

# Optimally Reconstructing Weighted Graphs Using Queries

(Extended Abstract)

Hanna Mazzawi\*

Technion - Israel Institute of Technology

## Abstract

In this paper, we consider the problem of reconstructing a hidden graph with  $m$  edges using additive queries. Given a graph  $G = (V, E)$  and a set of vertices  $S \subseteq V$ , an additive query,  $Q(S)$ , asks for the number of edges in the subgraph induced by  $S$ . The information theoretic lower bound for the query complexity of reconstructing a graph with  $n$  vertices and  $m$  edges is

$$\Omega\left(\frac{m \log \frac{n^2}{m}}{\log m}\right).$$

In this paper we give the first polynomial time algorithm with query complexity that matches this lower bound<sup>1</sup>. This solves the open problem by [S. Choi and J. Han Kim. Optimal Query Complexity Bounds for Finding Graphs. *STOC*, 749–758, 2008].

In the paper, we actually show an algorithm for the generalized problem of reconstructing weighted graphs. In the weighted case, an additive query,  $Q(S)$ , asks for the sum of weights of edges in the subgraph induced by  $S$ . The complexity of the algorithm also matches the information theoretic lower bound.

## 1 Introduction

The problem of reconstructing a hidden graph using additive queries is the following: Suppose we have a hidden graph  $G = (V, E)$ . Suppose that the vertices of  $G$  are known and its edges are unknown. The goal is to exactly reconstruct the graph using additive queries of the following form

$$Q(S) = \text{How many edges exists in the subgraph induced by } S?$$

where  $S \subseteq V$ .

Given an algorithm for the problem, we distinguish between two types of algorithms. Adaptive algorithms,

are algorithms in which the choice of a query may depend on earlier queries' answers. Non-adaptive algorithms are algorithms in which all queries are determined before receiving any answers. Obviously, adaptive algorithms are more powerful, however, in practice non-adaptive algorithms are desirable since the queries can be parallelized.

The graph reconstructing problem, motivated by applications to genome sequencing [6], has known significant progress recently [14, 13, 12, 6, 15, 10, 9]. The information theoretic lower bound for the query complexity of reconstructing a graph  $G = (V, E)$  is

$$\Omega\left(\frac{m \log \frac{n^2}{m}}{\log m}\right).$$

where  $n = |V|$  and  $m = |E|$ .

A tight upper bound of  $O(dn)$  for non-adaptive algorithms for the class of  $d$ -bounded degree graphs was given by Grebinski and Kucherov [12]. They also show a polynomial time algorithm for reconstructing a graph using  $O(n^2/\log n)$  queries. In [14] Grebinski proved a tight upper bound of  $O(dn)$  for reconstructing  $d$ -degenerate graphs, that is, graphs that their edges can be changed to directed edges where the out-degree of each vertex is bounded by  $d$ . In [15], Reyzin and Srivastava give a simple polynomial time algorithm for reconstructing a graph with  $m$  edges and  $n$  vertices using  $O(m \log n)$  queries. In a recent work [10], S. Choi and J. Han Kim show a tight upper bound for reconstructing any graph with  $m$  edges and  $n$  vertices using  $O((m \log \frac{n^2}{m})/\log m)$  queries.

All optimal algorithms in the literature are non-constructive and therefore proving upper bounds only. That is, they prove the existence of a set of queries, such that, the answers to this set uniquely identifies the hidden graph. However, it is unknown how to find this set of queries deterministically in polynomial time and moreover, given the answers to the desired set of queries, it is not known how to reconstruct the graph in polynomial time.

In this paper, we give the first deterministic polyno-

\*hanna@cs.technion.ac.il

<sup>1</sup>In this paper, by optimal we mean asymptotically. That is, up to a constant factor.

mial time algorithm that matches the information theoretic lower bound for reconstructing any graph with  $m$  edges. The algorithm asks its queries in  $\log n$  non-adaptive rounds. Moreover, we actually show a more general result. We show that our algorithm works when the hidden graph is a weighted graph. In the weighted case, an additive query,  $Q(S)$ , asks for the sum of weights for edges in the subgraph induced by  $S$ . Formally speaking, let  $G = (V, E, w)$  be a graph where  $E \subseteq (V \times V)$  and  $w : E \rightarrow \mathbb{Z}^+$ , where  $\mathbb{Z}^+$  is the set of positive integers. Let  $n = |V|$  and  $m = \sum_{e \in E} w(e)$ . Then, we show that we can reconstruct the hidden graph in polynomial time, using additive queries of the following form

$$Q(S) = \sum_{e \in E \cap (S \times S)} w(e),$$

where  $S \subseteq V$ . Our query complexity is  $O(k(n, m))$  where

$$k(n, m) = \begin{cases} \frac{m \log\left(\frac{n^2}{m} + 1\right)}{\log m} & m < n^2 \\ \frac{n^2 \log\left(\frac{m}{n^2} + 1\right)}{n^2} & n^2 \leq m < n^4 \\ n^2 & m \geq n^4 \end{cases}.$$

This complexity matches the information theoretic lower bound.

The paper is organized as follows: In Section 2 we present some definitions, notation and background. In Section 3, we present an algorithm for reconstructing graphs given an upper bound for the weight of every edge. In Section 4 we present our main algorithm, that is, we give an algorithm for reconstructing a hidden graph without having an upper bound on the weight of every edge. Finally, Section 5 contains open problems.

## 2 Preliminaries

In this section, we present definitions and notation. We also present some background and basic tools that will be used for analyzing the complexity of the algorithm.

**2.1 Notation** We denote by  $\mathbb{Z}$  the set of integers. We denote by  $\mathbb{Z}^+$  the set of positive integers and by  $\mathbb{N}_0 = \mathbb{Z}^+ \cup \{0\}$  the set of positive integers and zero. For a positive integer  $c$ , we denote by  $[c]$  the set  $\{1, 2, \dots, c\}$ .

Let  $A = (a_{ij}) \in \mathbb{Z}^{n \times n}$  be a matrix. We denote by  $A_r(i)$  its  $i$ th row and by  $A_c(i)$  its  $i$ th column. We denote by  $A[i, j|k, l]$  the submatrix of  $A$  formed by selecting rows  $\{i, i + 1, \dots, j\}$  and columns  $\{k, k + 1, \dots, l\}$ .

We denote by  $e_i$  the  $i$ th column of the identity matrix  $I_n$ . Given a vector  $w$  we denote by  $w_i$  the  $i$ th entry of  $w$ . Given  $\sigma \in \{0, 1\}$  we denote by  $(\sigma)^k$  the  $k$ -vector where every entry equals  $\sigma$ .

Let  $x$  be a vector or a matrix. We denote by  $wt(x)$

the Hamming weight of  $x$ , that is, the number of non-zero entries in  $x$ . We denote by  $\psi(x)$  the sum of its entries.

**2.2 Vector Reconstructing Algorithms** We now mention a known algorithm for reconstructing hidden vectors. This algorithm will be the building block of our graph reconstructing algorithm.

**THEOREM 2.1.** (*Bshouty, [7]*) *Let  $w \in \mathbb{N}_0^n$  be a hidden vector. Let  $q \in \mathbb{N}_0^n$  be a vector such that  $w_i \leq q_i$  for all  $i \in [n]$ . Then, there is a polynomial time non-adaptive algorithm that given  $q$ , it can reconstruct  $w$  using  $k$  queries of the form  $g(x) = x^T w$  (here  $x \in \{0, 1\}^n$ ) where*

$$k(\log k - 4) \leq 2wt(q) \log \left( \frac{\psi(q)}{wt(q)} + 1 \right).$$

**2.3 Algebraic View of the Problem** Let  $G = (V, E, w)$  be a non-directed weighted graph where  $V = \{v_1, v_2, \dots, v_n\}$ ,  $E \subseteq V \times V$  and  $w : E \rightarrow \mathbb{Z}^+$ . Let  $A_G = (a_{ij}) \in \mathbb{N}_0^{n \times n}$  be its adjacency matrix, that is,  $a_{ij}$  equals  $w((i, j))$  if  $(i, j) \in E$  and equals zero otherwise. Given a set of vertices  $V' \subseteq V$  define the vector  $a$  where  $a_i$  equals “1” if  $v_i \in V'$  and “0” otherwise. Then, we have

$$Q(V') = \frac{a^T A_G a}{2}.$$

Now, let  $z = x * y = (x_i y_i) \in \{0, 1\}^n$  and  $x_1 = x - z$ ,  $y_1 = y - z$ . Since  $A_G$  is symmetric we have

$$x^T A_G y = \frac{x^T A_G x}{2} + \frac{y^T A_G y}{2} + \frac{(x_1 + y_1)^T A_G (x_1 + y_1)}{2} - x_1^T A_G x_1 - y_1^T A_G y_1.$$

See [12]. Thus, the problem of reconstructing a graph  $G$  using additive queries is equivalent to reconstructing its adjacency matrix  $A_G$  using queries of the form

$$f(x, y) = x^T A_G y,$$

where  $x, y \in \{0, 1\}^n$ .

**2.4 Means** In this subsection we present some known inequalities that will help us analyze the complexity of the algorithm.

Let  $r_1, r_2, \dots, r_n$  be  $n$  real numbers. The *arithmetic mean* of  $r_1, r_2, \dots, r_n$  is

$$AM(r_1, r_2, \dots, r_n) = \frac{r_1 + r_2 + \dots + r_n}{n}.$$

The *geometric mean* is

$$GM(r_1, r_2, \dots, r_n) = (r_1 \cdot r_2 \cdots r_n)^{\frac{1}{n}}.$$

The harmonic mean is

$$HM(r_1, r_2, \dots, r_n) = n \left( \frac{1}{r_1} + \frac{1}{r_2} + \dots + \frac{1}{r_n} \right)^{-1}.$$

When  $r_i \geq 0$  for all  $i \in [n]$  then, the means satisfy

$$HM(r_1, \dots, r_n) \leq GM(r_1, \dots, r_n) \leq AM(r_1, \dots, r_n).$$

We now rely on the above fact for proving two useful lemmas.

LEMMA 2.1. Let  $b_1, b_2, \dots, b_t, s_1, s_2, \dots, s_t$  be positive numbers. Let  $B = b_1 + b_2 + \dots + b_t$  and  $S = s_1 + s_2 + \dots + s_t$ . Then, we have

$$\prod_{i=1}^t \left( \frac{b_i}{s_i} + 1 \right)^{s_i} \leq \left( \frac{B}{S} + 1 \right)^S.$$

*Proof.* Define  $r_{ij} = b_i/s_i + 1$  for all  $i \in [t]$  and  $j \in [s_i]$ . Then, we have

$$\begin{aligned} \left( \prod_{i=1}^t \left( \frac{b_i}{s_i} + 1 \right)^{s_i} \right)^{1/S} &= GM(r_{11}, \dots, r_{ts_t}) \\ &\leq AM(r_{11}, \dots, r_{ts_t}) = \left( \frac{B}{S} + 1 \right). \end{aligned}$$

This implies the result. ■

LEMMA 2.2. Let  $k_1, k_2, \dots, k_t$  be positive numbers. Let  $K = k_1 + k_2 + \dots + k_t$ . Then we have

$$\prod_{i=1}^t (k_i)^{k_i} \geq \left( \frac{K}{t} \right)^K.$$

*Proof.* Define  $r_{ij} = k_i$  for all  $i \in [t]$  and  $j \in [k_i]$ . Then we have

$$\begin{aligned} \left( \prod_{i=1}^t (k_i)^{k_i} \right)^{1/K} &= GM(r_{11}, \dots, r_{tk_t}) \\ &\geq HM(r_{11}, \dots, r_{tk_t}) = \frac{K}{t} \end{aligned}$$

This implies the result. ■

### 3 Bounded Matrix Reconstruction

In this section we present a non-adaptive algorithm for reconstructing a matrix given an upper bound on the value of every entry of the matrix.

THEOREM 3.1. Let  $A = (a_{ij}) \in \mathbb{N}_0^{n \times n}$  be a hidden matrix. Let  $B = (b_{ij}) \in \mathbb{N}_0^{n \times n}$  be any matrix such that,  $a_{ij} \leq b_{ij}$  for every  $i, j \in [n]$ . Let  $m = \psi(B)$ , that is,  $m$  equals the sum of entries in  $B$ . Then, there exists a

polynomial time non-adaptive algorithm that given  $B$  it can reconstruct  $A$  using

$$O \left( \min \left\{ \frac{wt(B) \log \frac{m}{wt(B)}}{\log wt(B)}, wt(B) \right\} \right)$$

queries of the form  $f(x, y) = x^T A y$ , where  $x, y$  are  $(0, 1)$ -vectors of size  $n$ .

Before proving Theorem 3.1 we give some definitions and prove useful lemmas. The algorithm we will present is iterative. At each iteration, it chooses a large set of unknown entries and determine their values. The matrix  $A$  is partly reconstructed at each iteration, that is, some of its entries are known to the algorithm, others are unknown.

At a given iteration, let  $M = (m_{ij}) \in (\mathbb{N}_0 \cup \{*\})^{n \times n}$  be a matrix that holds the known entries of  $A$ . That is,  $m_{ij} = a_{ij}$  for all entries  $a_{ij}$  that are known, and  $m_{ij} = *$  for all entries  $a_{ij}$  that are still unknown. For such matrix  $M$ , we say that  $M$  is a *partial copy* of  $A$ . Define  $\Gamma(m_{ij})$  to be “1” if  $m_{ij} = *$  and “0” otherwise. Notice that for any entry  $m_{ij}$  such that  $b_{ij} = 0$  we have that  $a_{ij} = m_{ij} = 0$  and therefore  $\Gamma(m_{ij}) = 0$ . We extend the definition of  $\Gamma$  in the following way: if  $x$  is a row, a column or a matrix then  $\Gamma(x)$  equals the number of entries in  $x$  that are equal to “\*”.

Now, Let  $(i_1, j_1)$  and  $(i_2, j_2)$  where  $i_1, i_2, j_1, j_2 \in [n]$  be two indices. We say that these indices are “independent” in  $M$  if  $i_1 \neq i_2, j_1 \neq j_2, \Gamma(m_{i_1 j_1}) = 1, \Gamma(m_{i_2 j_2}) = 1, \Gamma(m_{i_1 j_2}) = 0$  and  $\Gamma(m_{i_2 j_1}) = 0$ . A set of indices  $S$  is called “independent set of indices” in  $M$  if for every pair of indices  $u, v \in S$ ,  $u$  and  $v$  are independent in  $M$ .

We now give useful lemmas showing how to reconstruct various sets of unknown entries.

LEMMA 3.1. There exists an algorithm that reconstruct the  $i$ th row of  $A$ ,  $A_r(i)$ , (and therefore the  $i$ th column, since  $A$  is symmetric) using  $k$  queries where

$$k(\log k - 4) \leq 2wt(B_r(i)) \log \left( \frac{\psi(B_r(i))}{wt(B_r(i))} + 1 \right).$$

*Proof.* We can query the  $i$ th row  $A_r(i)$  by asking  $e_i^T A y$ . Thus, using Theorem 2.1 the result follows. ■

Next we have,

LEMMA 3.2. Let  $M$  be a partial copy of  $A$ . Let  $S = \{(i_1, j_1), (i_2, j_2), \dots, (i_{|S|}, j_{|S|})\}$  be an independent set of indices in  $M$  and let  $b = \sum_{(i,j) \in S} b_{ij}$ . Then, entries  $a_{i_1 j_1}, a_{i_2 j_2}, \dots, a_{i_{|S|} j_{|S|}}$  can be reconstructed using  $k$  queries where

$$k(\log k - 4) \leq 2|S| \log \left( \frac{b}{|S|} + 1 \right).$$

*Proof.* Let  $C = (c_{ij}) \in \mathbb{N}_0^{n \times n}$  be a matrix. Let

$$c_{ij} = (1 - \Gamma(m_{ij}))a_{ij}.$$

In other words,  $c_{ij}$  equals  $a_{ij}$  if  $a_{ij}$  is known and equals zero otherwise. The matrix  $C$  can be reconstructed without asking queries. Now, let us look at the matrix

$$D = A - C.$$

The matrix  $D$  has the following properties:

1. For every  $r \in [|S|]$  we have  $d_{i_r, j_r} = a_{i_r, j_r}$ .
2. For every  $(i, j)$  such that  $i = i_t, j = j_r$  where  $r, t \in [|S|]$  and  $r \neq t$ , we have  $d_{ij} = 0$ .
3. For any two vectors  $x, y \in \{0, 1\}^n$  we have  $x^T D y = x^T A y - x^T C y$ .

Next, let  $N = (a_{i_1 j_1}, a_{i_2 j_2}, \dots, a_{i_{|S|} j_{|S|}})$  be a vector. We now show how to query the vector  $N$ , that is, simulate  $g(z) = z^T N$  where  $z \in \{0, 1\}^{|S|}$  using queries of the form  $f(x, y) = x^T A y$  where  $x, y \in \{0, 1\}^n$ .

Let  $z \in \{0, 1\}^{|S|}$  be a vector. Define  $X(z) \in \{0, 1\}^n$  to be “1” in every entry  $i_r$  such that  $z_r = 1$  and zero otherwise (here  $r \in [|S|]$ ). Also, Let  $Y(z) \in \{0, 1\}^n$  be “1” in each entry  $j_r$  such that  $z_r = 1$  and zero otherwise. From property (1) and (2) we have that

$$z^T N = X(z)^T D Y(z).$$

Therefore, by property (3) we get

$$z^T N = X(z)^T A Y(z) - X(z)^T C Y(z).$$

Since  $C$  can be reconstructed without asking queries, we can simulate  $g(z)$  using  $f(x, y)$ . By Theorem 2.1 the result follows. ■

Finally, we have

**LEMMA 3.3.** *Let  $\alpha < 1/3$  be any constant and let  $M$  be a partial copy of  $A$ . If  $\Gamma(M) \geq wt(B)^{3\alpha}$  and for all  $i \in [n]$ ,  $\Gamma(M_r(i)) < wt(B)^\alpha$  and  $\Gamma(M_c(i)) < wt(B)^\alpha$ , then, there exists an independent set of indices in  $M$  of size at least  $wt(B)^\alpha/2$ .*

*Proof.* Choose any index  $(i, j)$  such that  $m_{ij} = *$ . Now, for every index  $(i, \ell_1)$  such that  $\Gamma(m_{i\ell_1}) = 1$  remove the column  $\ell_1$  from the matrix. Also, for every index  $(\ell_2, j)$  such that  $\Gamma(m_{\ell_2 j}) = 1$  remove the row  $\ell_2$  from the matrix. Since every row or column has at most  $wt(B)^\alpha$  unknown entries, then we have removed at most  $2wt(B)^{2\alpha} - 2wt(B)^\alpha + 1$  unknown entries. Note that the index  $(i, j)$  is independent with the remaining  $\Gamma(M) - 2wt(B)^{2\alpha} + 2wt(B)^\alpha - 1$  indices that contains unknown entries.

The above can be repeated

$$\frac{\Gamma(M)}{2wt(B)^{2\alpha} - 2wt(B)^\alpha + 1} \geq \frac{wt(B)^{3\alpha}}{2wt(B)^{2\alpha}}$$

times. Thus, by repeating the above we are able to find a set of  $(wt(B)^\alpha)/2$  independent indices in  $M$ . ■

Now, after proving the above lemmas we present our algorithm.

**3.1 The algorithm** The algorithm, presented in Figure 1, first checks whether

$$(3.1) \quad wt(B) \leq \frac{wt(B) \log \frac{\psi(B)}{wt(B)}}{\log wt(B)}.$$

In case (3.1) is true, the algorithm asks a query to determine the value of every entry  $a_{ij}$  such that  $b_{ij} > 0$  (the value  $a_{ij}$  equals  $e_i^T A e_j$ ). Otherwise, the algorithm works in iterations. At each iteration, the goal is to find a sufficiently large set (that is, of size  $\Omega(wt(B)^\alpha)$ , for some constant  $0 < \alpha < 1/3$ ) of unknown entries, and to determine their values using the known vector reconstructing algorithm. The algorithm first reconstruct all rows with at least  $wt(B)^\alpha$  unknown entries (see Lemma 3.1). Then, at each iteration, it finds a large set of independent indices and determine the value of the entries in those indices (Lemma 3.2 and Lemma 3.3). The algorithm continues iterating until at most  $wt(B)^{3\alpha}$  unknown entries are left in  $A$ . Then, it asks a query for each unknown entry.

This algorithm is non-adaptive. This is because the choice of a later query does not depend on an earlier query’s answer. In each iteration  $i$  the algorithm chooses a set of entries  $S_i$  to reconstruct. The choice of this set depends on which entries are known and which are unknown; However, we do not rely on the values of the known entries. That is, the sets  $S_1, S_2, \dots, S_t$  can be determined before any query is asked. Thus, the set of queries the algorithm asks can be determined before receiving any answers. After receiving all answers, the construction of the entries must be sequential. That is, we need to know all the values of entries in  $S_1, \dots, S_{i-1}$  to reconstruct  $S_i$ .

**3.2 Complexity Analysis** In this subsection we prove Theorem 3.1.

*Proof.* At every iteration the algorithm finds a set of entries of size  $\Omega(wt(B)^\alpha)$  and determines their values. Let  $S_i$  be the set of entries the algorithm reconstructed at iteration  $i$ . Note that  $S_i \cap S_j = \emptyset$  for all  $i, j \in [t]$  where  $i \neq j$  and  $t$  equals the number of iterations. let

Algorithm **Matrix Reconstruct**( $B \in \mathbb{N}_0^{n \times n}$ )

1. Define  $M = (m_{ij}) \in (\mathbb{N}_0 \cup \{*\})^{n \times n}$ .
2. **For** every  $i, j \in [n]$  **do**
3.      $m_{ij} \leftarrow 0$ .
4.     **if**  $b_{ij} > 0$  **then**  $m_{ij} \leftarrow *$ .
5. **End For**.
  
6. **if**  $wt(B) \leq \frac{wt(B) \log \frac{\psi(B)}{wt(B)}}{\log wt(B)}$  **then**
7.     Goto 18.
8. **End if**.
  
9. **While**(  $\Gamma(M) > wt(B)^{3\alpha}$ )
10.     **if** exists  $i$  such that  $\Gamma(M_r(i)) > wt(B)^\alpha$  **then**
11.         reconstruct  $M_r(i)$ .
12.         update  $M_c(i)$  (since  $A$  is symmetric).
13.         Goto 9.
14.     **End if**.
15.     Find an independent set of indices  $S$  in  $M$  such that  $|S| > wt(B)^\alpha/2$ .
16.     Determine the values of the entries in  $S$ .
17. **End While**.
  
18. **For** every  $(i, j)$  such that  $\Gamma(m_{ij}) = 1$  **do**
19.     Ask  $f(e_i, e_j)$ .
20. **End For**.

---

Algorithm:     **Matrix Reconstruct**

Description:   A reconstruction algorithm for a matrix  $A$ .

Complexity:    $O\left(\min\left\{\frac{wt(B) \log \frac{\psi(B)}{wt(B)}}{\log wt(B)}, wt(B)\right\}\right)$

Figure 1: Algorithm for reconstructing a matrix given upper bounds on the entries

$b_i = \sum_{(u,v) \in S_i} b_{uv}$  and  $s_i = |S_i|$ . By Theorem 2.1 the number of queries used for reconstructing the  $i$ th set, denoted by  $k_i$ , satisfies

$$k_i(\log k_i - 4) \leq 2s_i \log \left( \frac{b_i}{s_i} + 1 \right).$$

Thus,

$$\sum_i k_i(\log k_i - 4) \leq \sum_i 2s_i \log \left( \frac{b_i}{s_i} + 1 \right).$$

Let  $S = s_1 + s_2 + \dots + s_t$ . We have

$$\begin{aligned} \sum_i 2s_i \log \left( \frac{b_i}{s_i} + 1 \right) &= 2 \log \prod_i \left( \frac{b_i}{s_i} + 1 \right)^{s_i} \\ (3.2) \quad &\leq 2 \log \left( \frac{m}{S} + 1 \right)^S \\ &= 2S \log \left( \frac{m}{S} + 1 \right) \\ (3.3) \quad &\leq 2wt(B) \log \left( \frac{m}{wt(B)} + 1 \right). \end{aligned}$$

In (3.2) we use Lemma 2.1. The inequality (3.3) follows from the fact that the function  $f(x) = x \log \left( \frac{m}{x} + 1 \right)$  is monotone non-decreasing for  $x \in [1, m]$  and the fact that  $S \leq wt(B) \leq m$ .

On the other hand, denote by  $K = k_1 + k_2 + \dots + k_t$ . We have

$$\begin{aligned} \sum_i k_i(\log k_i - 4) &= \sum_i k_i(\log k_i) - 4K \\ &= \log \prod_i (k_i)^{k_i} - 4K \\ (3.4) \quad &\geq K \log \frac{K}{t} - 4K \end{aligned}$$

In (3.4) we use Lemma 2.2. Therefore, the total number of queries  $K$  for reconstructing the sets satisfies

$$K \log \frac{K}{t} - 4K \leq 2wt(B) \log \left( \frac{m}{wt(B)} + 1 \right).$$

Since for all  $i$  we have  $s_i = \Omega(wt(B)^\alpha)$ , we conclude that the number of sets is

$$t = O(wt(B)^{1-\alpha}).$$

Thus,

$$K = O \left( \frac{wt(B) \log \left( \frac{m}{wt(B)} + 1 \right)}{\log wt(B)} \right).$$

For more details see [7]. To conclude the total number of queries asked is at most

$$O \left( \frac{wt(B) \log \left( \frac{m}{wt(B)} + 1 \right)}{\log wt(B)} \right) + wt(B)^{3\alpha}.$$

Since  $\alpha < 1/3$  the result follows. ■

#### 4 The Main algorithm

In the previous section we showed how to reconstruct a matrix given an upper bound on every entry of the matrix. In this section, we show how to reconstruct the matrix without having those upper bounds.

Formally speaking,

**THEOREM 4.1.** *Let  $A = (a_{ij}) \in \mathbb{N}_0^{n \times n}$  be a hidden matrix such that the sum of the entries of  $A$  is equal to  $m$ . Then, there exists a polynomial time algorithm that can reconstruct  $A$  using*

$$O \left( \frac{m \log \frac{n^2}{m}}{\log m} \right)$$

queries of the form  $f_A(x, y) = x^T A y$ , where  $x, y$  are  $(0, 1)$ -vectors of size  $n$ . This algorithm asks its queries in  $\log n$  non-adaptive rounds.

*Proof.* First, assume without loss of generality that  $\log n$  is an integer. Define the following matrices  $A^{(0)}, A^{(1)}, \dots, A^{(\log n)}$  where  $A^{(i)} \in \mathbb{N}_0^{2^{2^i} \times 2^{2^i}}$ . Denote by  $a_{r,j}^{(i)}$  the  $(r, j)$ th entry of  $A^{(i)}$ . Let

$$a_{r,j}^{(i)} = \psi(A[h_1^{(i)}(r), h_2^{(i)}(r) | h_1^{(i)}(j), h_2^{(i)}(j)])$$

where  $h_1^{(i)}(z) = (z-1)n/2^i + 1$  and  $h_2^{(i)}(z) = zn/2^i$  for  $z \in \mathbb{Z}$ . In other words, if we divide  $A$  into  $2^{2^i}$  blocks  $A_{r,j}$  of size  $\frac{n}{2^i} \times \frac{n}{2^i}$ , that is,

$$A = \begin{pmatrix} \begin{array}{c|c|c|c} A_{1,1} & A_{1,2} & \dots & A_{1,2^i} \\ \hline A_{2,1} & A_{2,2} & \dots & A_{2,2^i} \\ \hline \vdots & \vdots & & \vdots \\ \hline A_{2^i,1} & A_{2^i,2} & \dots & A_{2^i,2^i} \end{array} \end{pmatrix},$$

then,  $a_{r,j}^{(i)} = \psi(A_{r,j})$  (recall that  $\psi(x)$  equals the sum of entries in  $x$ ). Note that,  $A^{(0)}$  has one entry that is equal to  $m$ . Also note that  $A^{(\log n)} = A$ .

**LEMMA 4.1.** *We can simulate queries for the matrices  $A^{(i)}$  using queries for the matrix  $A$ .*

*Proof.* Simply note that  $x^T A^{(i)} y = (x')^T A y'$  where

$$x' = (x_0)^{\frac{n}{2^i}} \cdot (x_1)^{\frac{n}{2^i}} \cdots (x_{2^i})^{\frac{n}{2^i}}$$

and

$$y' = (y_0)^{\frac{n}{2^i}} \cdot (y_1)^{\frac{n}{2^i}} \cdots (y_{2^i})^{\frac{n}{2^i}},$$

and  $\cdot$  denote concatenation. ■

We now present our main algorithm. Our algorithm is iterative. At iteration  $i$  the algorithm knows  $A^{(i-1)}$  and the goal is to reconstruct the matrix  $A^{(i)}$ .

Note that

$$a_{rj}^{(i-1)} = \sum_{d=2r-1}^{2r} \sum_{\ell=2j-1}^{2j} a_{d\ell}^{(i)}.$$

Thus, every entry  $a_{rj}^{(i)}$  is bounded by  $a_{\lceil \frac{r}{2} \rceil \lceil \frac{j}{2} \rceil}^{(i-1)}$ .

Define  $B^{(i)} = (b_{rj}^{(i)}) \in \mathbb{N}_0^{2^i \times 2^i}$  where

$$b_{rj}^{(i)} = a_{\lceil \frac{r}{2} \rceil \lceil \frac{j}{2} \rceil}^{(i-1)}.$$

Then, we have that  $a_{rj}^{(i)} \leq b_{rj}^{(i)}$  and  $\psi(B^{(i)}) = 4\psi(A^{(i-1)}) = 4m$ . Thus, using Lemma 4.1 and Theorem 3.1, we are able to reconstruct  $A^{(i)}$ .

Now, since  $A^{(\log n)}$  is equal to  $A$ , then after  $\log n$  iteration the algorithm has successfully reconstructed the hidden matrix.

**4.1 Complexity Analysis** In this subsection we prove Theorem 4.1.

*Proof.* Define

$$b_i = wt(B^{(i)}).$$

Also define

$$K(i) = \min \left\{ \frac{b_i \left( \log \frac{4m}{b_i} + 1 \right)}{\log b_i}, b_i \right\}.$$

By Theorem 3.1, our total complexity is

$$O \left( \sum_{i=1}^{\log n} K(i) \right).$$

We will divide this sum to three parts  $i \leq \frac{1}{4} \log m$ ,  $\frac{1}{4} \log m < i \leq \frac{1}{2} \log m$  and  $\frac{1}{2} \log m < i \leq \log n$  and estimate each part separately.

The first part

$$\sum_{i=1}^{\frac{1}{4} \log m} K(i) \leq \sum_{i=1}^{\frac{1}{4} \log m} wt(B^{(i)}) \leq \sum_{i=1}^{\frac{1}{4} \log m} 4^i = O(\sqrt{m}).$$

For the second part, we note that if  $b_i \leq \sqrt{m}$  then  $K(i) \leq b_i \leq \sqrt{m}$ . Otherwise,  $b_i > \sqrt{m}$  and then  $K(i) \leq \frac{b_i \log \left( \frac{4m}{b_i} + 1 \right)}{\log b_i} \leq \frac{b_i \log \left( \frac{4m}{b_i} + 1 \right)}{\log \sqrt{m}}$ . Therefore,

$$K(i) < \frac{b_i \log \left( \frac{4m}{b_i} + 1 \right)}{\log \sqrt{m}} + \sqrt{m}.$$

Now we have

$$\begin{aligned} \sum_{i=\frac{1}{4} \log m+1}^{\frac{1}{2} \log m} K(i) &\leq \sum_{i=\frac{1}{4} \log m+1}^{\frac{1}{2} \log m} \left( \frac{b_i \log \left( \frac{4m}{b_i} + 1 \right)}{\log \sqrt{m}} + \sqrt{m} \right) \\ &= \frac{1}{4} \sqrt{m} \log m + \sum_{i=\frac{1}{4} \log m+1}^{\frac{1}{2} \log m} \frac{b_i \log \left( \frac{4m}{b_i} + 1 \right)}{\log \sqrt{m}} \\ (4.5) \quad &\leq \frac{1}{4} \sqrt{m} \log m + \sum_{i=\frac{1}{4} \log m+1}^{\frac{1}{2} \log m} \frac{4^i \log \left( \frac{4m}{4^i} + 1 \right)}{\log \sqrt{m}} \\ &\leq \frac{1}{4} \sqrt{m} \log m + \sum_{i=\frac{1}{4} \log m+1}^{\frac{1}{2} \log m} \frac{4^i \log (8m - \log 4^i)}{\log \sqrt{m}} \\ &= \frac{1}{4} \sqrt{m} \log m + \frac{1}{\log \sqrt{m}} \sum_{i=\frac{1}{4} \log m+1}^{\frac{1}{2} \log m} (4^i \log 8m - 2i 4^i) \\ &= \frac{1}{4} \sqrt{m} \log m \\ &\quad + \frac{\left( \frac{4m}{3} - \frac{4\sqrt{m}}{3} \right) \log 8m - \left( \frac{4m}{3} \log m - O(m) \right)}{\log \sqrt{m}} \\ &= O \left( \frac{m}{\log m} \right). \end{aligned}$$

Inequality (4.5) follows from the fact that the function  $f(x) = x \log \left( \frac{4m}{x} + 1 \right)$  is a monotone non-decreasing function for  $x \in [1, 4m]$  and the fact that  $b_i \leq 4^i \leq m$ .

Finally, the last part, that is,  $1/2 \log m < i \leq \log n$ , we have two cases: if  $b_i < \sqrt{m}$  then,  $K(i) \leq b_i \leq \sqrt{m}$ , otherwise, we have  $b_i \geq \sqrt{m}$ , and then

$$K(i) \leq \frac{b_i \log \left( \frac{4m}{b_i} + 1 \right)}{\log b_i} \leq \frac{b_i \log \frac{8m}{b_i}}{\log \sqrt{m}} \leq \frac{b_i \frac{8m}{b_i}}{\log \sqrt{m}} = \frac{16m}{\log m}.$$

In both cases we have

$$K(i) = O \left( \frac{m}{\log m} \right).$$

Thus,

$$\sum_{i=\frac{1}{2} \log m+1}^{\log n} K(i) = O \left( \frac{m \log \frac{n}{\sqrt{m}}}{\log m} \right) = O \left( \frac{m \log \frac{n^2}{m}}{\log m} \right).$$

By adding all three part together, the result immediately follows. ■

The algorithm is also optimal when  $m > n^2$ . A different analysis for this case is given in Appendix A.

## 5 Open Problems

Our main algorithm asks the queries in  $\log n$  non-adaptive rounds. The number of rounds can be easily reduced to  $\frac{\log n}{c}$  for any constant  $c$  by reconstructing  $A^{(0)}, A^{(c)}, A^{(2c)}, \dots, A^{(\log n)}$  instead of reconstructing all  $A^{(i)}$ . This action increases the number of queries need to be asked by a constant factor. An open question is: can we reduce the number of rounds to be a constant without increasing the query complexity?

Also, in [10, 9, 8] non-constructive algorithms are given for reconstructing weighted graphs with real number weights. While our algorithm works with weights that are integers, an explicit construction for reconstructing weighted graphs with real number weights is still open.

## Acknowledgements

I would like to thank Nader H. Bshouty for useful discussions and comments.

## References

- [1] M. Aigner. Combinatorial Search. *John Wiley and Sons*, 1988.
- [2] N. Alon and V. Asodi. Learning a Hidden Subgraph. *SIAM J. Discrete Math.*, 18, 4, 697–712, 2005.
- [3] N. Alon, R. Beigel, S. Kasif, S. Rudich and B. Sudakov. Learning a Hidden Matching. *SIAM J. Comput.* 33, 2, 487–501, 2004.
- [4] D. Angluin. and J. Chen. Learning a Hidden Graph Using  $O(\log n)$  Queries per Edge. *COLT*, 210–223, 2004.
- [5] D. Angluin and J. Chen. Learning a Hidden Hypergraph. *Journal of Machine Learning Research*, 7, 2215–2236, 2006.
- [6] M. Bouvel, V. Grebinski, G. Kucherov: Combinatorial Search on Graphs Motivated by Bioinformatics Applications: A Brief Survey. *WG*, 16–27, 2005.
- [7] N. H. Bshouty. Optimal Algorithms for the Coin Weighing Problem with a Spring Scale. *COLT*, 2009.
- [8] N. H. Bshouty and H. Mazzawi. On Parity Check (0,1)-Marix over  $Z_p$ . *ECCC*, 2009.
- [9] N. H. Bshouty and H. Mazzawi. Reconstructing Weighted Graphs with Minimal Query Complexity. *ALT*, 2009.
- [10] S. Choi, J. Han Kim. Optimal Query Complexity Bounds for Finding Graphs. *STOC*, 749–758, 2008.
- [11] D. Du and F. K. Hwang. Combinatorial group testing and its application, Volume 3 of Series on applied mathematics. *World Science*, 1993.
- [12] V. Grebinski and G. Kucherov. Optimal Reconstruction of Graphs Under the Additive Model. *Algorithmica*, 28(1), 104–124, 2000.
- [13] V. Grebiniski and G. Kucherov. Reconstructing a hamiltonian cycle by querying the graph: Application

to DNA physical mapping. *Discrete Applied Mathematics*, 88, 147–165, 1998.

- [14] V. Grebinski. On the Power of Additive Combinatorial Search Model. *COCOON*, 194–203, 1998.
- [15] L. Reyzin and N. Srivastava. Learning and Verifying Graphs using Queries with a Focus on Edge Counting. *ALT*, 2007.
- [16] E. Sperner. Ein Satz ber Untermengen einer endlichen Menge. *Math Z.*, 27, 544–548, 1928.

## Appendix A

We now give a complexity analysis for the main algorithm in case  $m > n^2$ . Define

$$K(i) = \min \left\{ \frac{wt(B^{(i)}) \left( \log \frac{4m}{wt(B^{(i)})} + 1 \right)}{\log wt(B^{(i)})}, wt(B^{(i)}) \right\}.$$

Also define

$$b_i = wt(B^{(i)})$$

Our total complexity is

$$O \left( \sum_{i=1}^{\log n} K(i) \right).$$

When  $m > n^2$  we have

$$\begin{aligned} \sum_{i=1}^{\log n} K(i) &= \sum_{i=1}^{\frac{1}{2} \log n} K(i) + \sum_{i=\frac{1}{2} \log n+1}^{\log n} K(i) \\ &\leq \sum_{i=1}^{\frac{1}{2} \log n} 4^i + \sum_{i=\frac{1}{2} \log n+1}^{\log n} \frac{b_i \log \left( \frac{4m}{b_i} + 1 \right)}{\log n} + n \\ &= O(n \log n) + \sum_{i=\frac{1}{2} \log n+1}^{\log n} \frac{b_i \log \left( \frac{4m}{b_i} + 1 \right)}{\log n} \\ &= O(n \log n) + \sum_{i=\frac{1}{2} \log n+1}^{\log n} \frac{4^i \log \left( \frac{4m}{4^i} + 1 \right)}{\log n} \\ &\leq O(n \log n) + \sum_{i=\frac{1}{2} \log n+1}^{\log n} \frac{4^i (\log 8m - \log 4^i)}{\log n} \\ &= O(n \log n) \\ &\quad + \frac{\frac{4}{3} (n^2 - O(n)) \log 8m - \left( \frac{8}{3} n^2 \log n - O(n^2) \right)}{\log n} \\ &\leq O(n \log n) + \frac{\frac{4}{3} n^2 \log \frac{m}{n^2} + O(n^2)}{\log n} \\ &= O \left( \frac{n^2 \log \frac{m}{n^2}}{\log n} \right). \end{aligned}$$

This proves that the algorithm is optimal even when  $m > n^2$ . ■