國 立 交 通 大 學

應用數學系

碩 士 論 文

使用循序式演算法尋找隱藏圖

Learning a Hidden Graph with Adaptive Algorithm

研 究 生：施智懷

指導教授：傅恆霖　教授

中 華 民 國 九 十 七 年 七 月

使用循序式演算法尋找隱藏圖

Learning a Hidden Graph with Adaptive Algorithm

研 究 生：施智懷　　Student：Chie-Huai Shih

指導教授：傅恆霖　　Advisor：Hung-Lin Fu

國 立 交 通 大 學

應 用 數 學 系

碩 士 論 文

A Thesis

Submitted to Department of Applied Mathematics
College of Science
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master
in
Applied Mathematics
July 2008
Hsinchu, Taiwan, Republic of China

中 華 民 國 九 十 七 年 七 月

使用循序式演算法尋找隱藏圖

研究生：施智懷　　指導老師：傅恆霖　教授

國 立 交 通 大 學

應 用 數 學 系

摘　要

我們考慮如何只使用邊偵測問題(edge-detecting query)的情況下去尋找隱藏圖，邊偵測問題可以回答任何點集合內是否包含至少一條邊。Grebinski 以及 kucherow[5]提供一個確定性演算法去尋找隱藏的漢米爾頓圈(Hamiltonian cycle)，使用最多$O(n\log n)$個問題；Beigel 等人[4]提出一個確定性演算法去解決配對圖(bipartite)，使用八個平行的問題群，總共$O(n\log n)$個問題，尋找隱藏的漢米爾頓圈或是配對圖可以應用到基因排序計劃。Angluin 以及陳[2]針對一般的隱藏圖使用最多$12m\log n$個問題。在這個論文中，我們提供一個循序式演算法解決一般的隱藏圖，當此圖點數為n、邊數為m時，最多花費$(2\log n+9)m$個問題。

Learning a Hidden Graph with Adaptive Algorithm

Student: Chie-Huai Shih Advisor: Hung-Lin Fu

Department of Applied Mathematics
National Chiao Tung University

Abstract

We consider the problem of learning a hidden graph using edge-detecting queries in a model where the only allowed operation is to query whether a set of vertices induces an edge of the hidden graph or not. Grebinski and Kucherov [5] give a deterministic adaptive algorithm for learning Hamiltonian cycles using $O(n \log n)$ queries. Beigel *et al.*[4] describe an 8-round deterministic algorithm for learning matchings using $O(n \log n)$ queries, which has direct application in genome sequencing projects. Angluin and Chen [2] use at most $12m \log n$ queries in their algorithm for learning a general graph. In this thesis we present an adaptive algorithm that learns a general graph with n vertices and m edges using at most $(2 \log n + 9)m$ queries.

誌　　謝

當碩士論文寫到此頁，則代表我的碩士生涯要正式落幕了，兩年的研究所時光看似漫長，實則如過眼雲煙轉瞬即逝。雖然這不是一部完美的論文，但這部論文的完成，要感謝的真的人很多，僅以此文表達我的誠摯謝意。

首先要感謝我的指導老師：傅恆霖教授。他在 meeting 或者上課時，總能把生硬的數學轉化成讓學生能輕易吸收的語言，還常常鼓勵學生，使我能繼續堅持數學這條路。傅教授從大四開始就提供我選課方面的建議，這些課程對我在寫此篇論文上幫助頗大。傅老師不只是一位優秀的學者，他在各方面的執著與毅力，讓我十分的佩服。

還要感謝陳邱媛教授，上過陳老師的圖論課，讓我開始對圖論產生了興趣，進而選擇加入組合組這個大家庭。還要感謝組合組的每位教授，我很幸運的修過每位教授所開的課程，在寫論文的過程中，常常翻閱上課筆記或講義，尋找靈感，教授們所教導的知識幫助我處理許多難題。

另外還要感謝傅老師的所有學生：賓賓、志銘、嘉芬、robin、貓頭、惠蘭、敏筠、若宇、政軒、奇聰、逸軒、舜婷，在我報告時提供許多寶貴的意見，尤其是陳宏賓學長跟惠蘭學姐，假如沒有學長姐的幫忙，今天我可能還在煩腦是否要邁向碩三的生活，這個主題是靠賓賓學長提供的，惠蘭學姐也提供了相關論文的結果。另外還要感謝所有跟我同屆的組合組同學：鈺傑、威雄、皜文、政緯、佩純、偉帆、雅榕、志文、子鴻。有你們的參與，讓我研究所生涯多彩多姿！

最後感謝我的家人，爸爸媽媽在經濟上支持我，也鼓勵我繼續朝著理想前進，你們是我能畢業的重要支柱。

目錄

Chapter 1.
Introduction

1.1. Motivation

This paper is motivated by an important problem in computational molecular biology that arises in whole-genome shotgun sequencing. Shotgun sequencing is a throughput technique resulting in the sequencing of a large number of bacterial genomes, mouse genomes and the celebrated human genomes. In all such projects, we are left with a collection of contigs that for special reasons cannot be assembled with general assembly algorithms. For completeness of sequencing, the contigs must be oriented and the gaps between them must be sequenced using other methods. When the number of gaps is small, the technique "polymerase chain reaction (PCR)" initiates a set of bidirectional molecular walks along the gaps in the sequence; these walks are facilitated by PCR and primers are used.

Now, if we are left with n (small) contigs, then the exhaustive PCR technique tests all possible $\binom{2n}{2}$ pairs of $2n$ primers by placing two primers per tube. On the other hand, if the number of gaps is large, instead of testing all pairs, primers are pooled using more (than two) primers per tube; this is the so-called multiplex PCR technique. So, our goal is to provide optimal strategies for pooling the primers to minimize the number of biological experiments needed in the gap-closing process.

Therefore, the problem of gap-closing can be stated more generally as follows. We are given a set of chemicals, a guarantee that each chemical react with at most one of the others (because only primers on opposite sides of the same gap create a reaction), and an experimental mechanism to determine whether a reaction occurs when they are combined in a tube. Our goal is to determine which pairs of chemicals react with each other by using a minimum number of experiments.

1.2. Mathematical formulation

Due to the nature of primers reaction, our problem can be modeled as the problem of learning a hidden graph given vertex set and an allowed query operation. The problem of learning a hidden graph is the following. Imagine that there is a graph $G = (V, E)$ whose vertices are known to us and whose edges are not. We wish to identify all the edges of G by asking some edge-detecting queries of the form

$$Q_G(S): \text{does } S \text{ include at least one edge of } G?$$

Here, S is a subset of V. Therefore, the problem is to find an algorithm to reconstruct the hidden graph by using as few queries as possible. Distinctly, time is also very important, so we may want to parallelize the experiments as fewer rounds as possible.

An important aspect of an algorithm in this model is the number of parallel rounds. An algorithm is non-adaptive if the whole of queries makes chosen before the answer to earlier queries, in other words, a non-adaptive algorithm is a 1-round algorithm. An algorithm is adaptive if the queries may conduct one by one.

So far, there are several related works which have been done. In 1997, Grebinski and Kucherov [5] considered the problem of finding a Hamilton cycle motivated by the study of DNA physical mapping. They obtain a deterministic adaptive $O(n \log n)$ algorithm. (All the logarithms we use throughout this paper will be in base 2.) Later, in 2001, Beigel et al. [4] describe an 8-round deterministic algorithm for learning matchings using $O(n \log n)$ queries, which has direct application in genome sequencing projects. For randomized algorithms, a 1-round Monte Carlo algorithm for learning matchings was given by Alon et al. [1], which succeeds with high probability. Quite recently, a more precise estimation of the number of queries on adaptive model of learning a general graph was obtained by Angluin and Chen [2]. They use at most $12m \log n$ queries in their algorithm for learning a general graph where m is the number of edges and n is the number of vertices. Also, they prove

that the asymptotic lower bound of the number of queries is $\Omega(n \log n)$ which achieves the asymptotic lower bound in this model.

In this thesis, we further improve the upper bound of the number of queries used in this model when adaptive algorithm is utilized to learn a hidden general graph. Mainly, we prove that our adaptive algorithm of learning a hidden graph with m edges defined on a set of n vertices uses at most $(2 \log n + 9)m$ queries. As a consequence, using the algorithm, we can also reconstruct a Hamilton cycle or a matching using at most $(2 \log n + 9)m$ queries which achieves the asymptotic lower bound.

Chapter 2. Preliminaries

2.1. Notations

2.1.1. Notations of Graphs

In graph theory, a simple graph is a pair $G = (V, E)$ where V is the set of vertices and E is the set of edges. An subgraph H of a graph G is said to be induced if, for any pair of vertices u and v of H, uv is an edge of H if and only if uv is an edge of G. In other words, H is an induced subgraph of G if it has all the edges that appear in G over the same vertex set. If the vertex set of H is the subset S of $V(G)$, then H can be written as $G[S]$ and is said to be induced by S.

An undirected graph is a graph which every edge is undirected. A set $I \subseteq V$ is an independent set of G if it contains no edge of G. A bipartite graph is a simple graph in which the vertex set can be decomposed into two independent sets.

A matching in a graph is a set of edges that do not share vertices. A maximal matching is a matching M of a graph G with the property that if any edge not in M is added to M, it is no longer a matching.

A Hamiltonian cycle (or Hamiltonian circuit) is a cycle in an undirected graph which visits each vertex exactly once and also returns to the starting vertex. The Hamiltonian cycle on n vertices has n vertices and n edges.

A complete graph is a simple graph in which every pair of distinct vertices is connected by an edge. The complete graph on n vertices has n vertices and $\binom{n}{2}$ edges, and is denoted by K_n.

A tree is a graph in which any two vertices are connected by exactly one path. Alternatively, any connected graph with no cycles is a tree.

2.1.2. Notations of computer science

In computer science, a computation tree is a tree of nodes and edges. Each node in the tree represents a single computational state, while each edge represents a transition to the next possible computation. The length of the path from the root to a given node is the depth of the node.

A binary tree is a tree data structure in which each node has at most two children. Typically the child nodes are called left and right. The root node of a tree is the node with no parents. There is at most one root node in a rooted tree. Nodes at the bottommost level of the tree are called leaf nodes. Since they are at the bottommost level, they do not have any children. An internal node or inner node is any node of a tree that has child nodes and is thus not a leaf node.

In computational complexity theory, big O notation is often used to describe how the size of the input data affects an algorithm's usage of computational resources (usually running time or memory). The symbol O is used to describe an asymptotic upper bound for the magnitude of a function in terms of another, usually simpler, function. There are also other symbols o, Ω, ω, and θ for various other upper, lower, and tight bounds, these are useful in the analysis of the complexity of algorithms.

2.2. Models

There are four types of queries which lead to four different mathematical models in learning a hidden graph G.

(1) *Multi-vertex model.* For a set of vertices $\{a_1, a_2, \dots, a_r\}$, ask whether $K_{\{a_1,a_2,\dots,a_r\}} \cap G = G[\{a_1, a_2, \dots, a_r\}]$ is non-empty, where $K_{\{a_1,a_2,\dots,a_r\}}$ is the complete graph on the set of vertices $\{a_1, a_2, \dots, a_r\}$.

(2) *Quantitative multi-vertex model.* For a set of vertices $\{a_1, a_2, \dots, a_r\}$, ask what the number of edges in $K_{\{a_1,a_2,\dots,a_r\}} \cap G$ is.

(3) *k-vertex model.* Assume that a k is predefined. For a set of vertices $\{a_1, a_2, \dots, a_r\}$, where $r \leq k$, ask whether $K_{\{a_1,a_2,\dots,a_r\}} \cap G$ is non-empty.

(4) *Quantitative k-vertex model.* For a set of vertices $\{a_1, a_2, \dots, a_r\}$, where $r \leq k$, what is the number of edges in $K_{\{a_1,a_2,\dots,a_r\}} \cap G$ is.

The model used in this paper will be (1) *Multi-vertex model.*

2.3. Lower bound

Theorem 2.3.1. [1] *For any $0 < \varepsilon < 2$, $\varepsilon m(\log n - 2)$ edge-detecting queries are required to identify a graph G drawn from the class of all graphs with n vertices and $m = n^{2-\varepsilon}$ edges.*

Proof. There are

$$\binom{\binom{n}{2}}{m} \geq \left(\frac{\binom{n}{2}}{m}\right)^m$$

graphs that have m edges. For any algorithm, its computation tree is a binary tree and has at least $\left(\frac{\binom{n}{2}}{m}\right)^m$ leaves, so the depth at least

$$\log\left(\frac{\binom{n}{2}}{m}\right)^m = m\log\left(\frac{\binom{n}{2}}{m}\right) = m\log\left(\frac{\binom{n}{2}}{n^{2-\varepsilon}}\right) \geq m\log\left(\frac{n^\varepsilon}{4}\right) = \varepsilon m(\log n - 2).$$

Therefore, the lower bound implies at least $\varepsilon m(\log n - 2)$ queries in the worst case. □

2.4. Two Algorithms

In order to prove our main result, we shall need two adaptive algorithms which have been done earlier. Both of them are by Angluin and Chen [1, 2]. The first one identifies an arbitrary edge in a non-empty hidden graph using $2\log n$ queries where n is the size of the vertex set V. For convenience, we will denote the algorithm by **Algorithm A (V).** And the second one finds the edges between two known independent sets S_1 and S_2. We will be using this algorithm in a special case either $|S_1| = 1$ or $|S_2| = 1$, we denote it by **Algorithm B (v, I)** where v is a

vertex and I is an independent set not contains $\boldsymbol{v}$. Note here that this algorithm uses $\log n + 1$ rounds with $2s\log|I| + 1$ queries where s is the number of edges between v and I. Moreover, if the answer of $Q(I \cup \{v\})$ is known, then we need at most $2s\log|I|$ queries to identify all the edges between v and I.

For completeness, we include them with slight modification.

- **Algorithm A (V)**

Part 1. ***FIND_ONE_VERTEX(V)***

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. Let v be the unique element in A.
11. **Output** $(v, S \backslash \{v\})$

Part 2. ***FIND_ONE_EDGE(V)***

1. $(v, S) \leftarrow$*FIND_ONE_VERTEX(V)*
2. **while** $|S| > 1$ do
3. Divide S arbitrarily into S_0 and S_1, such that $|S_0| = \lceil |S|/2 \rceil, |S_1| = \lfloor |S|/2 \rfloor$.
4. **if** $Q(S_0 \cup \{v\}) = 1$ **then**
5. $S \leftarrow S_0$
6. **else**
7. $S \leftarrow S_1$
8. **end if**

9. **end while**

10. Let w is the unique element in S.

11. **Output** (vw)

Lemma 2.4.1. ***FIND_ONE_VERTEX(V)*** *finds one relevant vertex in a vertex set* V *using at most* $\log n$ *edge-detecting queries.*

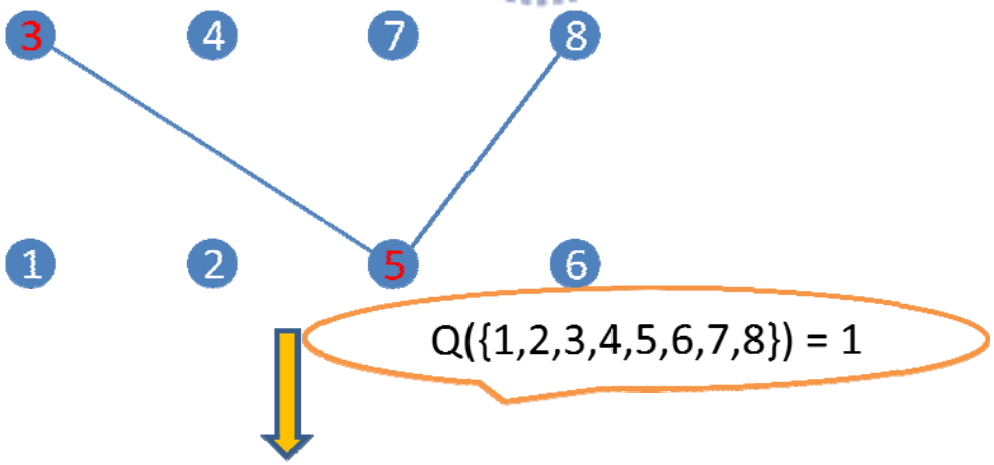
Proof. Since the size of A halves at each iteration, after at most $\log n$ iterations, A has exactly one relevant vertex. The algorithm takes at most $\log n$ edge-detecting queries in total, as it makes one query in each iteration. □

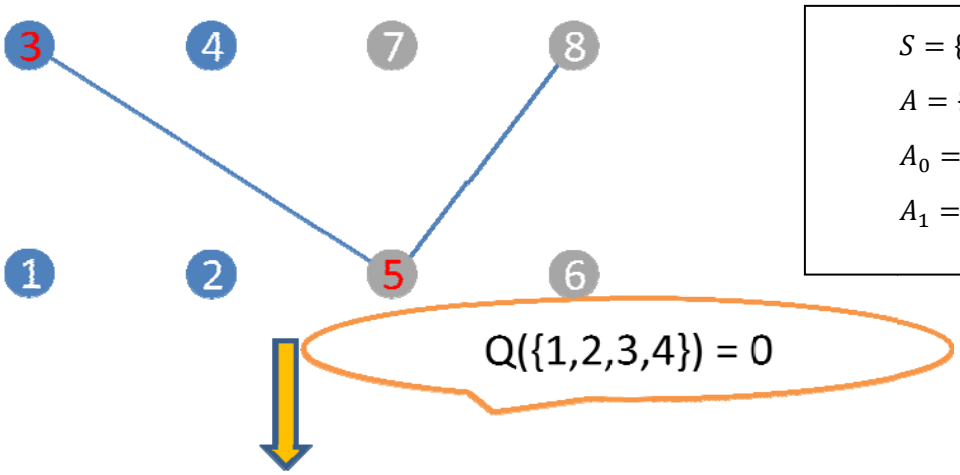
Lemma 2.4.2. Algorithm A (*V*) *finds one edge uses at most* $2\log n$ *edge-detecting queries where* n *is the size of* V.

Proof. By **Lemma 2.4.1.**, ***FIND_ONE_VERTEX(V)*** using at most $\log n$ queries. Similarly, the size of S halves at each iteration in ***FIND_ONE_EDGE(V)***, then it only uses $\log n + \log n = 2\log n$ to find an edge. □

Example 2.4.3.

PART 1.



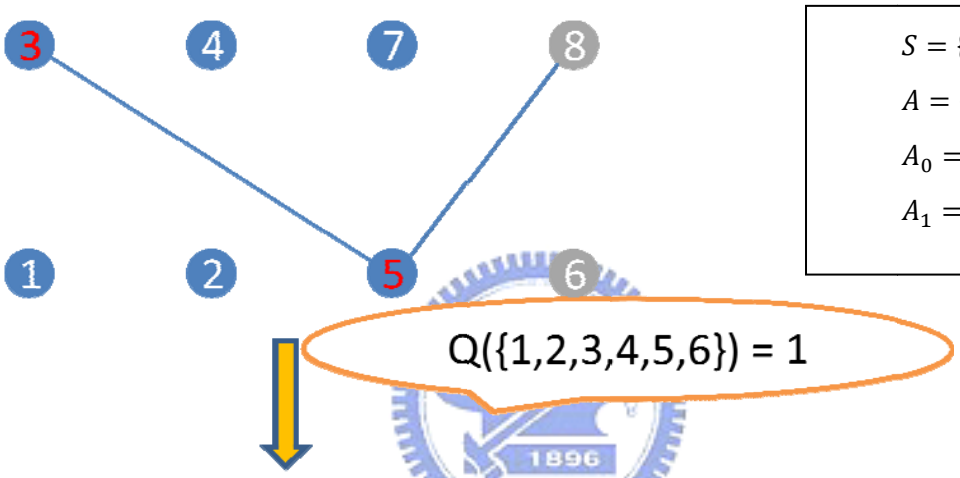


$S = \{1,2,3,4,5,6,7,8\}$

$A = \{1,2,3,4,5,6,7,8\}$

$A_0 = \{5,6,7,8\}$

$A_1 = \{1,2,3,4\}$

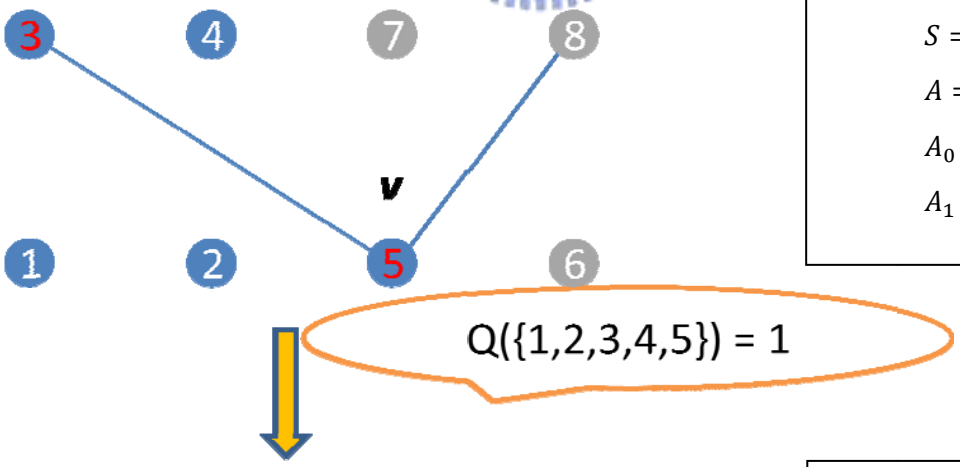


$S = \{1,2,3,4,5,6,7,8\}$

$A = \{5,6,7,8\}$

$A_0 = \{7,8\}$

$A_1 = \{5,6\}$



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

$A = \{5,6\}$

$A_0 = \{6\}$

$A_1 = \{5\}$

$v = 5, S \setminus \{v\} = \{1,2,3,4\}$

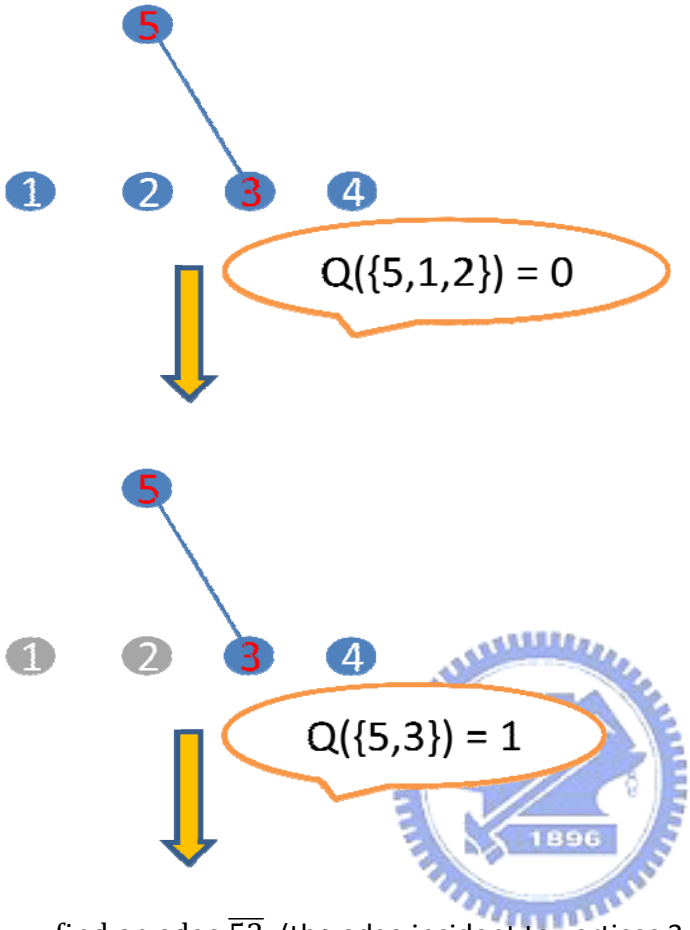
$S = \{1,2,3,4,5\}$

$A = \{5\}$

$|A| = 1$

$v = 5$

PART 2.



find an edge $\overline{53}$ (the edge incident to vertices 3 and 5)

- **Algorithm B (v , I)**

VERTEX_INDEPENDENT_SET(v , I)

1. **while** $|I| > 1$ **do**
2. Divide I arbitrarily into I_0 and I_1, such that $|I_0| = \lceil |I|/2 \rceil$, $|I_1| = \lfloor |I|/2 \rfloor$.
3. **if** $Q(I_0 \cup \{v\}) = 1$ **then**
4. $I \leftarrow I_0$
5. $E \leftarrow E \cup$ *VERTEX_INDEPENDENT_SET(v, I)*
6. **end if**
7. **if** $Q(I_1 \cup \{v\}) = 1$ **then**

8. $I \leftarrow I_1$

9. $E \leftarrow E \cup$ *VERTEX_INDEPENDENT_SET*(v, I).

10. **end if**

11. **end while**

12. **if** $|I| = 1$ **do**

13. Let w is the unique element in I.

14. $E \leftarrow \{vw\}$

15. **end if**

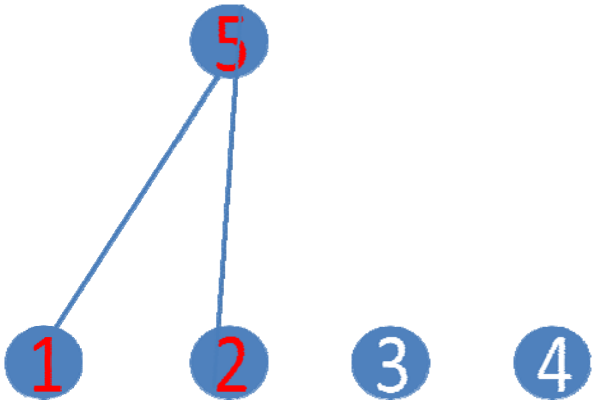
16. **Output** (E)

Lemma 2.4.4. Algorithm B $(\boldsymbol{v}, \boldsymbol{I})$ *identifies edges between* S_1 *and* S_2 *using no more than* $2s \log n + 1$ *edge-detecting queries where* n *is the size of* I *and* s *is the number of edges between* $\boldsymbol{v}$ *and* I.

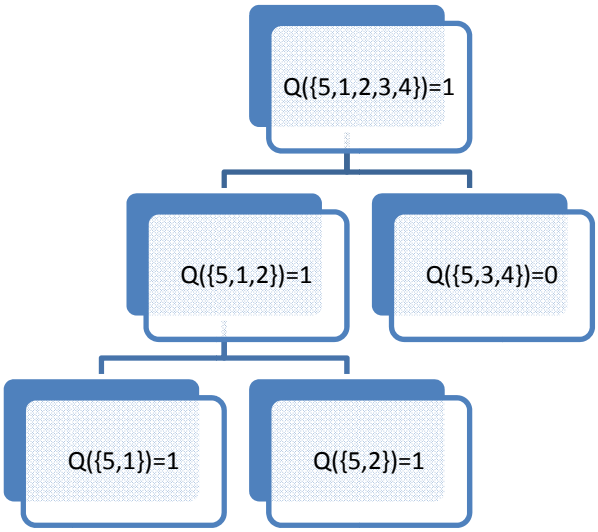
Proof. If we consider the computation tree for this algorithm, the maximum depth does not exceed $\lceil \log |I| \rceil$. Each edge is corresponding to an leaf, and at each internal node of the path from the root to the leaf, the algorithm asks at most 2 queries and the length of path at most $\lceil \log |I| \rceil$. Therefore, the algorithm asks at most $2s \log n + 1$ queries. □

Example 2.4.5.

G:



Computation tree



Find edges $\overline{51}$ and $\overline{52}$.

Chapter 3.
Main Result

3.1 Algorithms

We start with presenting our algorithms.

Note here that if there are edges between two independent sets, we may find all of the edges by using **Algorithm B ($\boldsymbol{v}$, $\boldsymbol{I}$).** The following algorithm is using a maximal matching to partition the vertices of G into several bipartite graphs and an independent set. In other words, we provide an algorithm to partition the vertex set into several independent sets. Our first objective is to minimize the number of independent sets.

Algorithm 1. MAXIMAL_MATCHING($\boldsymbol{V}$)

1. $V' \leftarrow V, i \leftarrow 1, M \leftarrow \emptyset$
2. **while** $Q(V') = 1$ **do**
3. $\mathrm{x_i y_i} \leftarrow$*FIND_ONE_VERTEX(V)*
4. $M \leftarrow M \cup \{x_i y_i\}, V' \leftarrow V' \backslash \{x_i y_i\}, i \leftarrow i + 1$
5. **end while**
6. **Output** (i)

Algorithm 2. PARTITION_OF_VERTEX_SET($\boldsymbol{V}$)

1. $V^{"} \leftarrow V$
2. $k \leftarrow$MAXIMAL_MATCHING($\boldsymbol{V}$)
3. $X_i = \{x_i\}, Y_i = \{y_i\}\ \forall 1 \leq i \leq k$
4. **for** $i = 1, i \leq \mathrm{k}, i++$ **do**
5. **for** $j = 1, j < i, j++$ **do**
6. **if** $X_j = \emptyset$ **then**
7. $X_j \leftarrow x_i, Y_j \leftarrow y_i, X_i \leftarrow \emptyset, Y_i \leftarrow \emptyset$

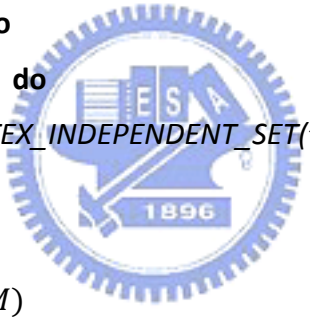
8. **break**(1-loop)
9. **else**
10. Make queries on $X_j \cup \{x_i\}, Y_j \cup \{y_i\}, X_j \cup \{y_i\}, Y_j \cup \{x_i\}$
11. **if** $Q(X_j \cup \{x_i\}) = Q(Y_j \cup \{y_i\}) = 0$ **then**
12. $X_j \leftarrow X_j \cup \{x_i\}, Y_j \leftarrow Y_j \cup \{y_i\}, X_i \leftarrow \emptyset, Y_i \leftarrow \emptyset$
13. **break**(1-loop)
14. **else if** $Q(X_j \cup \{y_i\}) = Q(Y_j \cup \{x_i\}) = 0$ **then**
15. $X_j \leftarrow X_j \cup \{y_i\}, Y_j \leftarrow Y_j \cup \{x_i\}, X_i \leftarrow \emptyset, Y_i \leftarrow \emptyset$
16. **break**(1-loop)
17. **end if**
18. **end if**
19. **end for**
20. **end for**
21. $\exists! p \ s.t. \ X_P \neq \emptyset, X_{p+1} = \emptyset$
22. **Output** (p)

Remark: After implementing **Algorithm 2**, it is easily seen that the vertex sets X_j, Y_j *and* X_0 for $i = 1,2,\dots,p$, are independent sets in G where $X_0 = V \setminus \cup_{i=1}^{p}(X_i \cup Y_i) = V \setminus \cup_{i=1}^{p}\{x_i \cup y_i\}$. Now, if we can identify all the edges between two different independent sets, then the graph G is reconstructed.

Algorithm 3. HIDDEN_GRAPH(V)

1. $p \leftarrow$PARTITION_OF_VERTEX_SET(V)
2. $X_0 = V \setminus \cup_{i=1}^{p}(X_i \cup Y_i)$
3. $E_1 = \emptyset$
4. **for** $i = 1, i \leq p, i++$ **do**
5. **for** $v \in X_i$ **do**
6. $\exists! w$ s.t. $wv \in M$
7. $E_1 = E_1 \cup$ *VERTEX_INDEPENDENT_SET($v, Y_i \setminus \{w\}$)*
8. **end for**
9. **end for**

10. $E_2 = \emptyset$
11. **for** $i = 2, i \leq p, i++$ **do**
12. **for** $v \in X_i \cup Y_i$ **do**
13. **for** $j = 1, j < i, j++$ **do**
14. **if** $Q(X_j \cup \{v\}) = 1$ **then**
15. $E_2 = E_2 \cup$ *VERTEX_INDEPENDENT_SET*(v, X_j)
16. **end if**
17. **if** $Q(Y_j \cup \{v\}) = 1$ **then**
18. $E_2 = E_2 \cup$ *VERTEX_INDEPENDENT_SET*(v, Y_j)
19. **end if**
20. **end for**
21. **end for**
22. **end for**
23. $E_3 = \emptyset$
24. **for** $i = 1, i \leq p, i++$ **do**
25. **for every** $v \in X_i \cup Y_i$ **do**
26. $E_3 = E_3 \cup$ *VERTEX_INDEPENDENT_SET*(v, X_0)
27. **end for**
28. **end for**
29. **Output** $(E_1 \cup E_2 \cup E_3 \cup M)$



3.2. Analysis

3.2.1. Query complexity

The following two observations are essential in determining the complexity of our algorithms. First, (1) for every $1 \leq i \leq k$ there exists a unique l_i such that $\{x_i, y_i\} \subseteq X_{l_i} \cup Y_{l_i}$. And (2) there exist at least two edges in E_2 between two vertex sets $X_i \cup Y_i$ and $X_j \cup Y_j$ for every $i \neq j$.

Also, for convenience of counting the number of queries, we let $|M| = m_0, |E_1| = m_1, |E_2| = m_2$ and $|E_3| = m_3$ (i.e. $\mathrm{m} = m_0 + m_1 + m_2 + m_3$).

In **Algorithm 1**, since we need one query to check whether the vertex set V is independent or not before using the algorithm **Algorithm A (*V*)** and we use **Algorithm A (*V*)** to identify a maximal matching with size m_0, the complexity is equal to the sum of $m_0 + 1$ and $(2\log n)m_0$.

In **Algorithm 2**, it suffices to consider the number of queries in the 2nd and 10th line. By observations (1) and (2) mentioned above, we know the 10th line will be repeated at most $\lfloor m_2/2 \rfloor + m_0$ times and thus in total $(5 + 2\log n)m_0 + 1 + 2m_2$ queries.

In **Algorithm 3**, since we use the **Algorithm B (*v* , *I*)** to find the edge sets E_1, E_2 and E_3, the 7th line was repeated m_0 times where m_0 is the size of the maximal matching M. In other words, **Algorithm B (*v* , *I*)** was called m_0 times in this line, the number of queries in this line is therefore $m_0 + (2\log n)m_1$. In the 15th and 18th lines, we use the **Algorithm B (*v* , *I*)** to find the edge set E_2 and we already had some information before calling the **Algorithm B (*v* , *I*)**, the number of queries in 15th and 18th lines is $(2\log n)m_2$. Finally, in the 26th lines, we identify the edge set E_3 and **Algorithm B (*v* , *I*)** was called $2m_0$ times, so the number of queries is $2m_0 + (2\log n)m_3$. The following tables show the above facts.

Algorithm 1.

Line	Number of queries
2	$m_0 + 1$
3	$(2\log n)m_0$
Total	$(1 + 2\log n)m_0 + 1$

Algorithm 2.

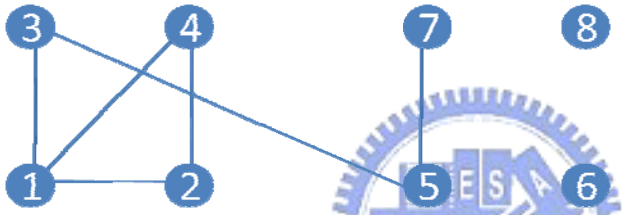
Line	Number of queries
2	$(1 + 2\log n)m_0 + 1$
10	$4m_0 + 2m_2$
Total	$(5 + 2\log n)m_0 + 1 + 2m_2$

Algorithm 3.

Line	Number of queries
1	$(5+2\log n)m_0+1+2m_2$
7	$m_0+(2\log n)m_1$
14+17	0 (all of queries be answered in algorithm 2. , 10th line)
15+18	$(2\log n)m_2$
26	$2m_0+(2\log n)m_3$
Total	$(m_0+m_1+m_2+m_3)2\log n+8m_0+1\leq m(2\log n+9)$

Example 3.1.1.

$\boldsymbol{G}$:



Algorithm 1.

(1.) *FIND_ONE_VERTEX*($\{1,2,3,4,5,6,7,8\}$) $\Rightarrow$ *find edge* $\overline{\mathbf{13}}$.

(2.) *FIND_ONE_VERTEX*($\{2,4,5,6,7,8\}$) $\Rightarrow$ *find edge* $\overline{\mathbf{24}}$.

(3.) *FIND_ONE_VERTEX*($\{5,6,7,8\}$) $\Rightarrow$ *find edge* $\overline{\mathbf{57}}$.

$M=\{\overline{13},\overline{24},\overline{57}\}$

Algorithm 2.

(1.) $\mathrm{Q}(\{1,2\})=1,\mathrm{Q}(\{3,4\})=1,\mathrm{Q}(\{1,4\})=1,\mathrm{Q}(\{3,2\})=0$

(2.) $\mathrm{Q}(\{3,7\})=0,\mathrm{Q}(\{1,5\})=0,\mathrm{Q}(\{3,5\})=1,\mathrm{Q}(\{1,7\})=0$

$X_1=\{1,5\},\ Y_1=\{3,7\},X_2=\{2\},\ Y_1=\{4\}$

Algorithm 3.

$X_0=\{6,8\},Y_1=\{3,7\},X_1=\{1,5\},\ Y_2=\{4\},X_2=\{2\}$

(1.) *VERTEX_INDEPENDENT_SET*(3 , X_0)

(2.) *VERTEX_INDEPENDENT_SET(*7 *,*X_0*)*

(3.) *VERTEX_INDEPENDENT_SET(*1 *,*X_0*)*

VERTEX_INDEPENDENT_SET(1 *,*$Y_1\backslash\{3\}$*)*

(4.) *VERTEX_INDEPENDENT_SET(*5 *,*X_0*)*

VERTEX_INDEPENDENT_SET(5 *,*$Y_1\backslash\{7\}$*)* $\Rightarrow$ *find edge* $\overline{\mathbf{53}}$

(5.) *VERTEX_INDEPENDENT_SET(*4 *,*X_0*)*

VERTEX_INDEPENDENT_SET(4 *,*Y_1*)* $\Rightarrow$ *find edge* $\overline{\mathbf{41}}$

VERTEX_INDEPENDENT_SET(4 *,*X_1*)*

(6.) *VERTEX_INDEPENDENT_SET(*2 *,*X_0*)*

VERTEX_INDEPENDENT_SET(2 *,*Y_1*)*

VERTEX_INDEPENDENT_SET(2 *,*X_1*)* $\Rightarrow$ *find edge* $\overline{\mathbf{21}}$

NOTE. $Y_2\backslash\{4\} = \emptyset$

COMPLETED……

3.2.3. Worst case of this algorithm

We prove that our adaptive algorithm of learning a hidden graph with m edges defined on a set of n vertices uses at most $(2\log n + 9)m$ queries. This algorithm is efficient to reconstruct a graph when the number of edges is small. If the number of edges of a hidden graph is $\binom{n}{2}$, in above analysis, the upper bound is $(2\log n + 9)\binom{n}{2}$, but the trivial algorithm only uses $\binom{n}{2}$ queries to obtain a hidden graph. The following will give another concept to analysis the upper bound and the lower bound of this algorithm.

Lemma 3.2.1. *In Multi-vertex model,* $\binom{n}{2}$ *edge-detecting queries are required to identify a graph* G *drawn from the class of all graphs with* n *vertices and* $m = \binom{n}{2}$ *edges*.

Proof. It is clearly, every edge e in G must be identified by an edge-detecting queries $\mathrm{Q}(V^*)$ where $V^* \subseteq V$ and $(K_{V^*}\backslash\{e\}) \cap E(G) = \phi$. □

Lemma 3.2.2. ***Algorithm B*** **(**v **,** I**)** *identifies edges between* v *and* I *using no more than* $2|I| - 1$ *edge-detecting queries.*

Proof. For convenience, we may assume the size of I is 2 to the power. We consider the computation tree for this algorithm**.** This computation tree is a full binary tree, the number of leaves in this tree is equal the size of I and the number of nodes is $2|I| - 1$. In general, we may delete some pair of leaves from a full binary tree to obtain its computation tree. □

Theorem 3.2.3. *Our adaptive algorithm to learn a general graph* $G = (V, E)$ *does not exceed* $1 + \frac{n}{2} + n\log n + 3\binom{n}{2}$ *where* n *is the size of* V.

Proof. In **Algorithm 1**, since the size of maximal matching does not exceed $n/2$, the worst case is the sum of $n/2 + 1$ and $n\log n$.

In **Algorithm 2**, the 10th line does not repeat more than $\binom{|M|}{2}$ times where M is the maximal matching of G. So the worst case in this line does not exceed $\binom{n}{2}$ queries.

After **Algorithm 1** and **Algorithm 2**, the vertex set V will be partition into several vertex sets X_0, X_i, Y_i where $1 \le i \le p$. Assign indices $1, 2, \ldots, n$ to the vertices of V according to some restriction as following. First, (1) for every $1 \le i \le p$, if $v \in Y_i$ and $w \in X_i$, then the index of v be smaller than the index of w. And (2) if $v \in X_i$ and $w \in X_j$, then the index of v be smaller than the index of w when $0 \le i < j \le p$.

In **Algorithm 3**, we use the **Algorithm B** (v, I) to find all the edges and **Lemma 3.2.2.** gives a bound for **Algorithm B** (v, I). The number of queries does not exceed double of the sum of each size I when calling the **Algorithm B** (v, I). Consider the vertex v and the independent set I in **Algorithm B** (v, I), in above paragraph, we know that the index of v greater the index of every vertex in I, then the number of queries of **Algorithm 3** is at most $2\binom{n}{2}$. Therefore, we can reconstruct a hidden graph using no more than $1 + \frac{n}{2} + n\log n + 3\binom{n}{2}$ queries. □

3.2.3. Number of parallel non-adaptive rounds

Because calling the **Algorithm B (*v* , *I)*** in **Algorithm 3** can be parallelized to find all edges in $(2\log n + 1)$ rounds, this saves the rounds used in reconstructing the hidden graph sharply. But we don't have a good idea to reduce the rounds of **Algorithm 1** and **Algorithm 2**.

Concluding remarks

In this paper, we have presented a new adaptive algorithm to find a hidden graph. It is not difficult to see that the number of queries used in our algorithm is around the lower bound which we expect to achieve especially when the size of the graph is far less than the order of the hidden graph. Here are a couple of works which we would like to accomplish in the near future:

1. Reduce the rounds of **Algorithm 1** (i.e., obtain an efficient algorithm to find a maximal matching).
2. Learning a hidden graph in *Quantitative k-multi-vertex model*.

References

[1] N. Alon, R. Beigel, S. Kasif, S. Rudich,and B. Sudakov, Learning a hidden matching, The 43rd Annual IEEE Symposium on Foundations of Computer Scienc*e*, 197–206, 2002.

[2] D. Angluin and J. Chen. Learning a hidden graph using O(log n) queries per edge. Manuscript, 2006.

[3] D. Angluin and J. Chen. Learning a hidden hypergraph, J. of Machine Learning Research 7, 2215-2236, 2007.

[4] R. Beigel, N. Alon, S. Kasif, M. S. Apaydin and L. Fortnow., An optimal procedure for gap closing in whole genome shotgun sequencing, In *RECOMB*, 22–30, 2001.

[5] V. Grebinski and G. Kucherov, Optimal query bounds for reconstructing a Hamiltonian cycle in complete graphs, In fifth Israel symposium on the Theory of Computing Syste*ms*, 166-173, 1997.

[6] V. Grebinski and G. Kucherov.,Reconstructing a Hamiltonian cycle by querying the graph: Application to DNA physical mapping. Discrete Applied Math*.,* 88(1-3): 147–165, 1998.