Almost Tight Upper Bound for Finding Fourier Coefficients of Bounded Pseudo-Boolean Functions

Sung-Soon Choi[*]
Random Graph Research Center
Yonsei University
Seoul, 120-749 Korea
ss.choi@yonsei.ac.kr

Kyomin Jung
Department of Mathematics
MIT
Cambridge, MA02139, USA
kmjung@mit.edu

Jeong Han Kim[†]
Department of Mathematics and
Random Graph Research Center
Yonsei University
Seoul, 120-749 Korea
jehkim@yonsei.ac.kr

Abstract

A pseudo-Boolean function is a real-valued function defined on $\{0,1\}^n$. A k-bounded function is a pseudo-Boolean function that can be expressed as a sum of subfunctions each of which depends on at most k input bits. The k-bounded functions for constant k play an important role in a number of research areas including molecular biology, biophysics, and evolutionary computation. In this paper, we consider the problem of finding the Fourier coefficients of k-bounded functions with a series of function evaluations at any input strings. Suppose that a k-bounded function f with m non-zero Fourier coefficients is given. Our main result is to present an adaptive randomized algorithm to find the Fourier coefficients of f with high probability in $\mathcal{O}(m \log n)$ function evaluations for constant k. Up to date, the best known upper bound is $\mathcal{O}(\alpha(n,m) m \log n)$, where $\alpha(n,m)$ is between $n^{\frac{1}{2}}$ and n depending on m. Thus, our bound improves the previous bound by a factor of $\Omega\left(n^{\frac{1}{2}}\right)$. Also, it is almost tight with respect to the known lower bound $\Omega\left(\frac{m \log n}{\log m}\right)$. To obtain the main result, we first show that the problem of finding the Fourier coefficients of a k-bounded function is reduced to the problem of finding a k-bounded hypergraph with a certain type of queries under an oracle with one-sided error. For this, we devise a method to test with one-sided error whether there is a dependency within some set of input bits among a collection of sets of input bits. Then, we give a randomized algorithm for the hypergraph finding problem and obtain the desired bound by analyzing the algorithm based on a large deviation result for a sum of independent random variables.

[*]This work was supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MOST) (No. R16-2007-075-01000-0).

[†]This work was partially supported by Yonsei University Research Funds 2006-1-0078 and 2007-1-0025, and by the second stage of the Brain Korea 21 Project in 2007, and by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD) (KRF-2006-312-C00455).

1 Introduction

A *pseudo-Boolean* function is a real-valued function defined on the set of binary strings of fixed length. If a pseudo-Boolean function can be expressed as a sum of subfunctions each of which depends on at most k input bits, it is called *k-bounded*. Given a 2-SAT formula, for example, the number of clauses an assignment satisfies is a 2-bounded pseudo-Boolean function of the assignment. Note that a k-bounded pseudo-Boolean function is a polynomial of Boolean variables of degree k or less, and vice versa. In this paper, we consider the problem of finding the Fourier coefficients of k-bounded pseudo-Boolean functions. In the problem, we assume the oracle that, given any binary string, returns the function value at the string. Our main concern is the query complexity to solve the problem, i.e., the number of function evaluations required to find the Fourier coefficients of k-bounded pseudo-Boolean functions. (Unless otherwise specified, a k-bounded function means a k-bounded pseudo-Boolean function in this paper.)

The k-bounded functions have played an important role in molecular biology and biophysics. In those areas, a number of mathematical models have been proposed to study the evolution of a population of organisms (or biological objects) [Ewe79, FL70, KL87, Lew74, MP89]. In many of the models including the NK model [Kau89], k-bounded functions have been used to measure the fitness of an organism in an environment. In the NK model [Kau89], each subfunction represents the contribution of a gene of the organism to the overall fitness, interacting with a fixed number of other genes. Hence, a k-bounded function may be regarded as a sum of subfunctions each of which depends on at most k genes. The k-bounded functions with small k in the NK model induce the fitness landscapes of reasonable evolvability and complexity, which were used for describing the evolution of living systems [Kau93]. They were also used as a benchmark for comparing the landscapes arising in RNA folding [FSBB$^+$93]. In this regard, k-bounded functions with small k have been paid attention.

The k-bounded functions have been also used as testbed problems for comparing the performance of heuristic algorithms in the area of evolutionary computation [CC06, HG97, MM99, MG99, PG00]. The problem of maximizing arbitrary k-bounded functions is NP-hard even for $k = 2$ as it is at least as hard as the MAX-2-SAT problem [GJS76]. The larger the value of k is, the higher is the degree of the depen-

dency among the input bits in a k-bounded function. By controlling the degree of the dependency (the value of k), in general, we may control the difficulty of the problem of maximizing the k-bounded functions. There are good heuristic algorithms to approximate the maximum of a k-bounded function when the dependency among the input bits are known [dBIV97, Gol89, MM99, PGCP00, Str04].

Fourier transform is a formal approach to define the dependency among the input bits of a pseudo-Boolean function. There have been a number of papers addressing the problem of finding the Fourier coefficients of a k-bounded function $f : \{0,1\}^n \to \mathbb{R}$ with constant k. Kargupta and Park [KP01] presented a deterministic algorithm using $\mathcal{O}(n^k)$ function evaluations. Later, Heckendorn and Wright [HW03, HW04] proposed a randomized algorithm for the problem. They analyzed the algorithm to show that, with negligible error probability, it finds the Fourier coefficients in $\mathcal{O}(n^2 \log n)$ function evaluations on average for the k-bounded functions with $\mathcal{O}(n)$ non-zero Fourier coefficients generated from a random model. For the k-bounded functions with m non-zero Fourier coefficients, Choi, Jung, and Moon [CJM08] proved that any randomized algorithm requires $\Omega\left(\frac{m \log n}{\log m}\right)$ function evaluations to find the Fourier coefficients with error probability at most a given constant. By analyzing the algorithm of Heckendorn and Wright, they also proved that $\mathcal{O}(\alpha(n,m) m \log n)$ function evaluations, where $\alpha(n,m)$ is between $n^{\frac{1}{2}}$ and n depending on m, are enough to find the Fourier coefficients. Recently, for 2-bounded functions of which non-zero Fourier coefficients are between n^{-a} and n^b in absolute value for some positive constants a and b, Choi and Kim [CK08] showed that there exists a deterministic algorithm using $\mathcal{O}\left(\frac{m \log n}{\log m}\right)$ function evaluations, provided that $m \geq n^{\varepsilon}$ for any constant $\varepsilon > 0$. This algorithm is non-adaptive while the previous algorithms are adaptive.[1] However, an explicit construction of the algorithm is unknown.

Our main result is

Theorem 1 *Suppose that f is a k-bounded function defined on $\{0,1\}^n$ for constant k and that f has m non-zero Fourier coefficients. Then, there exists an adaptive algorithm to find the Fourier coefficients of f in $\mathcal{O}(m \log n)$ function evaluations with probability $1 - \mathcal{O}\left(\frac{1}{n}\right)$.*

We prove Theorem 1 by showing an explicit construction of the desired algorithm. This result improves the best known upper bound $\mathcal{O}(\alpha(n,m) m \log n)$ by a factor of $\Omega\left(n^{\frac{1}{2}}\right)$ and it is almost tight with respect to the lower bound $\Omega\left(\frac{m \log n}{\log m}\right)$.

We should note that there have been a number of papers addressing the problem of finding the Fourier coefficients of Boolean functions [BJT04, BT96, Jac97, KM93, Man94]. The KM algorithm [KM93] is one of the most famous algorithms for the problem and most of the subsequent algorithms have been based on the algorithm. These algorithms for Boolean functions can be extended to pseudo-Boolean functions. However, the extensions of the algorithms do not give a good bound for k-bounded pseudo-Boolean functions. One of the main reasons is that their query complexities depend on the values of the target function. For example, for a k-bounded function f, (to the best of our knowledge) the most efficient extension [BJT04] among those has the query complexity of $\Omega\left(r\left(\frac{B}{\theta}\right)^2\right)$, where r is the number of input bits on which f depends, B is the maximum absolute value of f, and θ is the minimum absolute value of the non-zero Fourier coefficients of f. Thus, the query complexity may be made arbitrarily large depending on B and θ.[2] The query complexity of our algorithm is independent of the values of the target function.

To prove Theorem 1, we first show that the problem of finding the Fourier coefficients of a k-bounded function is reduced to the problem of finding a k-bounded hypergraph[3] (with a certain type of queries under a probabilistic oracle). For a pseudo-Boolean function f defined on $\{0,1\}^n$, we consider the hypergraph representing the dependency among the input bits as follows. Suppose that H is a subset of $[n]$, where $[n]$ is the set of the integers from 1 to n. We say that there is a *linkage* among the input bits in H if, for any additive expression of f, $f = \sum_i f_i$, there is j such that H is included in the support set of f_j.[4] The *linkage graph* of f is a hypergraph $G_f = ([n], E)$, where each bit in $[n]$ represents a vertex and a subset H of $[n]$ belongs to the edge set E if and only if there is a linkage among the bits in H.

For example, consider the following function:

$$f(x_1, x_2, x_3, x_4, x_5) = 5x_1x_2 - 3x_2x_3x_4.$$

If we let $f_1(x_1, x_2) = 5x_1x_2$ and $f_2(x_2, x_3, x_4) = -3x_2x_3x_4$, f can be represented as an additive expression, $f = f_1 + f_2$. In this expression, each subfunction of f has a support set of which size is at most three and so f is 3-bounded. It can be shown that the support sets of f_1 and f_2, $\{1,2\}$ and $\{2,3,4\}$, are hyperedges of G_f. By definition of linkage, the non-empty subsets of $\{1,2\}$ and $\{2,3,4\}$ are also hyperedges of G_f. Generally, if a set of vertices is a hyperedge of G_f, then any non-empty subset of the set is also a hyperedge of G_f. We call this property the *hierarchical property* among hyperedges. The linkage graph G_f has nine hyperedges: $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$, $\{1,2\}$, $\{2,3\}$, $\{2,4\}$, $\{3,4\}$, and $\{2,3,4\}$. There is no hyperedge containing 5 since f does not depend on x_5.

It is known that, for a k-bounded function with constant k, the problem of finding the Fourier coefficients is asymptotically equivalent to the problem of finding the linkage graph in terms of the number of function evaluations [HW03, HW04].

[1] An algorithm is called *adaptive* if the algorithm uses a sequence of queries in which some queries depend on the previous queries. Otherwise, it is called *non-adaptive*.

[2] To see a typical behavior of the complexity, we may consider the NK model [Kau89]. The NK model with parameters $N = n$ and $K = k - 1$ generates a class of k-bounded functions that are expressed as a sum of n subfunctions. When f is a function randomly generated from the NK model with parameters $N = n$ and $K = k - 1$ for constant k, it is not difficult to show that $r = \Theta(n)$, $B = \Omega(n)$, and $\theta = \mathcal{O}(1)$ with high probability. Thus, the query complexity of the algorithm [BJT04] for f is $\Omega\left(n^3\right)$ with high probability while the query complexity of our algorithm for f is $\mathcal{O}(n \log n)$.

[3] A hypergraph is k-bounded if the order of each hyperedge is at most k.

[4] The term linkage is from genetics and it means the interaction among the genes.

(The description of the asymptotic equivalence is provided in Section 2.) In a hypergraph, we say that a hyperedge *crosses among* certain disjoint sets of vertices if the number of the sets is equal to the order of the hyperedge and each of the sets contains exactly one vertex in the hyperedge. (By definition, a hyperedge of order one crosses among any set of vertices including the hyperedge.) Our main contribution is to show that, given a collection of disjoint sets of vertices, the existence of a hyperedge of the linkage graph crossing among those sets is testable with one-sided error by using a constant number of function evaluations.

Theorem 2 *Suppose that f is a k-bounded function defined on $\{0,1\}^n$ and $S_1, \ldots, S_j$ are j disjoint subsets of $[n]$. Then, we can use 2^j function evaluations of f to test the existence of a hyperedge in the linkage graph G_f crossing among S_i's, where the test result is correct with probability at least $\frac{1}{2^{2^k}}$ if such a hyperedge exists and it is correct with probability 1 otherwise.*

Theorem 2 is an extension of a previous theorem of Heckendorn and Wright [HW04] (Proposition 1 in Section 2), which holds only for the case when each of S_i's is a singleton set of vertices. To prove Theorem 2, we devise a random perturbation method for testing the existence of a hyperedge. It tests the existence of a hyperedge by flipping a randomly generated string at certain bit positions and evaluating the function values at the flipped strings. We obtain the desired result by analyzing the method. The analysis extensively uses the properties of basis functions in the Fourier transform of a k-bounded function.

Theorem 2 implies that the problem of finding the linkage graph of a k-bounded function is reduced to the following graph finding problem. Suppose that a hypergraph G has n vertices and m hyperedges and the hyperedges of G are unknown. A *cross-membership query* asks the existence of a hyperedge crossing among certain disjoint sets of vertices. We assume the *oracle with one-sided error* δ as follows. Given a cross-membership query, the oracle correctly answers with probability at least $1-\delta$ if the true answer for the query is YES and it correctly answers with probability 1 otherwise. The problem is to find the hyperedges of G by using as few queries to the oracle as possible.

In fact, it is enough for our purpose to consider the hypergraph finding problem for the k-bounded hypergraphs with the hierarchical property. Since we think that the problem is of self interest, however, we consider the problem for arbitrary k-bounded hypergraphs. We present an adaptive randomized algorithm for the problem to show

Theorem 3 *Suppose that G is an unknown k-bounded hypergraph with n vertices and m edges for constant k. Then, for any constant $0 \leq \delta < 1$, the hyperedges of G can be found with probability $1-\mathcal{O}\left(\frac{1}{n}\right)$ by using $\mathcal{O}(m \log n)$ cross-membership queries under the oracle with one-sided error δ. (The number of cross-membership queries is $2^{\mathcal{O}(k)}$ in k.)*

Our algorithm for Theorem 3 iteratively uses binary search to find the hyperedges. In this sense, it is analogous to the algorithm of Angluin and Chen [AC04, AC05, AC06] for the hypergraph finding problem with edge-detecting queries or to the algorithm of Reyzin and Srivastava [RS07] for the graph finding problem with edge-counting (or additive) queries.[5] On the other hand, since the answers of the oracle may contain errors in our situation, we need to handle the error bound more carefully, which is the main task in proving Theorem 3. A large deviation result for a sum of independent random variables with geometric distribution is crucially used for the task. (There have been a number of papers addressing the problem of finding a graph or a hypergraph by using various types of queries. For example, see [AA04, AA05, ABK$^+$02, ABK$^+$04, AC06, BAA$^+$01, BGK05, CK08, GK00].)

Theorem 1 is obtained from Theorems 2 and 3 and the equivalence between the problems of finding the Fourier coefficients and the linkage graph.

The remainder of the paper is organized as follows. In Section 2, we review some basic facts and previous results for the problem of finding the Fourier coefficients of k-bounded functions. In Section 3, we prove Theorem 2, which states the linkage testability of a linkage graph, by proving relevant lemmas. Section 4 deals with the graph finding problem with cross-membership queries under the probabilistic oracle as an independent problem. In the section, we give a randomized algorithm for the problem and analyze it to obtain Theorem 3. In Section 5, some remarks on the query and time complexity of the proposed algorithm are provided along with a factor of improving the complexity. Finally, concluding remarks closes the paper in Section 6.

2 Preliminaries

2.1 Linkage Test Function

Munetomo and Goldberg [MG99] proposed a perturbation method to test for the existence of linkage in a 2-subset of $[n]$. Given a 2-subset S and a string x, it checks the nonlinearity between the two bits in H by flipping the two bits of x individually and simultaneously and adding/subtracting the function values at the flipped strings. Heckendorn and Wright [HW04] generalized the method to detect linkage for subsets of any order. Suppose that f is a pseudo-Boolean function defined on $\{0,1\}^n$, S is a subset of $[n]$, and x is a string in $\{0,1\}^n$. They considered the *linkage test function* $\mathfrak{L}$ depending on f, S, and x as follows:

$$\mathfrak{L}(f,S,x) = \sum_{A \subseteq S} (-1)^{|A|} f\left(x \oplus 1_A\right).$$

Here, 1_A represents the string consisting of ones in the bit positions of A and zeros in the rest. For two strings $x, y \in \{0,1\}^n$, $x \oplus y$ means the bitwise addition modulo 2 of x and y. The linkage test function $\mathfrak{L}$ performs a series of function evaluations at x and the strings obtained by flipping x in order to detect the existence of the linkage among the bits in S. Heckendorn and Wright [HW04] proved the following theorem, which shows the usefulness of the linkage test function in finding hyperedges of G_f.

[5] An edge-detecting query asks the existence of an edge (or a hyperedge) in a set of vertices while an edge-counting query asks the number of edges (or hyperedges) in a set of vertices.

Proposition 1 *Suppose that f is a k-bounded function defined on $\{0,1\}^n$. Then, the followings hold:*
(a) *A subset S of $[n]$ is a hyperedge of G_f if and only if $\mathfrak{L}(f,S,x) \neq 0$ for some string $x \in \{0,1\}^n$.*
(b) *For a hyperedge S of order j in G_f, the probability that $\mathfrak{L}(f,S,x) \neq 0$ for a string x chosen uniformly at random from $\{0,1\}^n$ is at least $\frac{1}{2^{k-j}}$.*

Proposition 1 indicates that the linkage test function determines the existence of a hyperedge with one-sided error. Thus, by repeatedly evaluating the linkage test function for randomly chosen strings, we can make the error arbitrarily small. In particular, when k is a constant, this implies that a constant number of linkage tests (consequently, a constant number of function evaluations) is enough for determining the existence of a hyperedge with error probability at most a given constant. The hierarchical property among hyperedges implies that, for $j \geq 2$, a j-subset H can be a hyperedge only if every $(j-1)$-subset of H is a hyperedge. Based on this observation, Heckendorn and Wright [HW04] proposed a randomized algorithm that performs linkage test only for such a hyperedge candidate: The algorithm first detects the hyperedges of order one by investigating all the singleton subsets of $[n]$. Then, for j from 2 to k, it detects the hyperedges of order j by performing linkage test for the hyperedge candidates of order j that have been identified from the information of the hyperedges of lower order. Recently, the performance of the algorithm was fully analyzed by Choi *et al.* [CJM08]. Given a k-bounded function f with m hyperedges and a constant $\varepsilon > 0$, they showed that the algorithm finds the linkage graph G_f in $\mathcal{O}\left(\alpha(n,m)m\log n\right)$ function evaluations with error probability at most ε, where $\alpha(n,m)$ is between $n^{\frac{1}{2}}$ and n depending on m.

2.2 A Fourier Transform

Walsh transform is a Fourier transform for the space of pseudo-Boolean functions in which a pseudo-Boolean function is represented as a linear combination of 2^n basis functions called *Walsh functions* [Wal23]. For each subset H of $[n]$, the Walsh function corresponding to H, $\psi_H : \{0,1\}^n \to \mathbb{R}$, is defined as

$$\psi_H(x) = (-1)^{\sum_{i \in H} x[i]},$$

where $x[i]$ represents the i^{th} bit value in x. If we define an inner product of two pseudo-Boolean functions f and g as

$$\langle f, g \rangle = \sum_{x \in \{0,1\}^n} \frac{f(x) \cdot g(x)}{2^n},$$

the set of Walsh functions, $\{\psi_H \mid H \subseteq [n]\}$, becomes an orthonormal basis of the space of pseudo-Boolean functions. Hence, a pseudo-Boolean function f can be represented as

$$f = \sum_{H \subseteq [n]} \widehat{f}(H) \cdot \psi_H,$$

where $\widehat{f}(H) = \langle f, \psi_H \rangle$ is called the *Fourier coefficient* corresponding to H. Specifically, if $\widehat{f}(H) \neq 0$ and $\widehat{f}(H') = 0$ for any $H' \supsetneq H$, $\widehat{f}(H)$ is called a *maximal non-zero Fourier coefficient* of f. We refer to [HW99] for surveys of the properties of Walsh functions and Walsh transform in the space of pseudo-Boolean functions.

Heckendorn and Wright [HW04] provided a number of results to show the relation between the linkage test function and the Fourier coefficients. Some of them are summarized in the following proposition.

Proposition 2 *Suppose that f is a pseudo-Boolean function defined on $\{0,1\}^n$. Then, the followings hold:*
(a) *For a subset H of $[n]$, $\widehat{f}(H)$ is a maximal non-zero Fourier coefficient of f if and only if H is a maximal hyperedge of G_f.*
(b) *For a maximal hyperedge $H \subseteq [n]$,*

$$\widehat{f}(H) = \frac{\mathfrak{L}(f,H,0^n)}{2^{|H|}}.$$

(c) *For a subset H of $[n]$,*

$$\widehat{f}(H) = \frac{\mathfrak{L}(f,H,0^n)}{2^{|H|}} - \sum_{H \subsetneq H'} \widehat{f}(H').$$

(d) *For subsets H and H' of $[n]$ with $H \subseteq H'$,*

$$\mathfrak{L}(f,H',0^n) = \sum_{A \subseteq H' \setminus H} (-1)^{|A|} \mathfrak{L}(f,H,1_A).$$

Proposition 2 (a) says that the subsets of $[n]$ with maximal non-zero Fourier coefficients of f are the maximal hyperedges in the linkage graph of f. Thus, from Proposition 2 (b), the maximal non-zero Fourier coefficients of f are found by evaluating the linkage test function at the zero string for each maximal hyperedge. Once the maximal non-zero Fourier coefficients are found, the Fourier coefficients corresponding to the subsets of lower orders can be found by successively applying Proposition 2 (c). Proposition 2 (d) implies that no additional function evaluations are required for finding the Fourier coefficients corresponding to the subsets of lower orders. Hence, if f is k-bounded for constant k and m is the number of hyperedges in G_f, $\mathcal{O}(m)$ additional function evaluations are enough to find the Fourier coefficients of f. On the other hand, $\Omega\left(\frac{m \log n}{\log m}\right)$ function evaluations are required for finding the linkage graph of a k-bounded function for constant k as shown in [CJM08]. Thus, the problem of finding the Fourier coefficients of a k-bounded function for constant k is equivalent to the problem of finding the linkage graph in terms of the number of function evaluations required up to a constant factor.

3 Generalized Linkage Test

3.1 Generalized Linkage Test Function

Let f be a pseudo-Boolean function defined on $\{0,1\}^n$, $\mathcal{S}$ be a collection of disjoint subsets of $[n]$, and x be a string in $\{0,1\}^n$. We define the *generalized linkage test function* $\mathfrak{L}^*$ depending on f, $\mathcal{S}$, and x as follows:

$$\mathfrak{L}^*(f,\mathcal{S},x) = \sum_{\mathcal{S}' \subseteq \mathcal{S}} (-1)^{|\mathcal{S}'|} f\left(x \oplus \left(\bigoplus_{A \in \mathcal{S}'} 1_A\right)\right).$$

If we let $\mathcal{S}_H = \{\{a\} \mid a \in H\}$ for a subset H of $[n]$, we see that $\mathfrak{L}^*(f,\mathcal{S}_H,x) = \mathfrak{L}(f,H,x)$ for any $x \in \{0,1\}^n$.

The following lemmas describes the basic properties of the generalized linkage test function.

Lemma 4 *Suppose that $\mathcal{S}$ is a collection of disjoint subsets of $[n]$. Then, the followings hold:*
(a) (*Linearity*) *If $f_1, \ldots, f_\ell$ are pseudo-Boolean functions defined on $\{0,1\}^n$ and $c_1, \ldots, c_\ell$ are constants,*

$$\mathfrak{L}^*\left(\sum_{i=1}^{\ell} c_i f_i, \mathcal{S}, x\right) = \sum_{i=1}^{\ell} c_i \mathfrak{L}^*(f_i, \mathcal{S}, x)$$

for all $x \in \{0,1\}^n$.
(b) (*Recursion*) *If f is a pseudo-Boolean function defined on $\{0,1\}^n$,*

$$\mathfrak{L}^*(f, \mathcal{S}, x) = \mathfrak{L}^*(f, \mathcal{S} \setminus \{A\}, x) - \mathfrak{L}^*(f, \mathcal{S} \setminus \{A\}, x \oplus 1_A)$$

for any $A \in \mathcal{S}$ and any $x \in \{0,1\}^n$.

Proof: Omitted. ■

Lemma 5 *Suppose that f is a pseudo-Boolean function defined on $\{0,1\}^n$ and $\mathcal{S}$ is a collection of disjoint subsets of $[n]$. If the support set of f is disjoint with some $A \in \mathcal{S}$, $\mathfrak{L}^*(f, \mathcal{S}, x) = 0$ for all $x \in \{0,1\}^n$.*

Proof: Omitted. ■

3.2 Linkage Test Theorem

A collection of disjoint subsets of $[n]$, $\mathcal{R} = \{R_1, \ldots, R_j\}$, is called a *setwise subcollection* of $\mathcal{S}$ if $R_i \subseteq S_i$ for all $1 \leq i \leq j$. In this case, we denote by $\mathcal{R} \Subset \mathcal{S}$. Note that a setwise subcollection $\mathcal{R}$ of $\mathcal{S}$ is allowed to contain multiple empty sets from the definition. We consider a random model $\Gamma(\mathcal{S})$ that generates a setwise subcollection of $\mathcal{S}$ as follows: For each $S_i \in \mathcal{S}$, we select each element in S_i independently and with probability $\frac{1}{2}$ and put it into R_i. Then, we build a setwise subcollection $\mathcal{R}$ of $\mathcal{S}$ by letting $\mathcal{R} = \{R_i \mid 1 \leq i \leq j\}$. In the following, $\mathsf{str}(\mathcal{R})$ denotes the set of the strings, x's, such that x has the same bit value in the bit positions in R_i for all $1 \leq i \leq j$:

$$\mathsf{str}(\mathcal{R}) = \{x \in \{0,1\}^n \mid x[a] = x[b] \text{ for all } a, b \text{ such that } a, b \in R_i \text{ for some } i \text{ with } 1 \leq i \leq j\}.$$

Theorem 6 *Suppose that f is a k-bounded function and $\mathcal{S}$ is a collection of disjoint subsets of $[n]$. Then, the followings hold:*
(a) The linkage graph G_f contains a hyperedge crossing among $\mathcal{S}$ if and only if there exist $\mathcal{R} \Subset \mathcal{S}$ and $x \in \mathsf{str}(\mathcal{R})$ such that $\mathfrak{L}^*(f, \mathcal{R}, x) \neq 0$.
(b) If G_f contains a hyperedge crossing among $\mathcal{S}$, the probability that $\mathfrak{L}^*(f, \mathcal{R}, x) \neq 0$ for a randomly generated $\mathcal{R}$ from $\Gamma(\mathcal{S})$ and a string x chosen uniformly at random from $\mathsf{str}(\mathcal{R})$ is at least $\frac{1}{2^{2k}}$.

Theorem 6 implies Theorem 2 and provides an efficient method to test for the existence of a hyperedge crossing among a given collection of sets of vertices in the linkage graph.

Proof: Since (b) implies the only-if part of (a), we first prove the if part of (a) and then prove (b).

Suppose that G_f does not contain any hyperedge crossing among $\mathcal{S}$. Let $\mathcal{R}$ be a setwise subcollection of $\mathcal{S}$ and let x be a string in $\{0,1\}^n$. Since G_f does not contain any hyperedge crossing among $\mathcal{S}$, G_f does not contain any hyperedge crossing among $\mathcal{R}$. This implies that $\widehat{f}(H) = 0$ for all H such that $H \cap A \neq \emptyset$ for all $A \in \mathcal{R}$, by definition of G_f and Proposition 2. Thus, by Lemma 4 (a),

$$\mathfrak{L}^*(f, \mathcal{R}, x) = \sum_H \widehat{f}(H) \mathfrak{L}^*\left(\psi_H, \mathcal{R}, x\right),$$

where the summation is over the subsets, H's, such that $H \cap A = \emptyset$ for some $A \in \mathcal{R}$. Since the support set of ψ_H is H for any $H \subseteq [n]$, $\mathfrak{L}^*\left(\psi_H, \mathcal{R}, x\right) = 0$ for H's in the summation by Lemma 5 and so $\mathfrak{L}^*(f, \mathcal{R}, x) = 0$.

Now, consider the proof of (b). Let $\mathcal{S} = \{S_1, \ldots, S_j\}$. For a setwise subcollection $\mathcal{R}$ of $\mathcal{S}$, let $\mathcal{R} = \{R_1, \ldots, R_j\}$, where $R_i \subseteq S_i$ for all $1 \leq i \leq j$. Let $r_i = |R_i|$ and $r = \sum_{i=1}^{j} r_i$. For each R_i, set a distinct bit position $a_i \in [n - r + j]$ and, for each $i' \in [n] \setminus (\bigcup_i R_i)$, set a distinct bit position $b_{i'} \in [n - r + j]$. For each $H \subseteq [n]$, define $\varphi_{H,\mathcal{R}} : \{0,1\}^{n-r+j} \to \mathbb{R}$ as follows: If $|H \cap R_i|$ is odd for all $1 \leq i \leq j$,

$$\varphi_{H,\mathcal{R}}(y) = (-1)^{\sum_{i=1}^{j} y[a_i] + \sum_{i' \in H \setminus (\bigcup_i R_i)} y[b_{i'}]}$$

for any $y \in \{0,1\}^{n-r+j}$. Otherwise, $\varphi_{H,\mathcal{R}}$ is the zero function that assigns zero value to all input strings $y \in \{0,1\}^{n-r+j}$. For each $x \in \mathsf{str}(\mathcal{R})$, assign the string $y_{x,\mathcal{R}} \in \{0,1\}^{n-r+j}$ such that $y_{x,\mathcal{R}}[a_i] = x[a]$ for some $a \in R_i$ for all $1 \leq i \leq j$ and $y_{x,\mathcal{R}}[b_{i'}] = x[i']$ for all $i' \in [n] \setminus (\bigcup_i R_i)$. Note that $\{y_{x,\mathcal{R}} | x \in \mathsf{str}(\mathcal{R})\} = \{0,1\}^{n-r+j}$ and the sets $\mathsf{str}(\mathcal{R})$ and $\{y_{x,\mathcal{R}} | x \in \mathsf{str}(\mathcal{R})\}$ are in one-to-one correspondence. Letting $\mathcal{S}_\mathcal{R} = \{\{a_i\} \mid 1 \leq i \leq j\}$, we have

Claim 7 *For any $H \subseteq [n]$,*

$$\mathfrak{L}^*(\psi_H, \mathcal{R}, x) = \mathfrak{L}^*(\varphi_{H,\mathcal{R}}, \mathcal{S}_\mathcal{R}, y_{x,\mathcal{R}})$$

for all $x \in \mathsf{str}(\mathcal{R})$.

Proof: Suppose that $|H \cap R_i|$ is odd for all $1 \leq i \leq j$. Let u_i be a bit position in $H \cap R_i$ for $1 \leq i \leq j$. For all $x \in \mathsf{str}(\mathcal{R})$, $\sum_{u \in H \cap R_i} x[u] = |H \cap R_i| \cdot x[u_i] = x[u_i] \pmod 2$ for all $1 \leq i \leq j$ and so

$$\begin{aligned}
\psi_H(x) &= (-1)^{\sum_i \sum_{u \in H \cap R_i} x[u] + \sum_{i' \in H \setminus (\bigcup_i R_i)} x[i']} \\
&= (-1)^{\sum_i x[u_i] + \sum_{i' \in H \setminus (\bigcup_i R_i)} x[i']} \\
&= (-1)^{\sum_i y_{x,\mathcal{R}}[a_i] + \sum_{i' \in H \setminus (\bigcup_i R_i)} y_{x,\mathcal{R}}[b_{i'}]} \\
&= \varphi_{H,\mathcal{R}}(y_{x,\mathcal{R}}).
\end{aligned}$$

Let $x_{\mathcal{R}'} = x \oplus \left(\bigoplus_{A \in \mathcal{R}'} 1_A\right)$ for $\mathcal{R}' \subseteq \mathcal{R}$. If $x \in \mathsf{str}(\mathcal{R})$, $x_{\mathcal{R}'} \in \mathsf{str}(\mathcal{R})$ and $y_{x_{\mathcal{R}'},\mathcal{R}} = y_{x,\mathcal{R}} \oplus \left(\bigoplus_{i : R_i \in \mathcal{R}'} 1_{\{a_i\}}\right)$.

Hence, for all $x \in \mathsf{str}(\mathcal{R})$,

$$\begin{aligned}
&\mathfrak{L}^*(\psi_H, \mathcal{R}, x) \\
&= \sum_{\mathcal{R}' \subseteq \mathcal{R}} (-1)^{|\mathcal{R}'|} \psi_H \left(x \oplus \left(\bigoplus_{A \in \mathcal{R}'} 1_A \right) \right) \\
&= \sum_{\mathcal{R}' \subseteq \mathcal{R}} (-1)^{|\mathcal{R}'|} \psi_H \left(x_{\mathcal{R}'} \right) \\
&= \sum_{\mathcal{R}' \subseteq \mathcal{R}} (-1)^{|\mathcal{R}'|} \varphi_{H,\mathcal{R}} \left(y_{x_{\mathcal{R}'},\mathcal{R}} \right) \\
&= \sum_{\mathcal{R}' \subseteq \mathcal{R}} (-1)^{|\{a_i | R_i \in \mathcal{R}'\}|} \varphi_{H,\mathcal{R}} \left(y_{x,\mathcal{R}} \oplus \left(\bigoplus_{i:R_i \in \mathcal{R}'} 1_{\{a_i\}} \right) \right) \\
&= \sum_{\mathcal{S}' \subseteq \mathcal{S}_{\mathcal{R}}} (-1)^{|\mathcal{S}'|} \varphi_{H,\mathcal{R}} \left(y_{x,\mathcal{R}} \oplus \left(\bigoplus_{B \in \mathcal{S}'} 1_B \right) \right) \\
&= \mathfrak{L}^*(\varphi_{H,\mathcal{R}}, \mathcal{S}_{\mathcal{R}}, y_{x,\mathcal{R}}).
\end{aligned}$$

Now, suppose that $|H \cap R_i|$ is even for some i. For all $x \in \{0,1\}^n$, $\mathfrak{L}^*(\psi_H, \mathcal{R} \setminus \{R_i\}, x) = \mathfrak{L}^*(\psi_H, \mathcal{R} \setminus \{R_i\}, x \oplus 1_{R_i})$ and so $\mathfrak{L}^*(\psi_H, \mathcal{R}, x) = 0$ by Lemma 4 (b). Since $\varphi_{H,\mathcal{R}}$ is the zero function, on the other hand, $\mathfrak{L}^*(\varphi_{H,\mathcal{R}}, \mathcal{S}_{\mathcal{R}}, y) = 0$ for all $y \in \{0,1\}^{n-r+j}$. Hence,

$$\mathfrak{L}^*(\psi_H, \mathcal{R}, x) = \mathfrak{L}^*(\varphi_{H,\mathcal{R}}, \mathcal{S}_{\mathcal{R}}, y_{x,\mathcal{R}})$$

for all $x \in \mathsf{str}(\mathcal{R})$. ■

Define the pseudo-Boolean function $g_{f,\mathcal{R}} : \{0,1\}^{n-r+j} \to \mathbb{R}$ by

$$g_{f,\mathcal{R}} = \sum_{H \subseteq [n]} \widehat{f}(H) \cdot \varphi_{H,\mathcal{R}}.$$

Claim 8 *For all $x \in \mathsf{str}(\mathcal{R})$,*

$$\mathfrak{L}^*(f, \mathcal{R}, x) = \mathfrak{L}^*(g_{f,\mathcal{R}}, \mathcal{S}_{\mathcal{R}}, y_{x,\mathcal{R}}).$$

Proof: By Lemma 4 (a) and Claim 7,

$$\begin{aligned}
\mathfrak{L}^*(f, \mathcal{R}, x) &= \sum_{H \subseteq [n]} \widehat{f}(H) \cdot \mathfrak{L}^*(\psi_H, \mathcal{R}, x) \\
&= \sum_{H \subseteq [n]} \widehat{f}(H) \cdot \mathfrak{L}^*(\varphi_{H,\mathcal{R}}, \mathcal{S}_{\mathcal{R}}, y_{x,\mathcal{R}}) \\
&= \mathfrak{L}^*(g_{f,\mathcal{R}}, \mathcal{S}_{\mathcal{R}}, y_{x,\mathcal{R}}),
\end{aligned}$$

for all $x \in \mathsf{str}(\mathcal{R})$. ■

Suppose that G_f contains a hyperedge crossing among $\mathcal{S}$.

Claim 9 *Suppose that a setwise subcollection $\mathcal{R}$ is randomly generated from $\Gamma(\mathcal{S})$. Then, the probability that the linkage graph of $g_{f,\mathcal{R}}$ has the hyperedge crossing among $\mathcal{S}_{\mathcal{R}}$ is at least $\frac{1}{2^{k+j}}$.*

Proof: Since G_f contains a hyperedge crossing among $\mathcal{S}$, there exist subsets H's such that $\widehat{f}(H) \neq 0$ and $H \cap S_i \neq \emptyset$ for all $S_i \in \mathcal{S}$. Among those subsets, we choose a maximal subset H^* in viewpoint of the size of intersection with S_i's: For each $1 \leq i \leq j$, $|H^* \cap S_i| \geq |H \cap S_i|$ for any H such that $\widehat{f}(H) \neq 0$, $H \cap S_i \neq \emptyset$ for all $S_i \in \mathcal{S}$, and $|H \cap S_l| = |H^* \cap S_l|$ for all $1 \leq l \leq i-1$. Let A_i be a set consisting of an element in $H^* \cap S_i$ and let $B_i = (H^* \cap S_i) \setminus A_i$. Let $\mathcal{R} = \{R_1, \ldots, R_j\}$, where $R_i \subseteq S_i$ for all $1 \leq i \leq j$. Since $A_i \cup B_i = H^* \cap S_i$ and $\sum_i |A_i \cup B_i| = \sum_i |H^* \cap S_i| \leq |H^*| \leq k$, the probability that $R_i \supseteq A_i$ and $R_i \not\supseteq B_i$ for all $1 \leq i \leq j$ is at least $\frac{1}{2^k}$.

Consider the condition that $R_i \supseteq A_i$ and $R_i \not\supseteq B_i$ for all $1 \leq i \leq j$. Denote

$$\begin{aligned}
\mathcal{H}^* = \{H \subseteq [n] \mid \widehat{f}(H) \neq 0,& \\
& H \supseteq (\cup_i B_i) \cup (H^* \setminus (\cup_i S_i)), \\
& \text{and } |H \cap S_i| = |H^* \cap S_i| \text{ for all } i\}.
\end{aligned}$$

It is clear that $H^* \in \mathcal{H}^*$. Given the condition, if $\varphi_{H,\mathcal{R}} = \varphi_{H^*,\mathcal{R}}$, H should be in $\mathcal{H}^*$. Thus, in the Walsh transform of $g_{f,\mathcal{R}}$, the Walsh coefficient corresponding to the Walsh function $\varphi_{H^*,\mathcal{R}}$ is equal to $\sum_H \widehat{f}(H)$, where the summation is over H's such that $H \in \mathcal{H}^*$ and $(H \cap S_i \setminus B_i) \subseteq R_i$ for all i. Since H^* was chosen in a maximal sense as mentioned, for any $H \in \mathcal{H}^*$, $|H \cap S_i \setminus B_i| = 1$ for all $1 \leq i \leq j$. Thus, when we choose each element in $S_i \setminus (A_i \cup B_i)$ independently and with probability $\frac{1}{2}$ and put it into R_i, the conditional probability that $\sum_H \widehat{f}(H) \neq 0$, where the summation is over H's such that $H \in \mathcal{H}^*$ and $(H \cap S_i \setminus B_i) \subseteq R_i$ for all i, is at least $\frac{1}{2^j}$. In this case, $\varphi_{H^*,\mathcal{R}}$ may be expressed as $\psi_{H'}$ for $H' \subseteq [n-r+j]$ such that

$$H' = \{a_i \mid 1 \leq i \leq j\} \cup \{b_{i'} \mid i' \in (\cup_i B_i) \cup (H^* \setminus (\cup_i S_i))\}$$

and the Walsh coefficient corresponding to $\psi_{H'}$ in the Walsh transform of $g_{f,\mathcal{R}}$ is non-zero. At this time, the linkage graph of $g_{f,\mathcal{R}}$ has the j-hyperedge crossing among $\mathcal{S}_{\mathcal{R}} = \{\{a_i\} \mid 1 \leq i \leq j\}$.

Therefore, the probability that the linkage graph of $g_{f,\mathcal{R}}$ has the hyperedge crossing among $\mathcal{S}_{\mathcal{R}}$ for a setwise subcollection $\mathcal{R}$ randomly generated from $\Gamma(\mathcal{S})$ is at least $\frac{1}{2^{k+j}}$ and the proof is completed. ■

Since f is a k-bounded function, $g_{f,\mathcal{R}}$ is also k-bounded. Thus, when the linkage graph of $g_{f,\mathcal{R}}$ has the hyperedge crossing among $\mathcal{S}_{\mathcal{R}}$, the probability that $\mathfrak{L}^*(g_{f,\mathcal{R}}, \mathcal{S}_{\mathcal{R}}, y) \neq 0$ for a string y chosen uniformly at random from $\{0,1\}^{n-r+j}$ is at least $\frac{1}{2^{k-j}}$ by Proposition 1 (b). Hence, by Claim 9, the probability that $\mathfrak{L}^*(g_{f,\mathcal{R}}, \mathcal{S}_{\mathcal{R}}, y) \neq 0$ for a setwise subcollection $\mathcal{R}$ randomly generated from $\Gamma(\mathcal{S})$ and a string y chosen uniformly at random from $\{0,1\}^{n-r+j}$ is at least $\frac{1}{2^{2k}}$. Since the sets $\mathsf{str}(\mathcal{R})$ and $\{0,1\}^{n-r+j} = \{y_{x,\mathcal{R}} \mid x \in \mathsf{str}(\mathcal{R})\}$ are in one-to-one correspondence, we have the part (b) of the theorem by Claim 8. ■

4 Finding Graphs with Cross-Membership Queries

In this section, we focus on the problem to find an unknown hypergraph with cross-membership queries under the oracle with one-sided error δ. Recall that, given a cross-membership query, the oracle with one-sided error δ correctly answers with probability at least $1-\delta$ if the true answer for the query is YES and it correctly answers with probability 1 otherwise. Section 4.1 presents a randomized algorithm for the graph finding problem. The algorithm is analyzed in Section 4.2, which induces Theorem 3.

GRAPHFINDINGALGORITHM(n,k,δ)
// E_j : *the set of the hyperedges of order j found so far*
// Q : *the set of the vertices in the hyperedges of order j found so far*
// W : *the set of the vertices v such that all the hyperedges of order j containing v have been found by the algorithm*
for j from 1 to k
$Q \leftarrow \emptyset$, $W \leftarrow \emptyset$;
$E_j \leftarrow \emptyset$;
repeat
$(S_i)_{i=1}^{j} \leftarrow$ CHECKEXISTENCE($\emptyset$,W,j);
if $(S_i)_{i=1}^{j} =$ NULL, break;
$v \leftarrow$ BINARYSEARCH($(S_i)_{i=1}^{j}$,1);
$Q \leftarrow Q \cup \{v\}$;
while $Q \setminus W \neq \emptyset$
choose a vertex v in $Q \setminus W$;
$E_{v,j} \leftarrow$ FINDHYPEREDGES($\{v\}$,W,j);
$E_j \leftarrow E_j \cup E_{v,j}$;
$Q \leftarrow Q \cup \left(\bigcup_{H \in E_{v,j}} H\right)$;
$W \leftarrow W \cup \{v\}$;
$E \leftarrow \bigcup_{j=1}^{k} E_j$;
return E;

Figure 1: Main procedure of the algorithm GFA (The output of GFA is the set of the hyperedges of the input graph that have been found. For the subprocedures, CHECKEXISTENCE, BINARYSEARCH, and FINDHYPEREDGES, see Figures 2, 3, and 4, respectively.)

4.1 Algorithm for Finding Graphs

In this section, we present the algorithm to find an unknown hypergraph with cross-membership queries under the oracle with one-sided error δ, the *Graph Finding Algorithm* (GFA). The algorithm GFA takes three arguments: The number of vertices of the unknown hypergraph n, the order of the hypergraph k, and the error bound for the answer of the oracle $0 \leq \delta < 1$. It returns the set of the hyperedges of the hypergraph that have found. The algorithm GFA consists of the main procedure GRAPHFINDINGALGORITHM (Figure 1) and the three subprocedures CHECKEXISTENCE (Figure 2), BINARYSEARCH (Figure 3), and FINDHYPEREDGES (Figure 4). In the pseudocode, the values of n, k, and δ can be accessed by any procedure. All other variables are local to the given procedure.

Suppose that G is an unknown hypergraph given to GFA and let G_j be the induced subgraph of G consisting of the hyperedges of order j for $1 \leq j \leq k$. The algorithm GFA successively finds the hyperedges of G_1, G_2, and so on. After the algorithm finally finds the hyperedges of G_k, it returns all the hyperedges found so far. To find the hyperedges of G_j for $j = 1, \ldots, k$, the algorithm iteratively checks whether there is a hyperedge of order j that has not been found and, if such a hyperedge exists, the algorithm finds all the hyperedges in the connected component that the hyperedge belongs to. It continues this process until there is no more hyperedge that can be found.

In the main procedure GRAPHFINDINGALGORITHM, the variable Q contains the vertices in the hyperedges found so far. The variable W contains the vertices v such that all the hyperedges of order j containing v have been found by the algorithm. The variable E_j contains the hyperedges of order j found so far. To check the existence of a new connected component of two or more vertices in the subgraph consisting of the hyperedges of order j, GRAPHFINDINGALGORITHM calls the subprocedure CHECKEXISTENCE.

Given sets of vertices U and W and a positive integer j, the procedure CHECKEXISTENCE performs a randomized test for whether there is a hyperedge of order j that contains all the vertices in U and does not contain the vertices in W. For the purpose, it iteratively generates a collection of disjoint sets of vertices $(S_i)_{i=1}^{j}$ for a cross-membership query as follows. Letting $U = \{v_1, \ldots, v_{|U|}\}$, the set S_i is fixed with $S_i = \{v_i\}$ for $1 \leq i \leq |U|$. The sets $S_{|U|+1}, \ldots, S_j$ are generated as a uniform random partition of vertices in $[n] \setminus (U \cup W)$. If the oracle answers YES for the cross-membership query with some $(S_i)_{i=1}^{j}$, there is a hyperedge of order j crossing among S_i's, which contains the vertices in U and does not contain the vertices in W. In this case, CHECKEXISTENCE returns the generated sets $(S_i)_{i=1}^{j}$. If the oracle answers NO for all the generated collections of disjoint sets, CHECKEXISTENCE returns NULL regarding that there is no such a hyperedge.

If CHECKEXISTENCE returns NULL, GRAPHFINDINGALGORITHM regards that there is no hyperedge of order j and continues to find the hyperedges of order $j + 1$. If CHECKEXISTENCE returns a (non-NULL) collection of disjoint sets of vertices, this implies that there is a hyperedge of order j. To find a vertex in the hyperedge, GRAPHFINDINGALGORITHM calls the subprocedure BINARYSEARCH. Given a collection of disjoint sets of vertices $(S_i)_{i=1}^{j}$ and a positive integer r between 1 and j, the procedure BINARY-

CHECKEXISTENCE(U,W,j)
label the vertices in U as $v_1, \ldots, v_{|U|}$;
for i from 1 to $|U|$
$S_i \leftarrow \{v_i\}$;
for i from $|U|+1$ to j
$S_i \leftarrow \emptyset$;
repeat $\lceil \frac{e^j\sqrt{j+1}}{1-\delta} \log n \rceil$ times
for each $v \in [n] \setminus (U \cup W)$
choose i uniformly at random from $\{|U|+1, \ldots, j\}$;
$S_i \leftarrow S_i \cup \{v\}$;
if CMQ($S_1, \ldots, S_j$) = YES
return $(S_i)_{i=1}^j$;
return NULL;

Figure 2: Procedure to check the existence of a hyperedge of order j that contains all the vertices in U and does not contain the vertices in W (Here, CMQ($(S_i)_{i=1}^j$) is the answer of the oracle for the cross-membership query $(S_i)_{i=1}^j$.)

BINARYSEARCH($(S_i)_{i=1}^j$,r)
if $|S_r| = 1$, return the vertex in S_r;
repeat $\lceil \frac{6(j+1)}{1-\delta} \log n \rceil$ times
choose a subset S'_r of S_r uniformly at random among the subsets of order $\lfloor \frac{|S_r|}{2} \rfloor$;
if CMQ($S_1, \ldots, S_{r-1}, S'_r, S_{r+1}, \ldots, S_j$) = YES,
$S_r \leftarrow S'_r$;
if $|S_r| = 1$, return the vertex in S_r;
return a vertex in S_r;

Figure 3: Procedure to search a vertex in S_r that is contained in a hyperedge of order j crossing among $S_1, \ldots, S_j$ (Here, CMQ($(S_i)_{i=1}^j$) is the answer of the oracle for the cross-membership query $(S_i)_{i=1}^j$.)

SEARCH returns a vertex that is in S_r and in one of the hyperedges crossing among S_i's. Among the subsets of S_r of order $\lfloor \frac{|S_r|}{2} \rfloor$, it chooses a subset S'_r uniformly at random. For the sets of vertices $(S_i)_{i=1}^j$ in which S_r is replaced with S'_r, it asks the cross-membership query to check whether there is a hyperedge crossing among the sets. If the answer of the oracle is YES, i.e., if it turns out that there is a hyperedge crossing among the sets, it replaces S_r with S'_r. The procedure BINARYSEARCH repeats this process at most a specified number of times until there remains one vertex in S_r. If there remains one vertex in S_r before the specified number of iterations, BINARYSEARCH returns the vertex. Otherwise, it fails to exactly search the desired vertex and returns an arbitrary vertex in S_r.

Once a vertex in the new connected component is found by BINARYSEARCH, GRAPHFINDINGALGORITHM puts the vertex into Q and repeats the following process while $Q \setminus W \neq \emptyset$. It chooses a vertex v in $Q \setminus W$ and finds all the hyperedges of order j containing v by calling the subprocedure FINDHYPEREDGES. Given two sets of vertices U and W and a positive integer j, FINDHYPEREDGES returns the set of the hyperedges of order j that contain the vertices in U and do not contain the vertices in W. In the procedure FINDHYPEREDGES, the variable A contains the vertices such that the desired hyperedges of order j containing the vertices in A have been found. Initially, A is set to be empty. If $|U| = j$, U is the only hyperedge of order j containing the vertices in U and FINDHYPEREDGES returns the set consisting of U. Otherwise, it recursively finds the desired hyperedges of order j as follows. By calling CHECKEXISTENCE, It first checks whether there is a hyperedge of order j that contains the vertices in U and does not contain the vertices in W. If CHECKEXISTENCE returns NULL, FINDHYPEREDGES regards that there is no such a hyperedge and returns the set of the hyperedges found so far. Otherwise, it chooses a vertex v in the hyperedge by calling BINARYSEARCH. Then, it finds the hyperedges of order j that contain the vertices in $U \cup \{v\}$ and does not contain the vertices in $W \cup A$ by calling FINDHYPEREDGES recursively. After that, it puts v into A and continues to find the desired hyperedges of order j not containing the vertices in A.

After all the hyperedges of order j containing v are found, they are put into E_j. The vertices contained in the hyperedges are put into Q to mark that they are in the connected component being searched. The vertex v is put into W to prevent the hyperedges of order j containing v from being searched again.

4.2 Algorithm Analysis

In this section, we analyze the algorithm GFA to obtain Theorem 3. We first analyze the number of cross-membership

FINDHYPEREDGES(U,W,j)
  if $|U| = j$, return $\{U\}$;
  $E_{U,j} \leftarrow \emptyset$, $A \leftarrow \emptyset$;
  repeat
    $(S_i)_{i=1}^{j} \leftarrow$ CHECKEXISTENCE(U,$W \cup A$,j);
    if $(S_i)_{i=1}^{j} =$ NULL, break;
    $v \leftarrow$ BINARYSEARCH($(S_i)_{i=1}^{j}$,$|U|+1$);
    $E_{U,j} \leftarrow E_{U,j} \cup$ FINDHYPEREDGES($U \cup \{v\}$,$W \cup A$,j);
    $A \leftarrow A \cup \{v\}$;
  return $E_{U,j}$;

Figure 4: Procedure to find the hyperedges of order j that contain all the vertices in U and do not contain the vertices in W

queries used in GFA.

Lemma 10 *Suppose that G is an unknown k-bounded hypergraph with n vertices and m hyperedges for constant k. Then, for any constant $0 \leq \delta < 1$, GFA uses $\mathcal{O}(m \log n)$ cross-membership queries for G under the oracle with one-sided error δ.*

Proof: Omitted. ■

To analyze the error probability of GFA, we need a large deviation result for a sum of independent random variables following geometric distributions. A random variable X follows the geometric distribution with parameter p if, for a coin of which HEAD appears with probability p, X is the number of coin tosses until the first HEAD appears. It is easy to show that the expectation of X is $\frac{1}{p}$. We obtain the desired result by using the Chernoff bound as follows [Che52, MR95].

Proposition 3 *Suppose that, for some $0 < p \leq 1$, $X_1, \ldots, X_\ell$ are independent random variables such that $\Pr[X_i = 1] = p$ and $\Pr[X_i = 0] = 1 - p$ for all $1 \leq i \leq \ell$. Let $X = \sum_{i=1}^{\ell} X_i$. Then, for any $0 \leq \alpha < 1$,*

$$\Pr\left[X \leq (1-\alpha)\mathrm{E}[X]\right] \leq \exp\left(-\frac{\mathrm{E}[X]\alpha^2}{2}\right).$$

Now, we present the result for a sum of independent random variables following geometric distributions.

Lemma 11 *Suppose that, for some $0 < p \leq 1$, $X_1, \ldots, X_\ell$ are independent random variables each of which follows the geometric distribution with parameter p. Let $X = \sum_{i=1}^{\ell} X_i$. Then, for any $\alpha > 0$,*

$$\Pr\left[X > (1+\alpha)\mathrm{E}[X]\right] \leq \exp\left(-\frac{\alpha^2 \ell}{2(1+\alpha)}\right).$$

Proof: Omitted. ■

Lemma 12 *Suppose that G is an unknown k-bounded hypergraph with n vertices and m hyperedges for constant k. Then, for any $0 \leq \delta < 1$, GFA correctly finds the hyperedges of G with probability $1 - \mathcal{O}\left(\frac{1}{n}\right)$ under the oracle with one-sided error δ.*

Proof: We will show that the probability that GFA does not find all the hyperedges of G_j is $\mathcal{O}\left(\frac{1}{n}\right)$ for each j with $1 \leq j \leq k$. Then, the lemma follows by the union bound.

We first consider the probability that CHECKEXISTENCE performs incorrectly for given arguments U, W, and j. Suppose that there is no hyperedge of order j in G that contains the vertices in U and does not contain the vertices in W. In this case, CHECKEXISTENCE returns NULL and the probability of CHECKEXISTENCE being incorrect is zero. Suppose that there is a hyperedge of order j in G that contain the vertices in U and does not contain the vertices in W. Let $U = \{v_1, \ldots, v_{|U|}\}$ and let the hyperedge of order j be $\{v_1, \ldots, v_{|U|}, v_{|U|+1}, \ldots, v_j\}$. The probability that $v_{|U|+1}, \ldots, v_j$ are put into different S_i's is $\frac{(j-|U|)!}{(j-|U|)^{j-|U|}}$. When $v_{|U|+1}, \ldots, v_j$ are put into different S_i's, the probability that the oracle answers YES for the cross-membership query $(S_i)_{i=1}^{j}$ is at least $1-\delta$. Thus, for each iteration of the repeat loop in CHECKEXISTENCE, the probability that the hyperedge is not detected is at most $1 - \frac{(j-|U|)!}{(j-|U|)^{j-|U|}}(1-\delta)$. Hence, the probability that the hyperedge is not detected for $\lceil \frac{e^j\sqrt{j+1}}{1-\delta} \log n \rceil$ iterations of the repeat loop is at most

$$\left(1 - \frac{(j-|U|)!}{(j-|U|)^{j-|U|}}(1-\delta)\right)^{\frac{e^j\sqrt{j+1}}{1-\delta}\log n}.$$

By using the fact that $1 - x \leq e^{-x}$ for any real x, this value is at most

$$\exp\left(-\frac{(j-|U|)!e^j\sqrt{j+1}}{(j-|U|)^{j-|U|}}\log n\right).$$

After some calculation using the facts that $\frac{(j-|U|)!}{(j-|U|)^{j-|U|}} \geq \frac{j!}{j^j}$ and $j! > \sqrt{2\pi j}\left(\frac{j}{e}\right)^j e^{\frac{1}{12j+1}}$, we have

$$\begin{aligned}
&\exp\left(-\frac{(j-|U|)!e^j\sqrt{j+1}}{(j-|U|)^{j-|U|}}\log n\right) \\
&\quad \leq \exp\left(-(j+1)\log n\right) \\
&\quad = \frac{1}{n^{j+1}}.
\end{aligned}$$

Thus, the probability of CHECKEXISTENCE being incorrect is at most $\frac{1}{n^{j+1}}$.

Now, we bound the probability that BINARYSEARCH performs incorrectly for given arguments $(S_i)_{i=1}^{j}$ and r. To

this end, we consider an imaginary procedure BS' that is the same as BINARYSEARCH except that, in the procedure BS', the repeat loop continues until the size of S_r becomes one. In the repeat loop of BS', S_r is iteratively halved and updated. Suppose that the size of S_r becomes one after S_r is halved and updated t times. For $1 \leq i \leq t$, let X_i be the number of iterations of the repeat loop between the $(i-1)^{\text{th}}$ update and the i^{th} update of S_r. Let v be a vertex of a hyperedge crossing among S_i's that is in the initial S_r. When v is in the $(i-1)$ times updated S_r, the probability that v is chosen as an element of S'_r is at least $\frac{1}{3}$. (The extreme case is when the order of S_r is three.) Thus, X_i follows a geometric distribution with the parameter at least $\frac{1}{3}(1-\delta)$. If we let $X = \sum_{i=1}^{t} X_i$, by linearity of expectation,

$$\mathrm{E}[X] \leq \frac{3t}{1-\delta}.$$

Thus,

$$\begin{aligned}
&\Pr\left[X > \frac{6(j+1)}{1-\delta}\log n\right] \\
&\quad = \Pr\left[X > \left(\frac{2(j+1)\log n}{t}\right)\left(\frac{3t}{1-\delta}\right)\right] \\
&\quad \leq \Pr\left[X > \left(\frac{2(j+1)\log n}{t}\right)\mathrm{E}[X]\right].
\end{aligned}$$

Since X_i's are independent, letting $1+\alpha = \frac{2(j+1)\log n}{t}$, we apply Lemma 11 to the above inequality to obtain

$$\begin{aligned}
\Pr\left[X > \frac{6(j+1)}{1-\delta}\log n\right] &\leq \exp\left(-\frac{\alpha^2 t}{2(1+\alpha)}\right) \\
&\leq \exp\left(-(j+1)\log n\right) \\
&= \frac{1}{n^{j+1}}.
\end{aligned}$$

Thus, the probability of BINARYSEARCH performing incorrectly is at most $\frac{1}{n^{j+1}}$ as it is at most the probability of X being more than $\lceil \frac{6(j+1)}{1-\delta}\log n \rceil$.

The number of CHECKEXISTENCE and BINARYSEARCH being called for GFA to find the hyperedges of G_j are at most $j^2 m$, respectively. Thus, in the process of GFA finding the hyperedges of G_j, the probability that CHECKEXISTENCE or BINARYSEARCH incorrectly perform once or more times is at most $\frac{2j^2 m}{n^{j+1}} \leq \frac{2j^2 n^j}{n^{j+1}} = \frac{2j^2}{n}$, which is $\mathcal{O}\left(\frac{1}{n}\right)$ since $j \leq k$ for constant k. This means that, with probability $1 - \mathcal{O}\left(\frac{1}{n}\right)$, CHECKEXISTENCE and BINARYSEARCH performs correctly throughout the process of GFA finding the hyperedges of G_j.

Suppose the condition that CHECKEXISTENCE and BINARYSEARCH correctly perform throughout the process of GFA finding the hyperedges of G_j. We show that, given U, W, and j, FINDHYPEREDGES correctly return the set of the hyperedges of order j containing the vertices in U and not containing the vertices in W. Suppose that, for any $u \in A$, the hyperedges of order j containing the vertices in $U \cup \{u\}$ and not containing the vertices in W have been found by FINDHYPEREDGES. At this time, any hyperedge that has not been found is a hyperedge containing the vertices in $U \cup \{v\}$ and not containing the vertices in $W \cup A$ for some $v \notin U \cup W \cup A$. Thus, it must be found by a recursive call of FINDHYPEREDGES later.

Returning to the main procedure GRAPHFINDINGALGORITHM, for each vertex $v \in [n]$, the hyperedges of order j containing v are found by FINDHYPEREDGES in the while loop and so all the hyperedges of G_j are found by GFA. It is clear that the set of the hyperedges of order j returned by GFA is included in the set of the hyperedges of G_j. Thus, GFA finds the hyperedges of G_j correctly, given the condition that CHECKEXISTENCE and BINARYSEARCH correctly perform. Therefore, GFA correctly finds the hyperedges of G_j with probability $1 - \mathcal{O}\left(\frac{1}{n}\right)$. ■

Theorem 3 follows from Lemmas 10 and 12. Here, we mention that it is more straightforward to obtain $\mathcal{O}\left(m \log^2 n\right)$ algorithm for the hypergraph finding problem (and hence $\mathcal{O}\left(m \log^2 n\right)$ algorithm for finding the Fourier coefficients) by querying the oracle $\Theta(\log n)$ times for each cross-membership query to make the error probability $\mathcal{O}(1/\mathrm{poly}(n))$.

For the k-bounded hypergraph finding problem, it is not difficult to show that any randomized algorithm requires $\Omega(m \log n)$ cross-membership queries for constant k to make the error probability at most a given constant, provided that $m \leq n^{k-\varepsilon}$ for any constant $\varepsilon > 0$. (To obtain the lower bound, we may use Yao's minimax principle [Yao77] and the information-theoretic arguments based on the fact that, for a cross-membership query, the oracle returns one of two values.) Thus, GFA is optimal up to a constant factor, provided that $m \leq n^{k-\varepsilon}$ for any constant $\varepsilon > 0$. Note that this does not mean the optimality of the proposed algorithm for the problem of finding Fourier coefficients. While the oracle for the hypergraph finding problem gives binary values, function evaluations for the problem of finding Fourier coefficients give real values that may give more information about the Fourier coefficients.

5 Remarks on Query and Time Complexity

Suppose that we are given a k-bounded function f defined on $\{0,1\}^n$ with m non-zero Fourier coefficients. To find the Fourier coefficients of f, we first find the hyperedges of the linkage graph of f. From Theorem 6, we have the oracle with one-sided error $\delta = 1 - \frac{1}{2^{2k}}$ that gives the answer for a cross-membership query by using 2^k function evaluations. Since f has m non-zero Fourier coefficients, the linkage graph of f has at most $2^k m$ hyperedges. Given a k-bounded hypergraph with n vertices and at most $2^k m$ hyperedges, GFA uses $\mathcal{O}\left(\frac{(2e)^k k^{3.5}}{1-\delta} m \log n\right)$ cross-membership queries as shown in the proof of Lemma 10. Thus, we can find the hyperedges of the linkage graph of f (with high probability) by using $\mathcal{O}\left((16e)^k k^{3.5} m \log n\right)$ function evaluations.

Once the linkage graph of f is obtained, the Fourier coefficients can be found by using $\mathcal{O}\left(2^k m\right)$ additional function evaluations from Proposition 2. Thus, the overall query complexity of finding the Fourier coefficients of f (with high probability) is $\mathcal{O}\left((16e)^k k^{3.5} m \log n\right)$. This is $\mathcal{O}(m \log n)$ for constant k and Theorem 1 follows. Another important issue in practical applications is the time complexity of the algorithm. From the pseudocode of the proposed algorithm, we can check that the time complexity of the algorithm is

$\mathcal{O}(nm \log n)$ for constant k. (It is exponential in k.)

We should note that GFA does not assume the hierarchical property among the hyperedges. The query complexity of GFA can be improved for the restricted class of the k-bounded hypergraphs with the hierarchical property. Thus, the query complexity of finding the Fourier coefficients of a k-bounded function can be improved for general k. More concretely, to find the hyperedges of order j, we consider only the subsets of order j that contain some hyperedge of order $j-1$ that have been already found. This reduces it to $\mathcal{O}\left(\frac{j}{1-\delta} \log n\right)$ the number of iterations of the repeat loop in CHECKEXISTENCE for checking the existence of a hyperedge of order j. (It also reduces the number of CHECKEXISTENCE and BINARYSEARCH being called to $\mathcal{O}(km)$.) By this modification, the query complexity of GFA for finding a k-bounded hypergraph with n vertices and at most $2^k m$ hyperedges is reduced to $\mathcal{O}\left(\frac{2^k k^2}{1-\delta} m \log n\right)$. If we use this modified version of GFA, the query complexity of finding the Fourier coefficients is to be $\mathcal{O}\left((16)^k k^2 m \log n\right)$ for a k-bounded function defined on $\{0,1\}^n$ with m non-zero Fourier coefficients.

6 Conclusion

In this paper, we showed that the Fourier coefficients of a k-bounded function with m non-zero Fourier coefficients can be found in $\mathcal{O}(m \log n)$ function evaluations for constant k. To this end, we first showed that the problem of finding the Fourier coefficients of a k-bounded function is reduced to the problem of finding a k-bounded hypergraph with cross-membership queries under the oracle with one-sided error. Then, we gave a randomized algorithm for the hypergraph finding problem and analyzed it to obtain the desired bound.

As shown in the previous section, the query (and time) complexity of the proposed algorithm is exponential in k. Although the main concern of this paper is the case when k is constant, it would be worth trying to find an algorithm with better query (and time) complexity for general k.

References

[AA04] N. Alon and V. Asodi. Learning a hidden subgraph. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004)*, pages 110–121, 2004.

[AA05] N. Alon and V. Asodi. Learning a hidden subgraph. *SIAM Journal on Discrete Mathematics*, 18(4):697–712, 2005.

[ABK+02] N. Alon, R. Beigel, S. Kasif, S. Rudich, and B. Sudakov. Learning a hidden matching. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2002)*, pages 197–206, 2002.

[ABK+04] N. Alon, R. Beigel, S. Kasif, S. Rudich, and B. Sudakov. Learning a hidden matching. *SIAM Journal on Computing*, 33(2):487–501, 2004.

[AC04] D. Angluin and J. Chen. Learning a hidden graph using $\mathcal{O}(\log n)$ queries per edge. In *Proceedings of the 17th Annual Conference on Learning Theory (COLT 2004)*, pages 210–223, 2004.

[AC05] D. Angluin and J. Chen. Learning a hidden hypergraph. In *Proceedings of the 18th Annual Conference on Learning Theory (COLT 2005)*, pages 561–575, 2005.

[AC06] D. Angluin and J. Chen. Learning a hidden hypergraph. *Journal of Machine Learning Research*, 7:2215–2236, 2006.

[BAA+01] R. Beigel, N. Alon, M. S. Apaydin, L. Fortnow, and S. Kasif. An optimal procedure for gap closing in whole genome shotgun sequencing. In *Proceedings of the Fifth Annual International Conference on Computational Molecular Biology (RECOMB 2001)*, pages 22–30, 2001.

[BGK05] M. Bouvel, V. Grebinski, and G. Kucherov. Combinatorial search on graphs motivated by bioinformatics applications: A brief survey. In *the 31st International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2005)*, pages 16–27, 2005.

[BJT04] N. H. Bshouty, J. C. Jackson, and C. Tamon. More efficient PAC-learning of DNF with membership queries under the uniform distribution. *Journal of Computer and System Sciences*, 68(1):205–234, 2004.

[BT96] N. H. Bshouty and C. Tamon. On the Fourier spectrum of monotone functions. *Journal of the ACM*, 43(4):747–770, 1996.

[CC06] D. J. Coffin and C. D. Clack. gLINC: Identifying composability using group perturbation. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1133–1140, 2006.

[Che52] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–509, 1952.

[CJM08] S. S. Choi, K. Jung, and B. R. Moon. Lower and upper bounds for linkage discovery. *IEEE Trans. on Evolutionary Computation*, 2008. In press.

[CK08] S. S. Choi and J. H. Kim. Optimal query complexity bounds for finding graphs. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC 2008)*, 2008. To appear.

[dBIV97] J. S. de Bonet, C. L. Isbell, Jr., and P. Viola. MIMIC: Finding optima by estimating probability densities. In *Proceedings of the Advances in Neural Information Processing Systems*, volume 9, pages 424–430. The MIT Press, 1997.

[Ewe79] W. Ewens. *Mathematical Population Genetics*. Springer Verlag, 1979.

[FL70] I. Franklin and R. Lewontin. Is the gene the unit of selection ? *Genetics*, 65:707–734, 1970.

[FSBB+93] W. Fontana, P. Stadler, E. Bornberg-Bauer, T. Griesmacher, I. Hofacker, M. Tacker, P. Tarazona, E. Weinberger, and P. Schuster. RNA

folding and combinatory landscapes. *Physical Review E*, 47(3):2083–2099, 1993.

[GJS76] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[GK00] V. Grebinski and G. Kucherov. Optimal reconstruction of graphs under the additive model. *Algorithmica*, 28:104–124, 2000.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, 1989.

[HG97] G. R. Harik and D. E. Goldberg. Learning linkage. In *Foundations of Genetic Algorithms*, volume 4, pages 247–262. Morgan Kaufmann, 1997.

[HW99] R. B. Heckendorn and D. Whitley. Predicting epistasis directly from mathematical models. *Evolutionary Computation*, 7(1):69–101, 1999.

[HW03] R. B. Heckendorn and A. H. Wright. Efficient linkage discovery by limited probing. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003)*, pages 1003–1014, 2003.

[HW04] R. B. Heckendorn and A. H. Wright. Efficient linkage discovery by limited probing. *Evolutionary Computation*, 12(4):517–545, 2004.

[Jac97] J. Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. *Journal of Computer and System Sciences*, 55(3):42–65, 1997.

[Kau89] S. A. Kauffman. Adaptation on rugged fitness landscapes. In D. Stein, editor, *Lectures in the Sciences of Complexity*, pages 527–618. Addison Wesley, 1989.

[Kau93] S. A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, 1993.

[KL87] S. A. Kauffman and S. Levin. Towards a general theory of adaptive walks on rugged landscapes. *Journal of Theoretical Biology*, 128:11–45, 1987.

[KM93] E. Kushilevitz and Y. Mansour. Learning decision trees using the Fourier spectrum. *SIAM Journal on Computing*, 22(6):1331–1348, 1993.

[KP01] H. Kargupta and B. Park. Gene expression and fast construction of distributed evolutionary representation. *Evolutionary Computation*, 9(1):1–32, 2001.

[Lew74] R. Lewontin. *The Genetic Basis of Evolutionary Change*. Columbia University Press, 1974.

[Man94] Y. Mansour. Learning Boolean functions via the Fourier transform. In V. Roychowdhury, K. Y. Siu, and A. Orlitsky, editors, *Theoretical Advances in Neural Computation and Learning*, pages 391–424. Kluwer Academic, 1994.

[MG99] M. Munetomo and D. E. Goldberg. Identifying linkage groups by nonlinearity/non-monotonicity detection. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 433–440, 1999.

[MM99] H. Mühlenbein and T. Mahnig. FDA – A scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation*, 7(1):45–68, 1999.

[MP89] C. A. Macken and A. S. Perelson. Protein evolution on rugged landscapes. In *Proceedings of the National Academic of Science, USA*, volume 86, pages 6191–6195, 1989.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[PG00] M. Pelikan and D. E. Goldberg. Hierarchical problem solving by the Bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 267–274, 2000.

[PGCP00] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. Linkage problem, distribution estimation, and Bayesian networks. *Evolutionary Computation*, 8(3):311–340, 2000.

[RS07] L. Reyzin and N. Srivastava. Learning and verifying graphs using queries with a focus on edge counting. In *Proceedings of the 18th International Conference on Algorithmic Learning Theory (ALT 2007)*, pages 285–297, 2007.

[Str04] M. J. Streeter. Upper bounds on the time and space complexity of optimizing additively separable functions. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004)*, pages 186–197, 2004.

[Wal23] J. L. Walsh. A closed set of orthogonal functions. *American Journal of Mathematics*, 55:5–24, 1923.

[Yao77] A. C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science*, pages 222–227, 1977.