EDGE on two sets that share a common edge will produce the same edge in the worst case. We use several standard tricks to circumvent this obstacle. In fact, the family of samples constructed here is more complex than that used in Section 4, so as to ensure with high probability that the algorithm will make a certain amount of progress at each iteration. By doing so, we are able to reduce the number of iterations from m to $O(\min(2^r(\log m + r), (\log m + r)^2))$. The number of queries will also be reduced.

First of all, the sampling probabilities are halved in order to accommodate more edges. More precisely, imagine that we draw $(\chi, \frac{1}{2}p_H(\chi))$-samples instead of $(\chi, p_H(\chi))$-samples in the quadratic algorithm. Take a look at a sample drawn several iterations ago, which the quadratic algorithm did not call FIND-ONE-EDGE on. Such a sample will still have reasonable probability of excluding all the edges that have been found, as long as the degree of χ has not been increased by a factor of $2^{r-|\chi|}$ or equivalently the discovery probability of χ has not been decreased by half.

Second, the algorithm uses the best discovery probability for each relevant set. We call a relevant vertex set *minimal* if it has the minimum discovery probability among its subsets. In the quadratic algorithm, the goal is that one of the samples will produce an edge. According to the proof of Lemma 5, in the quadratic algorithm, we actually only need to draw samples for minimal relevant sets. In this algorithm, we hope that samples drawn for every relevant set will produce edges. But drawing samples for non-minimal relevant sets with discovery probabilities is not sufficient to avoid edges we have already found. Therefore, the best discovery probabilities are used.

Finally, besides samples drawn proportional to degrees, the algorithm also draws samples proportional to the contribution of each relevant set. The idea is simple. Draw more samples for those relevant sets that are more likely to produce a new edge. The algorithm maintains a *contribution* counter $c(\chi)$ for each relevant set χ, which records the number of new edges that χ-samples have produced. As we have already said, different calls to FIND-ONE-EDGE at each iteration may output the same edge. As all calls to FIND-ONE-EDGE at each iteration are made in parallel, it is not clear which sample each new edge should be attributed to. To solve this problem, calls to FIND-ONE-EDGE are processed sequentially in an arbitrary order.

In the algorithm, $\mathcal{F}_H$ consists of two parts: $\mathcal{F}_H^1$ and $\mathcal{F}_H^2$. In $\mathcal{F}_H^1$, the algorithm draws samples proportional to the contribution of each relevant set. $\mathcal{F}_H^2$ is closer to $\mathcal{F}_H$ in Section 4. Intuitively, a high-degree relevant set in the target hypergraph (not necessarily a high-degree relevant set in H), or a relevant set with small threshold probability is important, because an edge or a non-edge may not be found if its important relevant subsets are not found. The smaller the threshold probability a relevant set is, the more important it is. The algorithm uses samples in $\mathcal{F}_H^1$ to find edges while samples in $\mathcal{F}_H^2$ are mainly used to cover non-edges of H. $\mathcal{F}_H^2$ not only gives a short proof when H is indeed the target hypergraph, but also finds important relevant sets quickly. The design of $\mathcal{F}_H^2$ guarantees that if the contribution of the most important subset of an edge or a non-edge stops doubling, a more important relevant subset will be discovered with high probability.

5.2 The Algorithm

Let $H = (V, E)$ be the hypergraph the algorithm has found so far. δ' is a parameter we will specify later. The algorithm is shown in Algorithm 6. At each iteration, the algorithm operates in two phases, the query phase and the computation phase. In the query phase, the algorithm draws random samples and make queries. The queries can be made in $O(\log m + r)$ rounds, as queries of each call to PARA-FIND-ONE-EDGE can be made in $O(\log m + r)$ rounds. In the computation phase, the algorithm processes the query results to update the contribution counter of each relevant set and also adds newly found relevant sets.

Algorithm 6 The linear-query algorithm

```
    All PARA-FIND-ONE-EDGE's are called with parameter $\delta'$.
 1: $e \leftarrow$ PARA-FIND-ONE-EDGE($V$).
 2: $E \leftarrow \{e\}$. $c(\emptyset) \leftarrow 1$.
 3: repeat
      QUERY PHASE
 4:     Let $\mathcal{F}_H^1$ be a family that for every known relevant set $\chi$ contains $c(\chi) \cdot 2^{r+2} \ln \frac{1}{\delta'}$ $(\chi, \frac{1}{2} p_H^*(\chi))$-samples.
 5:     Let $\mathcal{F}_H^2$ be a family that for every known relevant set $\chi$ contains $2^{3r+3} d_H(\chi) \ln \frac{1}{\delta'}$ $(\chi, \frac{1}{4} p_H^*(\chi))$-samples.
 6:     Let $\mathcal{F}_H = \mathcal{F}_H^1 \cup \mathcal{F}_H^2$. Make queries on sets in $\mathcal{F}_H$ that are independent in $H$.
 7:     Call PARA-FIND-ONE-EDGE on all positive samples.
      COMPUTATION PHASE
 8:     For each known relevant set $\chi$, divide $\chi$-samples in $\mathcal{F}_H^1$ into $c(\chi)$ groups of size $2^{r+2} \ln \frac{1}{\delta'}$.
 9:     Process the samples in $\mathcal{F}_H^1$ group by group in an arbitrary order. Increase $c(\chi)$ by the number of new edges that $\chi$-samples produce. Add newly found edges to $E$.
10:     Process the samples in $\mathcal{F}_H^2$. Add newly found edges to $E$.
11:     For every newly found relevant set $\chi$, $c(\chi) \leftarrow 1$.
12: until no new edge is found
```

We will show that the algorithm terminates in $O(\min(2^r(\log m + r), (\log m + r)^2))$ iterations with high probability. Since $\sum_\chi d_H(\chi) \le 2^r m$ and $\sum_\chi c(\chi) \le (2^r + 1)m$ (note that $c(\chi)$ is one more than the number of new edges that χ-samples in $\mathcal{F}_H^1$ produce), the number of queries made at each iteration is at most $O(2^{4r} m \cdot poly(r, \log n, \log \frac{1}{\delta'}))$. Therefore, the total number of queries will be linear in the number of edges with high probability, as desired.

5.3 Analysis

Consider some iteration of the algorithm. Let H be the hypergraph the algorithm has found at the beginning of the iteration. Let e be an edge that has not yet been found. Let χ be a known subset of

e. χ can be either *active*, in which case a χ-sample is likely to contain an edge or *inactive* otherwise. Formally speaking,

Definition 15 *We say that χ is active if $\rho_\chi(\frac{1}{2}p^*_H(\chi)) \geq 1/2^{r+1}$ or, equivalently, $\frac{1}{2}p^*_H(\chi) \geq p_\chi$, and inactive otherwise.*

The following two assertions serve as the goals for each iteration.

Assertion 16 *Consider one group of χ-samples $\mathcal{G}$ in $\mathcal{F}^1_H$. Let H' be the hypergraph the algorithm has found before this group is processed. If χ is active, either $p^*_{H'}(\chi) < \frac{1}{2}p^*_H(\chi)$ or $\mathcal{G}$ will produce a new edge.*

Assertion 17 *If χ is inactive, then at the end of this iteration, either e has been found or a subset of e whose threshold probability is at most $\frac{1}{2}p_\chi$ has been found (a relevant subset is found when an edge that contains it is found).*

The following two lemmas show that both assertions hold with high probability.

Lemma 18 *Assertion 16 is true with probability at least* $1-\delta'$.

Proof If $p^*_{H'}(\chi) \geq \frac{1}{2}p^*_H(\chi)$, the probability that a χ-sample contains an edge in H' is at most

$$\sum_{\chi'\subseteq\chi} d_{H'}(\chi')(\frac{1}{2}p^*_H(\chi))^{r-|\chi'|} \leq \sum_{\chi'\subseteq\chi} d_{H'}(\chi')p^*_{H'}(\chi)^{r-|\chi'|} \leq \frac{2^{|\chi|}}{2^{r+|\chi|+2}} = \frac{1}{2^{r+2}}.$$

On the other hand, since χ is active, we have $\rho_\chi(\frac{1}{2}p^*_H(\chi)) \geq 1/2^{r+1}$. That is, with probability at least $1/2^{r+1}$ a χ-sample will contain an edge. Therefore the probability that a χ-sample contains a new edge is at least $1/2^{r+1} - 1/2^{r+2} = 1/2^{r+2}$. Recall that $\mathcal{G}$ contains $2^{r+2}\ln\frac{1}{\delta'}$ χ-samples. Therefore, with probability at least $1-\delta'$ there exists at least one sample in $\mathcal{G}$ that will produce a new edge. ■

Lemma 19 *Assertion 17 is true with probability at least* $1-\delta'$.

Proof Let $\chi^* \subseteq \chi$ have the minimum discovery probability among all subsets of χ at the beginning of the iteration. Thus, $p_H(\chi^*) = p^*_H(\chi)$ by the definition. Let us consider a χ^*-sample P_{χ^*} in $\mathcal{F}^2_H$. Let A be the collection of all subsets of e whose threshold probabilities are not less than $\frac{1}{2}p_\chi$. We do not want P_{χ^*} to contain any edge that contains χ' for any $\chi' \in A$ because they prevent us from discovering relevant sets with low threshold probabilities ($< \frac{1}{2}p_\chi$).

We observe that $\frac{1}{2}p_H(\chi^*) = \frac{1}{2}p^*_H(\chi) < p_\chi$ because χ is inactive. Thus, we have that $\forall\chi' \in A$,

$$\rho_{\chi'}(\frac{1}{4}p_H(\chi^*)) < \rho_{\chi'}(\frac{1}{2}p_\chi) \leq \rho_{\chi'}(p_{\chi'}) = 1/2^{r+1}.$$

Therefore,

$$Pr[\exists \text{ an edge } e' \subseteq P_{\chi^*}, e' \cap e \in A | e \subseteq P_{\chi^*}] \leq \sum_{\chi' \in A} \rho_{\chi'}(\frac{1}{4}p_H(\chi^*)) \leq 1/2.$$

Combining with the fact that

$$Pr[e \subseteq P_{\chi^*}] = (\frac{1}{4}p_H(\chi^*))^{r-|\chi^*|} = \frac{1}{2^{r+|\chi^*|+2+2r-2|\chi^*|}d_H(\chi^*)} \geq \frac{1}{2^{3r+2}d_H(\chi^*)},$$

we have that with probability at least $1/(2^{3r+3}d_H(\chi^*))$, P_{χ^*} contains e but does not contain any edge whose intersection with e is in A, in which case PARA-FIND-ONE-EDGE(P_{χ^*}) either outputs e or outputs an edge whose intersection with e has threshold probability at most $\frac{1}{2}p_\chi$. The probability that such a P_{χ^*} exists in $\mathcal{F}_H^2$ is at least $1-\delta'$, as we draw at least $2^{3r+3}d_H(\chi^*)\ln\frac{1}{\delta'}$ $(\chi^*, \frac{1}{4}p_H(\chi^*))$-samples. ■

Let H' be the hypergraph that has been found at the end of the iteration. Let $c_H(\chi)$ and $c_{H'}(\chi)$ be the values of $c(\chi)$ at the beginning and end of the iteration respectively. At each iteration, if no assertion is violated, one of the following two events happens.

1. Either $c_{H'}(\chi) \geq 2c_H(\chi)$ or $p^*_{H'}(\chi) < \frac{1}{2}p^*_H(\chi)$. ($c_H(\chi)$ doubles when each of the $c_H(\chi)$ groups of χ-samples in $\mathcal{F}_H^1$ succeeds in producing a new edge.)
2. Either e has been found or a subset of e whose threshold probability is at most $\frac{1}{2}p_\chi$ has been found.

That is, the two assertions guarantee that the algorithm makes definite progress at each iteration. The following lemma gives bound on the number of iterations of the algorithm.

Lemma 20 *Assuming no assertion is violated, the algorithm terminates in* $O(\min(2^r(\log m + r), (\log m + r)^2))$ *iterations.*

Proof First we remark that the minimum and maximum possible values for both discovery probabilities and threshold probabilities are $1/(2^{2r+1}m)$ and $1/2$ respectively, and the minimum and maximum possible values for $c(\chi)$ are 1 and $m+1$.

For each edge e, we divide the iterations into phases until e is found. Each phase is associated with a known relevant subset χ of e which has the minimum threshold probability at the beginning of the phase. A χ-phase ends when χ becomes inactive and then either e will be found or another relevant subset of e with at most half of χ's threshold probability will be found. Let us associate χ's threshold probability with a χ-phase. There are certainly at most 2^r phases because this is a bound on the number of subsets of e. Moreover, there are at most $O(\log m + r)$ phases as the associated threshold probability halves at the end of each phase. Furthermore, each phase takes at most $O(\log m + r)$ iterations, since either $c(\chi)$ doubles or the best discovery probability halves at each iteration. Therefore the algorithm terminates in $O(\min(2^r(\log m + r), (\log m + r)^2))$ iterations. ■

It is not hard to see that total number of assertions we need to satisfy before the algorithm succeeds is bounded by $poly(2^r, m)$, including the assertions that each PARA-FIND-ONE-EDGE will succeed. Choose $\delta' = \Theta(\delta/poly(2^r, m))$ and the algorithm will succeed with probability at least $1-\delta$. Although the choice of δ' requires knowledge of m, it is sufficient to use an upper bound of $\binom{n}{r}$, and we have that $\log\frac{1}{\delta'} \leq poly(r, \log n)\cdot\log\frac{1}{\delta}$. Since queries at each iteration are made in $O(\log m + r)$ rounds, it follows that

Theorem 21 *With probability at least $1-\delta$, Algorithm 6 learns an r-uniform hypergraph with m edges and n vertices, using $O(2^{4r}m \cdot poly(r, \log n, \log \frac{1}{\delta}))$ queries, in $O(\min(2^r(\log m + r)^2, (\log m + r)^3))$ rounds.*

6. Lower Bounds For Almost Uniform Hypergraphs

In this section, we derive a lower bound for the class of (r,Δ)-uniform hypergraphs. The following theorem is proved in Angluin and Chen (2004).

Theorem 22 $\Omega((2m/r)^{r/2})$ *edge-detecting queries are required to identify a hypergraph drawn from the class of all $(r, r-2)$-uniform hypergraphs with n vertices and m edges.*

We show that by a simple reduction this gives us a lower bound for general (r,Δ)-uniform hypergraphs.

Theorem 23 $\Omega((2m/(\Delta+2))^{1+\frac{\Delta}{2}})$ *edge-detecting queries are required to identify a hypergraph drawn from the class of all (r,Δ)-uniform hypergraphs with n vertices and m edges.*

Proof Given a $(\Delta+2,\Delta)$-uniform hypergraph $H=(V,E)$, let $H'=(V\cup V', E')$ be an (r,Δ)-uniform hypergraph, where $|V'| = r-\Delta-2$, $V'\cap V = \phi$ and $E' = \{e \cup V' | e \in E\}$. Any algorithm that learns H' can be converted to learn H with the same number of queries. ■

7. Learning Almost Uniform Hypergraphs

In this section, we extend our results to learning (r,Δ)-uniform hypergraphs. The query upper bound stated in the following theorem matches the lower bound of Theorem 23 in terms of dependence on m. The round upper bound is only $1+\Delta$ times more than that of Algorithm 6.

Theorem 24 *There is a randomized algorithm that learns an (r,Δ)-uniform hypergraph with m edges and n vertices with probability at least $1-\delta$, using $O(2^{O((1+\frac{\Delta}{2})r)} \cdot m^{1+\frac{\Delta}{2}} \cdot poly(\log n, \log \frac{1}{\delta})))$ queries. Furthermore, the queries can be made in $O((1+\Delta)\cdot \min(2^r(\log m + r)^2, (\log m + r)^3))$ rounds.*

7.1 The Algorithm

One of the main modifications is the use of new discovery probabilities. We first provide some intuition for the new discovery probabilities. We have been choosing the discovery probability for a relevant set χ to be inversely proportional to the $(r-|\chi|)^{th}$ root of its degree. It is so chosen that a χ-sample has good chance of excluding edges that contain χ. In an almost uniform hypergraph, we choose the discovery probabilities for the same purpose. In other words, we would like to choose p such that $\sum_{e\in E, e\supseteq\chi} p^{|e\backslash\chi|} \leq 1/2^{r+2}$. Similarly, we should set p to be inversely proportional to the w^{th} root of $d_H(\chi)$, where $w = \min_{e\supseteq\chi} |e\backslash\chi|$ is the minimum difference in cardinalities between edges containing χ and χ. However, w is no longer equal to $r-|\chi|$ as in uniform hypergraphs. There are two cases. When $|\chi| < r-\Delta$, we have $w \geq r-\Delta-|\chi|$ because the minimum edge size is $r-\Delta$; when $|\chi| \geq r-\Delta$, w can be as small as 1.

The case when $w = 1$ is special, as it implies that there exists an edge e such that $|e\backslash\chi| = 1$ or e has only one vertex v that χ does not have. We will call e a *1-edge* of χ. On one hand, any χ-sample containing v contains e, and hence is not an independent set; on the other hand, by excluding every vertex whose union with χ contains an edge, we can easily exclude all corresponding edges. Thus we remove these vertices from each χ-sample and the resulting sample, which we call a *modified χ-sample*, is an improvement over the original one. (We remark that this improvement is available for the uniform hypergraph problem in the case when $|\chi| = r-1$, but is not as important.) More specifically, let $\nu_H(\chi)$ be the set of all vertices v such that $\chi \cup \{v\}$ contains an edge in H. A modified (χ, p)-sample is a $(\chi, \nu_H(\chi), p)$-sample defined as follows.

Definition 25 *A* (χ, ν, p)-sample *($\chi \cap \nu = \emptyset$) is a random set of vertices that contains χ and does not contain any vertex in ν and contains each other vertex independently with probability p.*

Algorithm 7 Learning an (r, Δ)-uniform hypergraph

```
    All PARA-FIND-ONE-EDGE's are called with parameter $\delta'$.
 1: $e \leftarrow$ PARA-FIND-ONE-EDGE($V$).
 2: $E \leftarrow \{e\}$. $c(\emptyset) \leftarrow 1$.
 3: repeat
     QUERY PHASE
 4:    Let $\mathcal{F}_H^1$ be a family that for every known relevant set $\chi$ contains $c(\chi)\cdot 2^{r+2}\ln\frac{1}{\delta'}$ modified
       $(\chi, \frac{1}{2}p_H^*(\chi))$-samples and the same number of modified $(\chi, \frac{1}{2^{r+3+|\chi|}d_H(\chi)})$-samples.
 5:    Let $\mathcal{F}_H^2$ be a family that for every known relevant set $\chi$ contains $2(4/p_H(\chi))^{r-|\chi|}\ln\frac{1}{\delta'}$ modified
       $(\chi, \frac{1}{4}p_H^*(\chi))$-samples and $2^{r+4+|\chi|}d_H(\chi)\ln\frac{1}{\delta'}$ modified $(\chi, \frac{1}{2^{r+3+|\chi|}d_H(\chi)})$-samples.
 6:    Let $\mathcal{F}_H = \mathcal{F}_H^1 \cup \mathcal{F}_H^2$. Make queries on sets in $\mathcal{F}_H$ that are independent in $H$.
 7:    Call PARA-FIND-ONE-EDGE on all positive samples.
     COMPUTATION PHASE
 8:    For each relevant set $\chi$, divide $\chi$-samples in $\mathcal{F}_H^1$ in $c(\chi)$ groups of $2^{r+2}\ln\frac{1}{\delta'}$ modified
       $(\chi, \frac{1}{2}p_H^*(\chi))$-samples and the same number of modified $(\chi, \frac{1}{2^{r+3+|\chi|}d_H(\chi)})$-samples.
 9:    Process the samples in $\mathcal{F}_H^1$ group by group in an arbitrary order. Increase $c(\chi)$ by the number
       of new edges that $\chi$-samples produce. Add newly found edges to $E$.
10:    Process the samples in $\mathcal{F}_H^2$. Add newly found edges to $E$.
11:    1-edge-finder: For any $\chi$-sample $P_\chi \in \mathcal{F}_H^2$, let $e$ be the output of PARA-FIND-ONE-
       EDGE($P_\chi$). $\forall v \in e$, make a query on $\chi \cup \{v\}$ to test whether it is an edge. Add newly
       found edges to $E$.
12:    For every newly found relevant set $\chi$, $c(\chi) \leftarrow 1$.
13: until no new edge is found
```

We remark that we can use original χ-samples and obtain a much simpler algorithm than Algorithm 7. However, the query complexity will be roughly $m^{\Delta+2}$ instead of $m^{1+\frac{\Delta}{2}}$. The reduction of the complexity in the exponent of m is due to the fact that each modified χ-sample only needs to deal with edges that have at least 2 vertices that χ does not have. This leads to the definition of the *new discovery probability* as follows.

$$p_H(\chi) = \begin{cases} 1/(2^{r+|\chi|+2}d_H(\chi))^{1/(r-\Delta-|\chi|)}, & \text{if } |\chi| \le r-\Delta-2 \\ 1/(2^{r+|\chi|+2}d_H(\chi))^{1/2}, & \text{otherwise.} \end{cases}$$

We use the new discovery probabilities in Algorithm 7. Although we use modified samples, special care is still needed for 1-edges in order to parallelize the edge finding process. In fact, the majority of effort in developing Algorithm 7 is devoted to dealing with 1-edges.

In both $\mathcal{F}_H^1$ and $\mathcal{F}_H^2$, we draw $(\chi, \frac{1}{2^{r+3+|\chi|}d_H(\chi)})$-samples in addition. The reason for the design will be made clear in the analysis section. A group of χ-samples in $\mathcal{F}_H^1$ will consist of both $(\chi, \frac{1}{2}p_H^*(\chi))$-samples and $(\chi, \frac{1}{2^{r+3+|\chi|}d_H(\chi)})$-samples. $\mathcal{F}_H^2$ contains $(\chi, \frac{1}{4}p_H^*(\chi))$-samples as in Algorithm 6. Although, the number of $(\chi, \frac{1}{4}p_H^*(\chi))$-samples appears to be different from that of Algorithm 6, we remark that $2(4/p_H(\chi))^{r-|\chi|}\ln\frac{1}{\delta'}$ is bounded by $2^{3r+3}d_H(\chi)\ln\frac{1}{\delta'}$ under the definition of discovery probabilities in Section 5 and this group of samples are designed for essentially the same purpose as those for Algorithm 6. We also use a subroutine called *1-edge-finder*, specified in Algorithm 7.

7.2 Analysis

Round complexity

The following two definitions are analogous to those in Section 5. The extra subscript indicates that the new definitions depend on the already found sub-hypergraph H, while the previous definitions don't.

Definition 26 *Let $\rho_{\chi,H}(p)$ be the probability that a $(\chi, \nu_H(\chi), p)$-sample is positive, where χ is a vertex set that does not contain an edge.*

Definition 27 *Let $p_{\chi,H} = \min\left\{p|\rho_{\chi,H}(p) = 1/2^{r+1}\right\}$ be the* threshold probability *of χ.*

Now we bound the number of iterations of Algorithm 7. We divide the process of the algorithm into $(1+\Delta)$ phases, each of which is indexed by a number in $[r-\Delta, r]$. The phase l begins when all edges of size less than l have been found. Phase $r-\Delta$ is the first phase because there is no edge of size less than $r-\Delta$.

Let e be an edge of size l and χ be a known relevant subset of e. We need to deal with two cases : $|\chi| = l-1$ and $|\chi| \le l-2$, the latter of which is simpler as every 1-edge of χ has been discovered. We make the following definition.

Definition 28 *χ is active if it satisfies either of the following two conditions.*

1. *$|\chi| \le l-2$ and $\rho_{\chi,H}(\frac{1}{2}p_H^*(\chi)) \ge 1/2^{r+1}$.*
2. *$|\chi| = l-1$ and $\rho_{\chi,H}(\frac{1}{2}p_H^*(\chi)) \ge 1/2^{r+1}$ and $\rho_{\chi,H}(\frac{1}{2^{r+3+|\chi|}d_H(\chi)}) \ge 1/2^{r+1}$.*

It is inactive otherwise.

The definition is analogous to that in Section 5, and so are the following assertions. The assertions are made at phase l.

Assertion 29 *Consider one group of χ-samples $\mathcal{G}$ in $\mathcal{F}_H^1$. Let H' be the hypergraph the algorithm has found before the group is processed. If χ is active, one of the following three events happens.*

1. $p^*_{H'}(\chi) < \frac{1}{2}p^*_H(\chi)$,
2. $d_{H'}(\chi) > 2d_H(\chi)$, *or*
3. $\mathcal{G}$ *will produce a new edge.*

Assertion 30 *If χ is inactive, at the end of this iteration, e has been found or a subset of e whose threshold probability is at most $\frac{1}{2}p_{\chi,H}$ has been found.*

The two assertions guarantee that Algorithm 7 makes a certain progress at each iteration.

Lemma 31 *If no assertion is violated, phase l terminates in $O(\min(2^r(\log m + r), (\log m + r)^2))$ iterations.*

Proof We need to prove that every edge of size l can be found in the specified number iterations. Let e be an edge of size l. The proof proceeds similarly to that of Lemma 20. We divide the iterations into sub-phases, each of which is associated with a subset of e (we use sub-phases here to avoid confusion). Using an argument similar to that used in the proof of Lemma 20, we can show that each sub-phase takes $O(\log m + r)$ iterations. The only exception is that in this proof, the threshold probability of a set χ might not be fixed (it depends on the already found sub-hypergraph H). When more 1-edges of χ are found, $\rho_{\chi,H}(p)$ will decrease as more vertices are excluded from the sample. Therefore, $p_{\chi,H}$ might increase. After such a sub-phase, the associated threshold probability might not halve. However, this exception only happens when the subset associated with the sub-phase is of size $l-1$ and only happens $l \le r$ times as there are at most l such subsets and causes at most l additional sub-phases. Therefore, we get the same asymptotic bound on the number of sub-phases, which is $O(\min(2^r, \log m + r))$. This establishes the lemma. ■

Now we show the two assertions are true with high probability.

Lemma 32 *Assertion 29 is true for Algorithm 7 with probability at least* $1-\delta'$.

Proof $\mathcal{G}$ consists of two subgroups of samples with different sampling probabilities. In the analysis we will only consider one subgroup. In the case that $|\chi| \le l-2$, we use only $(\chi, \frac{1}{2}p^*_H(\chi))$-samples. In the case that $|\chi| = l-1$, we will use the subgroup with the smaller sampling probability. Let η be the sampling probability of the subgroup we consider. We have $\eta = \frac{1}{2}p^*_H(\chi)$ when $|\chi| \le l-2$ and $\eta = \min(\frac{1}{2}p^*_H(\chi), \frac{1}{2^{r+3+|\chi|}d_H(\chi)})$ when $|\chi| = l-1$. By our definition of active, in both cases $\rho_{\chi,H}(\eta) \ge 1/2^{r+1}$. The probability that a modified (χ,η)-sample contains an edge in H' is at most

$$\sum_{\chi' \subseteq \chi} d_{H'}(\chi') \cdot \eta^{\max(r-\Delta-|\chi'|,2)} + |\nu_{H'}(\chi) \backslash \nu_H(\chi)| \cdot \eta. \tag{1}$$

- When $|\chi| = l-2$, $|\nu_{H'}(\chi) \backslash \nu_H(\chi)| = 0$. Therefore, if $\eta = \frac{1}{2}p^*_H(\chi) \le p^*_{H'}(\chi)$, Equation (1) is at most $1/2^{r+2}$.
- When $|\chi| = l-1$, since every 1-edge of χ must contain χ in phase l, Equation (1) is bounded by
$$\sum_{\chi' \subset \chi} d_{H'}(\chi') \cdot \eta^{\max(r-\Delta-|\chi'|,2)} + d_{H'}(\chi) \cdot \eta.$$
If $\frac{1}{2}p^*_H(\chi) \le p^*_{H'}(\chi)$ and $d_{H'}(\chi) \le 2d_H(\chi)$, the above is bounded by $1/2^{r+2}$.

With probability at least $1/2^{r+2}$, a (χ,η)-sample contains an edge that is not contained in H'. Thus, with probability at least $1-\delta'$, $\mathcal{G}$ will produce a new edge. ■

Lemma 33 *Assertion 30 is true for Algorithm 7 with probability at least* $1-\delta'$.

Proof First we remark that if e has not been found, the probability that e is contained in a modified χ-sample is the same as for an unmodified one. This is because e does not contain any vertex in $\nu_H(\chi)$. Otherwise, e contains an edge in H, which violates our assumption that edges do not contain each other.

If $\rho_{\chi,H}(\frac{1}{2}p_H^*(\chi)) < 1/2^{r+1}$, the proof proceeds similarly to that of Lemma 19. We remark that the differences are that $\rho_{\chi,H}$ and $p_{\chi,H}$ are used instead of ρ_χ and p_χ and we draw more samples in $\mathcal{F}_H^2$ in Algorithm 7.

The remaining case is when $|\chi| = l-1$ and $\rho_{\chi,H}(\frac{1}{2^{r+3+|\chi|}d_H(\chi)}) < 1/2^{r+1}$. Consider a $(\chi, \frac{1}{2^{r+3+|\chi|}d_H(\chi)})$-sample P_χ in $\mathcal{F}_H^2$. Since e is of size l, we have $|e\backslash\chi| = 1$. Let $\{v\} = e\backslash\chi$. We have that

$$Pr[v \in P_\chi] = \frac{1}{2^{r+3+|\chi|}d_H(\chi)}$$

and

$$Pr[\exists \text{ an edge } e' \subseteq P_\chi \text{ such that } v \notin e' \mid v \in P_\chi] \le \rho_{\chi,H}(\frac{1}{2^{r+3+|\chi|}d_H(\chi)}) < 1/2^{r+1}.$$

Therefore, with probability at least $\frac{1}{2^{r+3+|\chi|}d_H(\chi)} \cdot (1-1/2^{r+1})$, P_χ contains v and contains only edges that are incident with v. Our 1-edge-finder will find e in this case. As we draw $2^{r+4+|\chi|}d_H(\chi)$ $\ln\frac{1}{\delta'}$ samples, e will be found with probability at least $1-\delta'$. ■

Since the algorithm has only $1+\Delta$ phases, the algorithm ends after $O((1+\Delta)\cdot\min(2^r(\log m + r), (\log m+r)^2))$ iterations. If no assertion is violated, the round complexity of Algorithm 7 is

$$O((1+\Delta)\cdot\min(2^r(\log m+r)^2, (\log m+r)^3))$$

We can choose δ' so that the algorithm succeeds with probability $1-\delta$ and $\log\frac{1}{\delta'} \le poly(r,\log n)\cdot\log\frac{1}{\delta}$.

Query complexity

The main discrepancy of the performance of this algorithm is due to the fact that in $\mathcal{F}_H^2$, the discovery probabilities are chosen as if all the edges were of minimum possible size, while the numbers of samples drawn are chosen as if all the non-edges (or potential edges) of H were of the maximum possible size. This causes the super-linear query complexity. At each iteration, the number of χ-samples in $\mathcal{F}_H^2$ is at most

$$2(4/p_H(\chi))^{r-|\chi|}\ln\frac{1}{\delta'} = \begin{cases} O((2^{O(r)}\cdot d_H(\chi))^{\frac{r-|\chi|}{r-\Delta-|\chi|}}\cdot\log\frac{1}{\delta'}) & \text{if } |\chi| \le r-\Delta-2 \\ O((2^{O(r)}\cdot d_H(\chi))^{1+\frac{\Delta}{2}}\cdot\log\frac{1}{\delta'}) & \text{otherwise.} \end{cases}$$

Note that $(r-|\chi|)/(r-\Delta-|\chi|)$ is at most $1+\frac{\Delta}{2}$ when $|\chi| \le r-\Delta-2$. Therefore, the number of modified χ-samples in $\mathcal{F}_H^2$ is at most $O((2^{O(r)}\cdot d_H(\chi))^{1+\frac{\Delta}{2}}\cdot\log\frac{1}{\delta'})$. Because $\sum_\chi d_H(\chi) \le 2^r m$

and $\forall\chi, d_H(\chi) \leq m$, we have $\sum_\chi d_H(\chi)^{1+\frac{\Delta}{2}} \leq (2^r m)^{1+\frac{\Delta}{2}}$. Therefore, the total number of queries the algorithm makes is bounded by

$$O(2^{O((1+\frac{\Delta}{2})r)} \cdot m^{1+\frac{\Delta}{2}} \cdot poly(\log n, \log \frac{1}{\delta})).$$

This finishes the proof of Theorem 24.

Acknowledgments

The conference version of this paper appears in Angluin and Chen (2005). The authors would like to thank the referees for helpful comments.

References

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 of Computing*, 33(2):487–501, 2004.

Dana Angluin and Jiang Chen. Learning a hidden graph using O(log n) queries per edge. In *Conference on Learning Theory*, pages 210–223. Springer, 2004.

Dana Angluin and Jiang Chen. Learning a hidden hypergraph. In *Conference on Learning Theory*, pages 561–575. Springer, 2005.

Richard Beigel, Noga Alon, Simon Kasif, Mehmet Serkan Apaydin, and Lance Fortnow. An optimal procedure for gap closing in whole genome shotgun sequencing. In *RECOMB*, pages 22–30, 2001.

Peter Damaschke. Adaptive versus nonadaptive attribute-efficient learning. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 590–596. ACM Press, 1998.

Vladimir Grebinski and Gregory Kucherov. Reconstructing a Hamiltonian cycle by querying the graph: Application to DNA physical mapping. *Discrete Applied Mathematics*, 88(1-3):147–165, 1998.

Vladimir Grebinski and Gregory Kucherov. Optimal reconstruction of graphs under the additive model. *Algorithmica*, 28(1):104–124, 2000.