

Lecture 14

①

Finding Signals (Continued)

1. Statistical Approach

2. Similarity-Based Approach

Uses spliced alignment algorithm

⋮

(*) The Twenty Questions Game with Genes

Given an (unknown) set I of integers in the interval $[1, n]$, reconstruct the set I by asking the minimum number of queries of the form "does a given interval contain an integer from I ?"

[Note: In this formulation, interval $[1, n]$ corresponds to

cDNA, " I " corresponds to exon boundaries in cDNA, and the

↓

queries correspond to PCR reactions defined by a pair of primers.

original gene with cut-out introns
(complementary copy of an mRNA)

實驗的範圍

(2)

$$(|I| = k)$$

If the number of exon boundaries k is known. Then we can do something on finding the lower bounds.

Lemma The number of queries is at least $\log_2 \binom{n}{k}$.

Proof. By decision tree model, the decision assumes sequential queries using one query at a time. Since there are $\binom{n}{k}$ possible outcomes and the answer is either yes or no, we conclude that we need at least h queries to obtain the answer where $2^h \geq \binom{n}{k}$. This concludes the proof. ■

(Note) $\log_2 \binom{n}{k} \approx k \log_2 n - k \log_2 k.$

$k = 1$ ✓

$k = 2$?

$$\log_2 \binom{n}{2} \leq h \leq 2 \log_2 n$$

$\uparrow$
(Informatic Lower Bound)

Gap

(3)

(*) If a biologist tolerates an error Δ in the positions of exon boundaries, then the lower bound on the number of queries is $\approx k \log_2 \frac{n}{\Delta} - k \log_2 k$.
(Not easy to verify!).

In practical situation, we use about 30 primers and
(typical cDNA),
3 rounds to find exon boundaries.

Hidden Graph Problem

A hidden graph G is known belonging to a given family $\mathcal{G}$ of labeled graphs on the set $N = [1, n]$.

The problem is to reconstruct G by asking queries as few as possible, where a query is of the form

(*) "Does $S \subseteq N$ induce one edge of G ?" That is " $G[S]$ at least contains an edge or $G[S]$ is not an empty graph (with no edges).

(*) Of course, we may ask $\binom{n}{2}$ pairs of vertices in $[1, n]$.
But, it takes too many queries.

④

Four models

- (1) No restriction on the size of S and the answer ^{is either} ~~are~~ "Yes" or "No".
(not both)
- (2) No restriction on the size of S and the answer is the number
of edges in $G[S]$.
- (3) $|S| \leq k$ and the answer is either Yes or No but not both.
- (4) $|S| \leq k$ and the answer is the size of $G[S]$.

- (1) General model
- (2) Quantity model
- (3) k -vertex general model
- (4) k -vertex quantity model

Algorithms

Step 1. Find a vertex of G .

Step 2. Find an edge of G .

Step 3. Find G . (The best idea so far is
via finding a maximal matching of
 G .)

5

Algorithm of Step 1 (Find-one-vertex)

1. $S \leftarrow N$

$$Q(S) = \begin{cases} 0, & G[S] \text{ contains no edges,} \\ 1, & \text{otherwise.} \end{cases}$$

2. if $Q(S) = 0$ then

3. Return $\emptyset$.

4. end if

5. $A \leftarrow N$

6. while $|A| > 1$ do

7. Partition (arbitrarily) A into two balanced sets A_0 and A_1

8. if $Q(S \setminus A_0) = 1$ then

9. $S \leftarrow S \setminus A_0, A \leftarrow A_1$.

10. else

11. $A \leftarrow A_0$

12. end if

13. end while

14. Return the element in A .

(*) If the hidden graph has m edges, then there are $\binom{n}{2}^m$ possible solutions for the graph.

(*) As mentioned above, if we apply the so-called binary splitting idea to find these m edges, then we need apply at least $\log_2 \left(\binom{n}{2}^m \right)$ queries.

(*) In fact, we need more than that. Observe that we need to find one edge to start, and then the second one, and so on. This fact was proved by Chang et al. later in a paper [1].

Theorem [1] $\lceil \log_2 \left(\sum_{i=0}^m \binom{n}{2}^i \right) \rceil$ edge-detecting queries are required to identify a graph drawn from the family of all graphs of order n and size m .

[1] Huilan Chang, H.L. Fu and Chih-Huai Shih, Learning a hidden graph, Optimization Letters, 8(2014), no.8, 2341-2348.

Proof. The main idea comes from the computation tree of search. From the leaves, we find the first edge. Then, delete the edge, ... (Omitted).

(*) After some effort, the following theorem was obtained.

Theorem [1]

For the family $\mathcal{G}$ of all graphs of size m on $[n]$, a hidden graph G can be reconstructed in $m \log_2 n + 10m + 3n$ edge-detecting queries.

(*) If we are dealing with hypergraphs with edge size larger than two, then it is getting more complicate.

Theorem [2] For an r -uniform hidden hypergraph ^{of size m} defined on $[n]$, we need $m r \log_2 n + (6e)^r m^{\frac{r+1}{2}}$ queries to reconstruct the graph.

[2] Learning a hidden uniform hypergraph (work with Huilan Chang and Chi-Huai Shih), Optim. Letters, 12 (2018), 55-62.

LEARNING A HIDDEN GRAPH

HUILAN CHANG, HUNG-LIN FU, AND CHIH-HUAI SHIH

ABSTRACT. We study the problem of learning a hidden graph by edge-detecting queries, each of which tells whether a set of vertices induces an edge of the hidden graph or not. We provide a new information-theoretic lower bound and give a more efficient adaptive algorithm to learn a general graph with n vertices and m edges in $m \log n + 10m + 3n$ edge-detecting queries.

1. INTRODUCTION

In a *graph learning problem*, a hidden graph G is known to belong to a given family $\mathcal{G}$ of labeled graphs on vertex set $[n] := \{1, 2, \dots, n\}$. Referring to the information of “belonging to $\mathcal{G}$ ”, we wish to identify G by *edge-detecting queries*, each of which tells whether a subset of $[n]$ induces an edge of G . Such a problem is motivated by applications in DNA physical mapping [14]. In the strategy proposed by Sorokin *et al.* [19] for physical mapping, some cloned fragments are produced to cover the whole DNA molecule with some gaps and then some of them are assembled to form longer continuous fragments (contigs). However, the information about the mutual placement of the contigs on the DNA sequence is lost. Multiplex PCR is a tool to tell whether there are two neighbouring contigs among a set of contigs so as to identify the relative replacements of the contigs. For example, if a DNA molecule is circular, then identifying every pair of close contigs can provide the order of contigs along the circular molecule; this identification work can be modeled by a graph learning problem on a family of Hamiltonian cycles and each Multiplex PCR experiment plays an edge-detecting role in graph learning problem.

Learning a hidden graph is also a subarea of combinatorial search on graphs [1]. Many conditions on $\mathcal{G}$ have been considered in the literature [3, 7, 8, 14, 15]: labeled graphs in $\mathcal{G}$ have prescribed topology such as Hamiltonian cycle, matching, tree, star, and clique; labeled graphs in $\mathcal{G}$ have bounded degree. The problem of learning a general graph is to deal with labeled graphs that have no restricted topology. Specifically, $\mathcal{G}$ collects all simple graphs on $[n]$. Angluin and Chen [6] proved that a hidden general graph can be identified in $12m \log n$ edge-detecting queries ($\log := \log_2$)

Date: May 1, 2014.

2010 Mathematics Subject Classification. 68W99, 05C90.

Key words and phrases. combinatorial search, graph learning, competitive group testing, complex model.

Partially supported by National Science Council, Taiwan under grant NSC 99-2811-M-009-056 (H.-L. Fu and C.-H. Shih) and 100-2115-M-390-004-MY2 (H. Chang).

*Optimization Letters, 8 (2014),
no. 8, 2341–2348.*

where m (unknown) is the number of edges in the hidden graph. This is the best known result to the best of our knowledge.

Learning a hidden general graph can be viewed as a variant of group testing which is a well-known field of combinatorial search. Given a set of items, each of which is either positive or negative, a *group test* on a subset of items determines whether it contains any positive item. The main task of *classical group testing problem* is to identify positive items by group tests. An extension of classical group testing is *the complex model*, where a set of complexes, each of which is a subset of basic items, is given and the property (positive or negative) of each complex is not yet determined; the corresponding query to identify positive complexes answers whether a subset of basic items contains at least one positive complex. In classical group testing, the number of positive items is usually assumed upper bounded while for *competitive group testing* [12], no information on the number of positive items is given. Learning a hidden graph is identical to a subcase of complex model where every set of two items is a complex; positive complexes correspond to edges and negative ones correspond to all pairs of items that are not edges. Many studies have been done on the classical competitive group testing (see [10, 11, 18] for recent development); however, to the best of our knowledge, very little is known about the competitive group testing on complex model.

An *affine plane* of order p contains p^2 points and $p^2 + p$ blocks of size p such that each pair of points occur together in exactly one of the blocks. Such an affine plane exists whenever p is a prime power [4]. The *affine plane method* proposed in [14, 20] and then employed in [9, 16] is to take an affine plane with the point set containing $[n]$ and then identify each subgraph induced by a block. In this paper, we concentrate on the study of learning a hidden general graph. We improve the trivial information-theoretic lower bound (see Section 2). Furthermore, by exploiting the affine plane method along with a fundamental algorithm, we can learn a general graph G on $[n]$ in $m \log n + 10m + 3n$ edge-detecting queries where m (unknown) is the number of edges of G , which is an improvement of the upper bound in [6] (see Section 3). Our algorithm performs as well as the algorithms in [9] for some families of graphs of known topology.

2. LOWER BOUND

Angluin and Chen [6] proved that for any $0 < \epsilon < 2$, $\epsilon m(\log n - 2)$ edge-detecting queries are required to identify a hidden graph G drawn from the family $\mathcal{G}$ of all graphs with n vertices and $m = n^{2-\epsilon}$ edges. In the following, we deal with the problem that the hidden G is drawn from the family of all graphs with n vertices, that is, the number of edges in G is unknown. We provide the following information-theoretic lower bound.

Theorem 2.1. $\lceil \log(\sum_{i=0}^m \binom{n}{i}) \rceil$ edge-detecting queries are required to identify a graph G drawn from the family of all graphs with n vertices if G has exactly m edges.

Before proving our main result on the lower bound, we need to establish some notions and a lemma. A query can be represented by a subset S of $[n]$ and we use $q(S)$ to denote its outcome while $q(S) = 1$ means the outcome is “yes” and $q(S) = 0$ means otherwise. For simplicity, we call a pair of vertices a *non-edge* if it does not induce an edge of the hidden graph. Let T_A denote the computation tree of an adaptive algorithm A that solves the problem of learning a general graph on $[n]$. Then each leaf represents a graph G on $[n]$ and in T_A the path from the root to the leaf G represents the query history of the hidden graph G under A . While applying an algorithm, pairs of vertices are determined as non-edges or edges gradually. For any query $S \subseteq [n]$ in the query history of G , let E_S^- (resp. E_S^+) collect all pairs of vertices that are determined as non-edges (resp. edges) in the partial history from the beginning to S . Let $\binom{S}{2} := \{\{i, j\} : i \neq j \in S\}$. Then $E_S^* := \binom{[n]}{2} \setminus (E_S^- \cup E_S^+)$ collects all pairs of vertices that are not determined right after S . Let $l_A(G)$ be the depth of a leaf G in T_A and $l_A(m) = \max\{l_A(G) : G \text{ has } m \text{ edges}\}$. First of all, we have the following result.

Lemma 2.2. Let $\mathcal{G}$ be the family of all graphs on $[n]$. Then for any algorithm A that solves the hidden graph problem on $\mathcal{G}$, there exists an algorithm A_m that also solves the hidden graph problem on $\mathcal{G}$ and satisfies $l_{A_m}(m) = l_A(m)$ and $l_{A_m}(r) \geq l_{A_m}(r-1)$ for any $r \leq m$.

Proof. Let A_m be a procedure obtained by adjusting A as follows: Suppose that G is the hidden graph in $\mathcal{G}$. While applying algorithm A , let S be the first query satisfying $|E_S^+ \cup E_S^*| = |\binom{[n]}{2} \setminus E_S^-| \leq m-1$ and S' be the query preceding S . Since the number of edges in the hidden graph is unknown, a positive query does not provide any information to declare any pair of vertices as non-edge. Hence if $q(S) = 1$, then $|\binom{[n]}{2} \setminus E_{S'}^-| = |\binom{[n]}{2} \setminus E_S^-| \leq m-1$, contradicting the selection of S . Therefore, $q(S) = 0$ and thus $\binom{S}{2} \cap E(G) = \emptyset$. Furthermore, assume without loss of generality that $\binom{S}{2} \not\subseteq E_{S'}^-$, since each pair of vertices in $E_{S'}^-$ has been determined as non-edge after S' . Now, we have an $e \in \binom{S}{2} \setminus E_{S'}^-$ that is an element in $E_{S'}^* \setminus E(G)$. In algorithm A_m , we turn to treat e as an edge and $q(S) = 1$; whenever a query after S contains e , the outcome is designated to be yes. After each pair of vertices is determined as edge or non-edge, return the hidden graph with e removed. Namely, we identify G by identifying some graph G' with $E(G') = E(G) \cup \{e\}$ where e is not an edge of G and is designated in the middle of excuting the algorithm A . It is clear that A_m also solves the hidden graph problem on $\mathcal{G}$.

Next, suppose that the hidden graph G has m edges. Then it is clear that any query S in the query history of the hidden graph G under algorithm A satisfies $|\binom{[n]}{2} \setminus E_S^-| \geq m$. Hence the query history is preserved to A_m . This implies $l_{A_m}(m) = l_A(m)$.

For $r \leq m$, let G_{r-1} be the worst case for A_m among all graphs with $r-1$ edges; namely, $l_{A_m}(G_{r-1}) = l_{A_m}(r-1)$. Let S be the first query satisfying $|E_S^+ \cup E_S^*| \leq m-1$ along the query history of G_{r-1} under A . Let G_r be the graph with $E(G_r) = E(G_{r-1}) \cup \{e\}$ where e is selected by the algorithm A_m as above. Then the query history of G_{r-1} is a part of G_r 's under A_m . Hence, $l_{A_m}(r) \geq l_{A_m}(G_r) \geq l_{A_m}(G_{r-1}) = l_{A_m}(r-1)$. $\blacksquare$

We are now ready to prove the main result on the lower bound.

Proof of Theorem 2.1: According to Lemma 2.2, $l_A(m) = l_{A_m}(m) \geq l_{A_m}(r)$ for any $r \leq m$. Then in T_{A_m} the depth of each leaf representing a hidden graph with at most m edges is at most $l_{A_m}(m)$. We have $l_A(m) = l_{A_m}(m) \geq \log(\sum_{i=0}^m \binom{\binom{n}{2}}{i})$. Namely, $\lceil \log(\sum_{i=0}^m \binom{\binom{n}{2}}{i}) \rceil$ queries are required to identify a graph (with m edges) drawn from the family of all graphs with n vertices. $\blacksquare$

Theorem 2.1 shows that the lower bound $\epsilon m(\log n - 2)$ cannot be achieved by any algorithm when the number of edges in the hidden graph is large.

3. EFFICIENT LEARNING OF A GENERAL GRAPH

To introduce our algorithm for identifying a hidden general graph, we start with some basic routines. Angluin and Chen [5] proved that

Lemma 3.1. *For any hidden hypergraph G on $[n]$ with each edge of size at most r , there is an algorithm that identifies an edge of G in $r \lceil \log n \rceil$ edge-detecting queries.*

A *binary splitting algorithm* [13] introduced to solve the classical group testing problem is as follows. Test the set of items. If the outcome is positive, test a set containing half of items, then we know that its outcome indicates either the tested half or the other half contains a positive item. Then iteratively halve the one with positives and test one of them until a positive item is obtained. Therefore, a binary splitting algorithm can identify all positive items in $d(\lceil \log n \rceil + 1) + 1$ group tests where d (unknown) is the number of positive items and the last 1 is contributed by the last test which ensures that there is no more positive item.

Let $G[S]$ denote the subgraph of G induced by a vertex set S and $G[X, Y]$ denote the bipartite subgraph of G induced by partite sets X and Y . A graph is called *non-trivial* if it contains at least one edge. The following result extended from the binary splitting algorithm plays a fundamental role in our main algorithm.

Lemma 3.2. *A bipartite graph G with partite sets X, Y can be identified in $|X| + m(\log |Y| + 2)$ edge-detecting queries where m (unknown) is the number of edges in G . In particular, if $1 = |X| \leq |Y|$ and G is known to be non-trivial, $m(\log |Y| + 2)$ edge-detecting queries are sufficient.*

Proof. G can be identified by the following procedure: for each $u \in X$ apply binary splitting algorithm on Y with u included in each test. Overall, it takes at most $|X| + m(\lceil \log |Y| \rceil + 1)$ queries. For $1 = |X| \leq |Y|$ and G is known to be non-trivial, a query on $\{u\} \cup Y$ can be saved if $X = \{u\}$. $\blacksquare$

We denote the procedure that identifies edges in a bipartite graph mentioned in the proof of Lemma 3.2 by IDENTIFY-BIPARTITE. We now give a 5-step algorithm IDENTIFY-HIDDEN-GRAPH (see Algorithm 1) to identify an arbitrary hidden graph G even though the number of edges in G is unknown. Our main strategy is to partition $[n]$, the set of vertices of G , into independent sets (see Step 1-2) and then identify the hidden subgraphs induced by all pairs of independent sets (see Step 3-5).

Algorithm 1 IDENTIFY-HIDDEN-GRAPH

Step 1 Identify a maximal matching:

Find a maximal matching M of G by iteratively identifying an edge and removing its two vertices from G until no more edge can be found. There remains an independent set, say I .

Return: M and I .

Step 2 Decompose $G[[n] \setminus I]$ into bipartite subgraphs:

Initialization: Take an edge $e = \{u, v\}$ from M and let $B_1 = G[\{u\}, \{v\}]$ and $k = 1$. Remove e from M .

Iteration: If $M = \emptyset$, then stop the iteration; otherwise, remove a new edge $e = \{u', v'\}$ from M and let $i = 1$.

(*) If $i > k$, then let $k \leftarrow k + 1$ and $B_k := G[\{u'\}, \{v'\}]$, and go back to the beginning of the iteration; otherwise, for $B_i := G[X, Y]$, apply queries on $X \cup \{u'\}$, $X \cup \{v'\}$, $Y \cup \{u'\}$ and $Y \cup \{v'\}$. If either $q(X \cup \{u'\}) = 0 = q(Y \cup \{v'\})$ or $q(X \cup \{v'\}) = 0 = q(Y \cup \{u'\})$, then include u', v' into X and Y correspondingly such that $B_i = G[X, Y]$ is bipartite and then go back to the beginning of the iteration; otherwise, let $i \leftarrow i + 1$ and go back to (*).

Return: $B_1, \dots, B_k$.

Step 3 Identify each $B_i := G[X_i, Y_i]$ by applying IDENTIFY-BIPARTITE on $G[\{u\}, Y_i \setminus \{v\}]$ for each $u \in X_i$ where $\{v, u\}$ is an edge in M .

Step 4 For any $B_i = G[X, Y]$ and $B_j = G[X', Y']$ with $1 \leq i < j \leq k$, apply IDENTIFY-BIPARTITE to $G[X, \{u\}]$, $G[Y, \{u\}]$, $G[X, \{v\}]$, and $G[Y, \{v\}]$ for every $u \in X'$ and $v \in Y'$ with $\{u, v\} \in M$.

Step 5 Apply IDENTIFY-BIPARTITE to $G[\{v\}, I]$ for each $v \in [n] \setminus I$.

Theorem 3.3. *For the family $\mathcal{G}$ of all graphs on $[n]$, a hidden graph G drawn from $\mathcal{G}$ can be identified in $m(\log n + 4) + \alpha'(G)(\log n + 5) + 1$ edge-detecting queries where m is the number of edges in G and $\alpha'(G)$ is the maximum size of matching in G .*

Proof. Denote the number of edges identified in Step i by m_i . Step 1 clearly returns a maximal matching M and an independent set I . Since identifying an edge can be done in $2\lceil \log n \rceil (\leq 2\log n + 2)$ queries (Lemma 3.1), M is identified in $|M|(2\log n + 2) + 1$ queries where the last test is to ensure that the remaining graph is trivial after M is identified.

Before each iteration in Step 2, some bipartite graphs $B_1, B_2, \dots, B_k$ are built. In the iteration, a new edge $e = \{u', v'\}$ is removed from M , and we find the smallest index i such that $B_i \cup \{e\}$ is bipartite; if no such a B_i exists, then the edge e is isolated as a new bipartite graph. Determining whether $B_i \cup \{e\}$ is bipartite for a bipartite graph $B_i = G[X, Y]$ can be accomplished by using the outcomes of the four queries $X \cup \{u'\}$, $X \cup \{v'\}$, $Y \cup \{u'\}$ and $Y \cup \{v'\}$. Finally, Step 2 returns bipartite graphs $B_1, \dots, B_k$, and $[n] \setminus I$ is successfully partitioned into independent sets which are partite sets of $B_1, \dots, B_k$. To analyze the number of queries spent in Step 2, we consider the following two cases of pairs of e and B_i in the iteration:

Case 1: (e, B_i) s where e is not merged into B_i . Then w.l.o.g. $q(X \cup \{u'\}) = 1 = q(Y \cup \{u'\})$ or $q(X \cup \{u'\}) = 1 = q(X \cup \{v'\})$. In either case, at least two edges containing u' or v' will be identified in Step 4. Therefore, the total number of queries spent in this case for all possible pairs of B_i and e is at most $2m_4$.

Case 2: (e, B_i) s where e is merged into B_i . Since each edge e is merged into B_i at most once, at most $4|M|$ queries are spent in this case.

Hence, there are at most $2m_4 + 4|M|$ queries spent in Step 2.

After Step 1 and Step 2, all the remaining hidden edges are between I and the partite sets of $B_1, \dots, B_k$ and can be classified into three types:

Type 1: edges in some B_i but not in M ,

Type 2: edges in $G[X, X']$, $G[X, Y']$, $G[Y, X']$ and $G[Y, Y']$ for some $B_i := G[X, Y]$ and $B_j := G[X', Y']$,

Type 3: edges containing a vertex in I .

Step 3-5 clearly identify edges of Type 1-3, respectively. Therefore, the correctness of IDENTIFY-HIDDEN-GRAPH is obtained. It remains to analyze the number of queries spent in step 3-5. In Step 3, since $|Y_i \setminus \{v\}| < |M| \leq \frac{n}{2}$ for $i = 1, \dots, k$, by Lemma 3.2 the total number of queries spent here is at most $m_3(\log \frac{n}{2} + 2) + |X_1| + |X_2| + \dots + |X_k| = m_3(\log n + 1) + |M|$. In step 4, according to the construction of B_i and B_j (see Step 2), at least two of $G[X, \{u\}]$, $G[Y, \{u\}]$, $G[X, \{v\}]$, and $G[Y, \{v\}]$ are known to be non-trivial and thus at most two of them are trivial. Therefore, in Step 4 the number of trivial subgraphs that IDENTIFY-BIPARTITE is applied to is at most m_4 . Hence by Lemma 3.2 Step 4 takes at most $\lceil m_4(\log \frac{n}{2} + 2) \rceil + m_4 \leq m_4 \log n + 2m_4$ queries. Finally, since $|[n] \setminus I| = 2|M|$, by Lemma 3.2 at most $2|M| + m_5(\log n + 2)$ queries are spent in Step 5.

Overall, IDENTIFY-HIDDEN-GRAPH identifies the hidden graph in

$$(m_3 + m_4 + m_5)(\log n + 4) + |M|(2 \log n + 9) + 1 = m(\log n + 4) + |M|(\log n + 5) + 1$$

queries where $m_3 + m_4 + m_5 + |M| = m$. Therefore, the theorem follows. $\blacksquare$

Recall that an affine plane of order p contains p^2 points and $p^2 + p$ blocks of size p such that each pair of points occur together in exactly one of the blocks. Such an affine plane exists whenever p is a prime power. The affine plane method is to take an affine plane with the point set containing $[n]$ and then identify each subgraph induced by a block. Since each pair of points occurs together in exactly one of the blocks, the hidden graph can be determined by identifying the subgraphs induced by blocks. Each subgraph induced by a block has exactly p vertices. Therefore, to make the method efficient, we need a small prime power p satisfying $p^2 \geq n$. For $n > 24^2$, there is a prime p such that $n \leq p^2 \leq 2n$ (obtained from [17]; see [9, 16]) and for $n \leq 24^2$, a prime power p satisfying $n \leq p^2 \leq 2n$ can be found easily. Hence,

Lemma 3.4. *For $n > 1$, there exists a prime power p such that $n \leq p^2 \leq 2n$.*

We then identify the graph induced by a block by using IDENTIFY-HIDDEN-GRAPH.

Theorem 3.5. *For the family $\mathcal{G}$ of all graphs on $[n]$, a hidden graph G drawn from $\mathcal{G}$ can be identified in $m \log n + 10m + 3n$ edge-detecting queries where m is the number of edges in G .*

Proof. By the definition of affine plane, we know that each hidden edge appears in exactly one of the subgraphs induced by blocks. Thus the total number of edges identified from blocks is exactly m . Let $\mathcal{B}$ be the set of blocks. Since $\sum_{B \in \mathcal{B}} \alpha'(G[B]) \leq m$ and the subgraph induced by a block has p ($\leq \sqrt{2n}$) vertices, by Theorem 3.3 the hidden graph can be identified in

$$\begin{aligned} & \sum_{B \in \mathcal{B}} (||G[B]||(\log p + 4) + \alpha'(G[B])(\log p + 5) + 1) \\ & \leq m(\log p + 4) + m(\log p + 5) + (p^2 + p) \\ & \leq m \log n + 10m + 2n + \sqrt{2n} \end{aligned}$$

queries where $||G[B]||$ is the number of edges in $G[B]$. $\blacksquare$

4. CONCLUSION

For the graph learning problem on the family of all graphs with n vertices, we first provide a better information-theoretic lower bound $\log(\sum_{i=0}^m \binom{n}{i})$ (m is the number of edges in the hidden graph). We further provide a deterministic algorithm, IDENTIFY-HIDDEN-GRAPH, together

with the affine plane method to identify the hidden graph in $m \log n + 10m + 3n$ edge-detecting queries. Furthermore, for some families of graphs of known topology, our algorithm exhibits the same economic performance as the algorithms presented in [9] do (see the following table).

	Hamiltonian cycles	Perfect matchings
Information lower bound	$n \log n$	$(1 + o(1))(\frac{n}{2} \log n)$
Chang <i>et al.</i> [9]	$n \log n + 10n$	$\frac{n}{2} \log n + 4n$
IDENTIFY-HIDDEN-GRAPH with affine plane method	$n \log n + 13n$	$\frac{n}{2} \log n + 8n$

Acknowledgements. The authors would like to express their gratitude to the referees for their valuable comments and suggestions in improving the presentation of this paper.

REFERENCES

- [1] Aigner, M.: Combinatorial Search. John Wiley and Sons (1988)
- [2] Alon, N., Asodi, V.: Learning a hidden subgraph. SIAM J. Discrete Math. 18, 697-712 (2005)
- [3] Alon, N., Beigel, R., Kasif, S., Rudich, S., Sudakov, B.: Learning a hidden matching. SIAM J. Comput. 33, 487-501 (2004)
- [4] Anderson, I.: Combinatorial Designs: Construction Methods. Ellis Horwood (1990)
- [5] Angluin, D., Chen, J.: Learning a hidden hypergraph. J. Mach. Learn. Res. 7, 2215-2236 (2006)
- [6] Angluin, D., Chen, J.: Learning a hidden graph using $O(\log n)$ queries per edge. J. Comput. Syst. Sci. 74, 546-556 (2008)
- [7] Beigel, R., Alon, N., Apaydin, M. S., Fortnow, L., Kasif, S.: An optimal procedure for gap closing in whole genome shotgun sequencing, Proc. 2001 RECOMB, ACM Press, 22-30
- [8] Bouvel, M., Grebinski, V., Kucherov, G.: Combinatorial search on graphs motivated by bioinformatics applications: a brief survey. WG, LNCS, 3787, 16-27 (2005)
- [9] Chang, H., Chen, H.-B., Fu, H. L., Shih, C. H.: Reconstruction of hidden graphs and threshold group testing. J. Combin. Optim. 22, 270-281 (2011)
- [10] Damaschke, P., Sheikh Muhammad, A.: Competitive group testing and learning hidden vertex covers with minimum adaptivity. Discrete Math., Alg. and Appl. 2, 291-311 (2010)
- [11] Damaschke, P., Sheikh Muhammad, A.: Bounds for nonadaptive group tests to estimate the amount of defectives. Discrete Math., Alg. and Appl. 3, 517-536 (2011)
- [12] Du, D. Z., Hwang, F. K.: Combinatorial Group Testing and its Applications (2nd Edition). World Scientific, Singapore (2000)
- [13] Du, D. Z., Hwang, F. K.: Pooling Designs and Nonadaptive Group Testing - Important Tools for DNA Sequencing. World Scientific (2006)
- [14] Grebinski, V., Kucherov, G.: Reconstructing a Hamiltonian cycle by querying the graph: Application to DNA physical mapping. Discrete Appl. Math., 88, 147-165 (1998)
- [15] Grebinski, V., Kucherov, G.: Optimal reconstruction of graphs under the additive model. Algorithmica, 28, 104-124 (2000)
- [16] Hwang, F. K., Liu, Y. C.: Error-tolerant pooling designs with inhibitors. J. Comput. Biol., 10, 231-236 (2003)
- [17] Nagura, J.: On the interval containing at least one prime number, Proceedings of the Japan Academy. Series A, 28, 177-181 (1952)
- [18] Schlaghoff, J., Triesch, E.: Improved Results for Competitive Group Testing. Comb., Prob. and Comp., 14, 191-202 (2005)
- [19] Sorokin, A., Lapidus, A., Capuano, V., Galleron, N., Pujic, P., Ehrlich, S.D.: A new approach using multiplex long accurate PCR and yeast artificial chromosomes for bacterial chromosome mapping and sequencing. Genome Res., 6, 448-453 (1996)

- [20] Tettelin, H., Radune, D., Kasif, S., Khouri, H., S. L. Salzberg, S. L.: Optimized multiplex PCR: Efficiently closing a whole-genome shotgun sequencing project. *Genomics*, 62, 500-507 (1996)

H. CHANG: DEPARTMENT OF APPLIED MATHEMATICS, NATIONAL UNIVERSITY OF KAOHSIUNG, KAOHSIUNG 811, TAIWAN, ROC
E-mail address: huilan0102@gmail.com

H.-L. FU: DEPARTMENT OF APPLIED MATHEMATICS, NATIONAL CHIAO TUNG UNIVERSITY, HSINCHU 30010, TAIWAN, ROC
E-mail address: hl fu@math.nctu.edu.tw

C.-H. SHIH: DEPARTMENT OF APPLIED MATHEMATICS, NATIONAL CHIAO TUNG UNIVERSITY, HSINCHU 30010, TAIWAN, ROC
E-mail address: skyking.shih@gmail.com

Learning a hidden uniform hypergraph

Huilan Chang¹  · Hung-Lin Fu² · Chih-Huai Shih²

Received: 9 September 2016 / Accepted: 11 March 2017
© Springer-Verlag Berlin Heidelberg 2017

Abstract Motivated by applications in genome sequencing, Grebinski and Kucherov (Discret Appl Math 88:147–165, 1998) studied the graph learning problem which is to identify a hidden graph drawn from a given class of graphs with vertex set $\{1, 2, \dots, n\}$ by edge-detecting queries. Each query tells whether a set of vertices induces any edge of the hidden graph or not. For the class of all hypergraphs whose edges have size at most r , Chodoriwsky and Moura (Theor Comput Sci 592:1–8, 2015) provided an adaptive algorithm that learns the class in $O(m^r \log n)$ queries if the hidden graph has m edges. In this paper, we provide an adaptive algorithm that learns the class of all r -uniform hypergraphs in $mr \log n + (6e)^r m^{\frac{r+1}{2}}$ queries if the hidden graph has m edges.

Keywords Graph learning · Group testing · Complex model · Adaptive algorithm

1 Introduction

A *graph learning problem* is to identify a hidden graph G drawn from a class $\mathcal{G}$ of labeled graphs by using *edge-detecting* queries of the following form

✉ Huilan Chang
huilan0102@gmail.com
Hung-Lin Fu
hlfu@math.nctu.edu.tw
Chih-Huai Shih
skyking.shih@gmail.com

¹ Department of Applied Mathematics, National University of Kaohsiung, Kaohsiung 811, Taiwan, Republic of China

² Department of Applied Mathematics, National Chiao Tung University, Hsinchu 30010, Taiwan, Republic of China

$Q(S)$: does a set S of vertices induce any edge of G ?

If S induces some edge of G , then the query $Q(S)$ is answered yes; otherwise, $Q(S)$ is answered no. The graph learning problem originated from applications in DNA physical mapping [10]. Such an application is based on a strategy proposed by Sorokin et al. [14]: In physical mapping, the whole DNA molecule is covered by some replicate cloned fragments with some gaps and some of the cloned fragments are assembled to form longer continuous fragments (contigs). However, the information about the mutual placement of the contigs on the DNA sequence is lost. One can identify the relative placements of two contigs by using multiplex polymerase chain reaction (*multiplex PCR*) which is an experiment telling whether there are two neighbouring contigs in a set of contigs. For example, if the structure of the DNA molecule is circular, then the order of contigs along the circular molecule can be determined by identifying every pair of neighbouring contigs. This identification work can be modeled by a graph learning problem on the class of Hamiltonian cycles and each Multiplex PCR experiment plays the edge-detecting role. Many conditions on $\mathcal{G}$ have been considered in the literature [2, 3, 5–7, 10, 11]. The problem of learning a general graph is to deal with labeled graphs that have no restricted topology; specifically, $\mathcal{G}$ consists of all simple graphs on the vertex set $[n] := \{1, 2, \dots, n\}$. Chang et al. [8] provided an adaptive algorithm that learns the class of all simple graphs on $[n]$ with at most $m \log n + 10m + 3n$ edge-detecting queries when the hidden graph has m edges.

Learning a hidden general graph can be described as a group testing problem which is a well-known field of combinatorial search [1]. Given a set of items, each of which is either positive or negative, the task of the *group testing problem* is to identify all positive items by group tests. A *group test* is applied to a subset of items and tells whether the subset contains any positive. An extension of the classical group testing is the *complex model*, where the property of being positive is not defined on single items but on subsets of items, called *complexes*. The corresponding group test used to identify positive complexes tells whether a subset of items contains any positive complex. Torney [15] first introduced the complex model and took substances in eukaryotic DNA transcription and RNA translation as examples of complexes. Group testing on complexes is widely applied in modern molecular and cellular biology. A prominent example is its application in the identification of protein-to-protein interactions (Lappe and Holm [12]; Li et al. [13]). In the signal transduction process, the protein-to-protein interactions of the signaling molecules can convey signals from the exterior of a cell to the inside of that cell. This conveying process plays a fundamental role in living cells. Information about the interactions between proteins improves our understanding of diseases and then provides the basis for new therapeutic approaches. The development of some laboratory approaches (Lappe and Holm [12]) enables the application of group testing to this problem. Li et al. [13] formulated this identification problem as a group testing problem in bipartite graphs which can be regarded as a special case of group testing on complexes. Learning a hidden graph is identical to a case of the complex model where every pair of items form a complex; a positive complex corresponds to an edge of the hidden graph and a negative complex corresponds to a pair of vertices which do not form an edge of the hidden graph.

Learning a hidden hypergraph is a natural generalization of the problem of learning a hidden graph. A hypergraph is r -uniform if each edge contains exactly r vertices. Thus a simple graph can be referred to as a 2-uniform hypergraph. An (r_1, r_2) -uniform hypergraph is a hypergraph whose minimum and maximum edge sizes are r_1 and r_2 , respectively. For the sake of simplicity, we always use the notion “hidden graph” even when the hidden one is a hypergraph; the term “hypergraph” is used only when we are defining a graph class that consists of some hypergraphs. Use m to denote the number of edges of the hidden graph. It is an unknown parameter if the graph class does not consist of graphs of a fixed number of edges. Angluin and Chen [4] showed that the class of all (r_1, r_2) -uniform hypergraphs is learnable with $O(f_1(r_1, r_2)m^{1+\frac{r_2-r_1}{2}} \text{poly}(\log n) \log(1/\delta))$ queries with probability at least $1 - \delta$, where f_1 depends only on r_1 and r_2 . Especially, they showed that the class of all r -uniform hypergraphs is learnable with $O(2^{4r}m \cdot \text{poly}(r, \log n) \log(1/\delta))$ queries with probability $1 - \delta$, where the queries can be made in $O(\min(2^r r^2 \log^2 n, r^3 \log^3 n))$ rounds. For the class of all hypergraphs whose maximum edge size is r , Chodoriwsky and Moura [9] provided an adaptive algorithm to identify a hidden graph G drawn from this class in $O(m^r \log n)$ queries.

In this paper, we focus on learning a hidden r -uniform hypergraph. We provide a deterministic adaptive algorithm to learn the class of all r -uniform hypergraphs within $mr \log n + (6e)^r m^{\frac{r+1}{2}}$ queries.

2 Lower bound

We start off this section by presenting some lower bounds. Angluin and Chen [5] provided an information-theoretic lower bound on learning the graph class that consists of all simple graphs of n vertices with a given number of edges. This result can be easily extended to the class of all r -uniform hypergraphs.

Theorem 2.1 *For any $0 < \epsilon < 1$, $\Omega(\epsilon mr \log n)$ edge-detecting queries are required to identify a hidden graph drawn from the class of all r -uniform hypergraphs with n vertices and $m = (\frac{n^{(1-\epsilon)}}{r})^r$ edges.*

Proof Since there are at least $\binom{n}{m} \geq \left(\frac{n}{m}\right)^m$ r -uniform hypergraphs with m edges and each edge-detecting query is answered 1 or 0, we obtain the following information-theoretic lower bound

$$\log \left(\frac{\left(\frac{n}{r}\right)^r}{m} \right)^m = m \log \frac{n^r}{r^r m} = \epsilon mr \log n.$$

□

Let $\mathcal{G}$ be the class of all $(2, r)$ -uniform hypergraphs on the vertex set $[n]$ with m edges. To provide a lower bound on learning the graph class $\mathcal{G}$, Angluin and Chen [5] considered a subclass $\mathcal{G}' \subset \mathcal{G}$, where each hypergraph in $\mathcal{G}'$ consists of exactly one edge of size r and some edges of size two. Adversary arguments are useful strategies to obtain lower bounds for combinatorial search problems. Angluin and Chen [5] used

this technique to obtain a lower bound on learning $\mathcal{G}'$ and thus gave the following lower bound on learning $\mathcal{G}$.

Theorem 2.2 (Angluin and Chen [5]) $\Omega((2m/r)^{r/2})$ edge-detecting queries are required to identify a hidden graph drawn from the class of all $(2, r)$ -uniform hypergraphs with n vertices and m edges.

3 Learning an r -uniform hypergraph

In this section, we introduce our adaptive algorithm RECONSTRUCT- r -UNIFORM-HYPERGRAPH that identifies a hidden graph G drawn from the class of all r -uniform hypergraphs on the vertex set $[n]$. Our algorithm first identifies an edge of the hidden graph. The complexity of identifying an edge has been studied by Angluin and Chen [4].

Lemma 3.1 (Angluin and Chen [4]) *For any hidden graph G on $[n]$ with each edge of size at most r , there is an algorithm that identifies an edge of G in $r \lceil \log n \rceil$ edge-detecting queries.*

Let S be a subset of vertices of a hypergraph G . A subgraph of G induced by S , denoted by $G[S]$, is a graph with vertex set S and whose edge set consists of edges of G that are subsets of S . To deal with the problem of learning a general graph (with each edge of size two), the following idea has been used by Chang et al. [8]: partition the vertex set $[n]$ into parts $S_1, S_2, \dots, S_l$ such that each edge of G has at most one vertex in $S_i, i = 1, 2, \dots, l$ and then identify the hidden subgraph induced by each pair of $S_1, S_2, \dots, S_l$. In our algorithm, we adopt this idea as a beginning strategy. Note that our algorithm is applicable even though the number of edges in G is unknown.

We first outline the steps of our algorithm (Algorithm 1). In Step 1, we first identify an edge, say e , and divide all vertices in $[n]$ into r parts such that e intersects each part in at most one vertex. We continue to identify edges induced by the union of every $r - 1$ parts and rearrange parts (new parts are created if necessary) such that each identified edge intersects each part in at most one vertex. Suppose that at the end of Step 1 there are l parts, say $S_1, S_2, \dots, S_l$. Then the union of any $r - 1$ parts from $S_1, S_2, \dots, S_l$ induces no edge, that is, all vertices of any unidentified edge belong to r distinct parts. Our next step (Step 2) is to identify every edge that contains a vertex belonging to no identified edge. Therefore, the remaining work is to identify every edge each of whose vertices belongs to some identified edge (see Step 3).

Lemma 3.2 *There are at most $r\sqrt{2m}$ parts created in Step 1 of RECONSTRUCT- r -UNIFORM-HYPERGRAPH if the hidden graph has m edges.*

Proof Suppose that there are l parts. We count in two different ways the number β of ordered pairs (v, e) such that v is a vertex and e is an edge containing v . Since the hidden graph is r -uniform and it contains at most m edges, $\beta \leq rm$. Focusing now on a vertex v , suppose that $v \in S_i$. That v is moved into S_i is due to the reason that each part S_j with $j < i$ has a vertex that shares an edge with v . Thus there are at least $\lceil \frac{i-1}{r-1} \rceil$ edges containing v . Therefore, $rm \geq \beta \geq \sum_{i=1}^l |S_i| \lceil \frac{i-1}{r-1} \rceil \geq \sum_{i=1}^l \lceil \frac{i-1}{r-1} \rceil \geq \frac{l^2}{2r}$ and the conclusion $l \leq r\sqrt{2m}$ follows. $\square$

Algorithm 1 RECONSTRUCT- r -UNIFORM-HYPERGRAPH

Step 1:

- 1: Identify an edge $e = \{v_1, \dots, v_r\}$ and create r parts: $S_1 = [n] \setminus \{v_2, \dots, v_r\}$, $S_i = \{v_i\}$ for $i = 2, 3, \dots, r$. Let $E = \{e\}$.
- 2: For every group of $r - 1$ parts (including every group of $r - 1$ parts containing S_i where S_i is a part expanded in the following process, that is, some vertex is moved into S_i) do

Let I be the index set of the $r - 1$ parts.While a query on $\bigcup_{i \in I} S_i$ is answered yes do

Identify an edge, say e , in $G[\bigcup_{i \in I} S_i]$, and include e into E . If more than one vertex of e are in the same part, then move one of them, say v , to the first available part such that v and any vertex in the part do not belong to the same identified edge (if there is no such a part, then create a new one and move v to it); repeat the process until all vertices of e belong to different parts.

End While.

End For.

- 3: Report E and all parts, say $S_1, S_2, \dots, S_l$.

Step 2:

- 1: For every r -subset I of $\{1, 2, \dots, l\}$ do

Let $U_I = \bigcup_{i \in I} S_i$, and $S_i^*(I) = \bigcup (S_i \cap e)$, where the union is running through all identified edges e in $G[U_I]$.

While there is some part $S_k, k \in I$, such that $S_k \setminus S_k^*(I) \neq \emptyset$ and a query on $U_I \setminus S_k^*(I)$ is answered yes do

Identify an edge e in $U_I \setminus S_k^*(I)$ and include e into E . For every $i \in I$, if e contains a vertex v in $S_i \setminus S_i^*(I)$, then move v from S_i to $S_i^*(I)$.

End While.

End For.

- 2: Report E .

Theorem 3.3 For the class $\mathcal{G}$ of all r -uniform hypergraphs on $[n]$, a hidden graph G drawn from $\mathcal{G}$ can be identified in at most $mr \log n + (6e)^r m^{\frac{r+1}{2}}$ edge-detecting queries where m is the number of edges of G .

Proof At the beginning, we identify an arbitrary edge and divide its vertices into r different parts $S_1, S_2, \dots, S_r$ (see Procedure 1 of Step 1). In Procedure 2 of Step 1, we sequentially identify all edges induced by the union of any $r - 1$ parts; once an edge is identified, its vertices are divided into r different parts and thus it would not be identified again. Therefore, after Step 1, the union of every $r - 1$ parts induces no edge, that is, the r vertices of each edge (identified or not) of G belongs to r different parts.

It remains to identify all hidden edges whose vertices are in r different parts. Consider the union of fixed r parts $U_I = \bigcup_{i \in I} S_i$ where $I = \{i_1, \dots, i_r\}$. According to the definition, for each $i \in I$, $S_i^*(I)$ consists of all vertices of S_i each of which belongs to some identified edge in $G[U_I]$. If $G[U_I]$ has an edge that contains a vertex in $S_i \setminus S_i^*(I)$, then e is not identified yet. The while-loop of Step 2 is to identify such edges: If $S_k \setminus S_k^*(I) \neq \emptyset$ and a query on $U_I \setminus S_k^*(I)$ is answered yes for some $k \in I$, then an edge induced by $U_I \setminus S_k^*(I)$ is to be identified and its vertices are to be moved from associated S_i to $S_i^*(I)$. Therefore, if some hidden edge in $G[U_I]$ is not identified after Step 2, then

Step 3:

1: For every r -subset I of $\{1, 2, \dots, l\}$ do

Let $U_I^* = \bigcup_{i \in I} S_i^*(I)$ and $s_I = \max_{i \in I} |S_i^*(I)|$. For every $i \in I$, partition $S_i^*(I)$ into $x_I = \lceil s_I^{\frac{r+1}{2r}} \rceil$

portions $G_{i,1}, G_{i,2}, \dots, G_{i,x_I}$ of size either $\lceil \frac{|S_i^*(I)|}{x_I} \rceil$ or $\lfloor \frac{|S_i^*(I)|}{x_I} \rfloor$. Suppose that $I = \{i_1, \dots, i_r\}$.

While there is an r -tuple of portions $(G_{i_1,k_1}, G_{i_2,k_2}, \dots, G_{i_r,k_r})$ such that a query on $\bigcup_{j=1}^r G_{i_j,k_j}$ is answered yes do

For every r -subset of $\bigcup_{j=1}^r G_{i_j,k_j}$ whose r vertices are from different portions and do not form an identified edge, make a query on the subset and include it into E if the query is answered yes.

End While.

End For.

2: Return E .

its vertices must be in $S_{i_1}^*(I), S_{i_2}^*(I), \dots, S_{i_r}^*(I)$, respectively. Step 3 is to identify the remaining unidentified edges by first partitioning $S_{i_j}^*(I)$ into parts and then identifying each edge whose vertices are in parts $S_{i_1}^*(I), S_{i_2}^*(I), \dots, S_{i_r}^*(I)$, respectively. Therefore, the correctness of RECONSTRUCT- r -UNIFORM-HYPERGRAPH is obtained.

Next, we analyze the number of queries made in Algorithm 1. Denote the number of edges identified in Step i by m_i . We can divide all queries made in Step 1 into two kinds: those spent in evaluating the condition of the while-loop and those used to identify edges in the body of the while-loop. Consider a group of $r-1$ parts. Suppose that the while-loop in Procedure 2 of Step 1 is entered m' times for this group. Then m' edges are identified and $m' + 1$ queries are spent in evaluating the condition. After Step 1, there are finally at most $r\sqrt{2m}$ parts (Lemma 3.2), and every group of $r-1$ of these parts is considered in the for-loop of Step 1. Besides, if a part S_i is expanded, then all groups of $r-1$ parts including S_i are also considered; once an edge is identified, at most $r-1$ parts are expanded and thus at most $(r-1) \cdot \binom{r\sqrt{2m}}{r-2}$ such groups (each consisting of an expanded part and $r-2$ other parts) are considered. Thus the number of queries spent in evaluating the condition of the while-loop is at most

$$(m_1 - 1) + \binom{r\sqrt{2m}}{r-1} + (m_1 - 1)(r-1) \binom{r\sqrt{2m}}{r-2}.$$

Since identifying an edge can be done in $r \lceil \log n \rceil$ queries (by Lemma 3.1), the overall number of queries made in Step 1 is at most

$$(m_1 - 1) + \binom{r\sqrt{2m}}{r-1} + (m_1 - 1)(r-1) \binom{r\sqrt{2m}}{r-2} + m_1 r \lceil \log n \rceil.$$

To simplify the formula, we need an upper bound on binomial coefficients. For any non-negative integer k , $e^k = \sum_{l=0}^{\infty} \frac{k^l}{l!} \geq \frac{k^k}{k!}$; thus $\binom{n}{k} \leq \frac{n^k}{k!} \leq \left(\frac{ne}{k}\right)^k$ for any $1 \leq k \leq n$. Therefore, the overall number of queries made in Step 1 is at most

$$\begin{aligned}
& (m_1 - 1) + \left(\frac{r\sqrt{2me}}{r-1} \right)^{r-1} + (m_1 - 1)(r-1) \left(\frac{r\sqrt{2me}}{r-2} \right)^{r-2} + m_1 r (\log n + 1) \\
& \leq (m_1 - 1) + (2\sqrt{2e})^{r-1} m^{\frac{r-1}{2}} + (3\sqrt{2e})^{r-2} (r-1) m^{\frac{r}{2}} + m_1 r (\log n + 1).
\end{aligned}$$

Similar to the analysis above, we can divide all queries made in Step 2 into two kinds: those spent in evaluating the condition of the while-loop and those used to identify edges in the body of the while-loop. For an iteration of the for-loop, if the while-loop is entered m' times, then m' edges are identified and at most $m' + r$ queries are spent in evaluating the condition of the while-loop. Therefore, the while-loop in Step 2 is evaluated at most $m_2 + r \binom{l}{r} \leq m_2 + r \binom{r\sqrt{2m}}{r}$ times and thus the overall number of queries made in Step 2 is at most

$$m_2 + (\sqrt{2e})^r m^{\frac{r}{2}} r + m_2 r (\log n + 1).$$

Finally, we analyze the number of queries made in Step 3. Let m_l be the number of edges in $G[U_l^*]$. Since all vertices in $S_i^*(I)$ belong to distinct identified edges in $G[U_l^*]$, $s_l = \max_{i \in I} |S_i^*(I)|$ (defined in Step 3) is at most m_l . We make at most $\left\lceil \frac{r+1}{s_l^{2r}} \right\rceil^r$ queries to check whether the union of each r -tuple of portions induces at least one edge; then we check all possible r -subsets of the union if the query on the union is answered yes. Thus this process takes at most $m_l \left\lceil \frac{s_l}{x_l} \right\rceil^r \leq m_l \left\lceil \frac{r-1}{s_l^{2r}} \right\rceil^r$ queries. Therefore, the number of queries made in Step 3 is at most

$$\begin{aligned}
& \sum_{l \in \binom{[l]}{r}} \left(\left\lceil \frac{r+1}{s_l^{2r}} \right\rceil^r + m_l \left\lceil \frac{r-1}{s_l^{2r}} \right\rceil^r \right) \leq \sum_{l \in \binom{[l]}{r}} \left(\left(m_l^{\frac{r+1}{2r}} + 1 \right)^r + m_l \left(m_l^{\frac{r-1}{2r}} + 1 \right)^r \right) \\
& \leq \sum_{l \in \binom{[l]}{r}} 2m_l^{\frac{r+1}{2}} \left(1 + m_l^{-\frac{r-1}{2r}} \right)^r \leq \sum_{l \in \binom{[l]}{r}} 2^{r+1} m_l^{\frac{r+1}{2}} \leq 2^{r+1} m^{\frac{r+1}{2}},
\end{aligned}$$

where l is the number of parts created in Step 1.

It is easy to observe that each edge is identified once and hence the overall cost of RECONSTRUCT- r -UNIFORM-HYPERGRAPH is at most

$$\begin{aligned}
& (m_1 + m_2)r(\log n + 1) + (m_1 + m_2) + (2\sqrt{2e})^{r-1} m^{\frac{r-1}{2}} \\
& + ((\sqrt{2e})^r r + (3\sqrt{2e})^{r-2} (r-1)) m^{\frac{r}{2}} + 2^{r+1} m^{\frac{r+1}{2}} \\
& \leq mr \log n + (6e)^r m^{\frac{r+1}{2}}.
\end{aligned}$$

Therefore, the result follows. $\square$

4 Concluding remark

We show that learning the class $\mathcal{G}$ of all r -uniform hypergraphs on $[n]$ can be done in $mr \log n + (6e)^r m^{\frac{r+1}{2}}$ queries when the hidden graph has m edges. In classical group testing, it is usually assumed that the number of positive items is much smaller than the size of the population; especially, for complex model, the number of positive complexes (corresponding to edges of the hidden graph) is much smaller than the number of all complexes (corresponding to all r -subsets of $[n]$). If the number of edges in the hidden graph is asymptotically smaller than $\log n$ and r is a constant, then our algorithm taking $O(mr \log n)$ queries to identify the hidden one is optimal, comparing to the information-theoretic lower bound provided in Theorem 2.1.

Let $\mathcal{G}'$ be the class of all $(2, r)$ -uniform hypergraphs with n vertices and m edges. Comparing to $\mathcal{G}$, the sizes of the edges of hypergraphs in $\mathcal{G}'$ are relaxed (ranging from 2 to r) but only hypergraphs of exactly m edges are included in $\mathcal{G}'$. We note that the lower bound $\Omega((2m/r)^{r/2})$ on learning $\mathcal{G}'$ and the second term of our upper bound on learning $\mathcal{G}$ asymptotically differ by a factor of $\sqrt{m}$ if r is a constant, although there is no reference comparing the difficulties of dealing with $\mathcal{G}'$ and $\mathcal{G}$.

Acknowledgements Partially supported by MOST Taiwan under the Grant MOST 104-2115-M-009-009 (H.-L. Fu) and MOST 104-2115-M-390-005-MY2 (H. Chang).

References

1. Aigner, M.: Combinatorial Search. Wiley, Hoboken (1988)
2. Alon, N., Asod, V.: Learning a hidden subgraph. *SIAM J. Discret. Math.* **18**, 697–712 (2005)
3. Alon, N., Beigel, R., Kasif, S., Rudich, S., Sudakov, B.: Learning a hidden matching. *SIAM J. Comput.* **33**, 487–501 (2004)
4. Angluin, D., Chen, J.: Learning a hidden hypergraph. *J. Mach. Learn. Res.* **7**, 2215–2236 (2006)
5. Angluin, D., Chen, J.: Learning a hidden graph using $O(\log n)$ queries per edge. *J. Comput. Syst. Sci.* **74**, 546–556 (2008)
6. Beigel, R., Alon, N., Apaydin, M.S., Fortnow, L., Kasif, S.: An optimal procedure for gap closing in whole genome shotgun sequencing. In: *Proceedings of RECOMB*. ACM Press, pp. 22–30 (2001)
7. Bouvel, M., Grebinski, V., Kucherov, G.: Combinatorial search on graphs motivated by bioinformatics applications: a brief survey. In: *WG. Lecture Notes in Computer Science*, vol. 3787, pp. 16–27 (2005)
8. Chang, H., Fu, H.L., Shih, C.H.: Learning a hidden graph. *Optim. Lett.* **8**, 2341–2348 (2014)
9. Chodorowsky, J., Moura, L.: An adaptive algorithm for group testing for complexes. *Theor. Comp. Sci.* **592**, 1–8 (2015)
10. Grebinski, V., Kucherov, G.: Reconstructing a Hamiltonian cycle by querying the graph: application to DNA physical mapping. *Discret. Appl. Math.* **88**, 147–165 (1998)
11. Grebinski, V., Kucherov, G.: Optimal reconstruction of graphs under the additive model. *Algorithmica* **28**, 104–124 (2000)
12. Lappe, M., Holm, L.: Unraveling protein interaction networks with near-optimal efficiency. *Nat. Biotechnol.* **22**, 98–103 (2003)
13. Li, Y., Thai, M., Liu, Z., Wu, W.: Protein-to-protein interactions and group testing in bipartite graphs. *Int. J. Bioinform. Res. Appl.* **1**, 414–419 (2005)
14. Sorokin, A., Lapidus, A., Capuano, V., Galleron, N., Pujic, P., Ehrlich, S.D.: A new approach using multiplex long accurate PCR and yeast artificial chromosomes for bacterial chromosome mapping and Sequencing. *Genome Res.* **6**, 448–453 (1996)
15. Torney, D.C.: Sets pooling designs. *Ann. Comb.* **3**, 95–101 (1999)