Identification of Partial Disjunction, Parity, and Threshold Functions*

Ryuhei Uehara, Kensei Tsuchida, and Ingo Wegener**

Addresses

Ryuhei Uehara: Faculty of Natural Sciences, Komazawa University, Komazawa 1-23-1, Setagaya-ku, Tokyo 154-8525, Japan, email: uehara@komazawa-u.ac.jp

Kensei Tsuchida: Faculty of Engineering, Toyo University, Kujirai, Kawagoe-shi, Saitama 350, Japan, email: kensei@krc.eng.toyo.ac.jp

Ingo Wegener: FB Informatik, LS II, University Dortmund, 44221 Dortmund, Germany, email: wegener@ls2.informatik.uni-dortmund.de

Abstract

Let F be a class of functions obtained by replacing some inputs of a Boolean function of a fixed type with some constants. The problem considered in this paper, which is called *attribute efficient learning*, is to identify "efficiently" a Boolean function g out of F by asking for the value of g at chosen inputs, where "efficiency" is measured in terms of the number of essential variables. We study the query complexity of attribute-efficient learning for three function classes that are respectively obtained from disjunction, parity, and threshold functions. In many cases, we obtain almost optimal upper and lower bound on the number of queries.

Key words: Complexity of Boolean functions, attribute-efficient learning, randomization.

*A preliminary version of this paper has been published as extended abstract in the Proceedings of the EuroCOLT '97, LNAI 1208, 171-184, 1997.

**Supported in part by DFG grant We 1066/8-1.

1 Introduction

Exact concept learning of Boolean concepts (or Boolean functions) was introduced by Angluin in [2]. The function classes known to be learnable are usually much smaller than the class of all Boolean functions. In particular, we will consider in this paper the problem of learning very restricted classes of functions; that is, each class F is obtained from Boolean functions of *one* type by replacing some of their input variables with some constants. (Variables that are assigned some constant are called *nonessential* whereas the others are called *essential*.) Moreover, we only consider the case where the original Boolean function is either a disjunction, a parity, or a threshold function, and nonessential variables are always replaced with constant zero. That is, F consists of functions obtained by replacing some of the variables of n-ary disjunction (resp., parity, threshold) function by zero. Hence, the identification of f in F is essentially the same as identifying the set S of essential variables of f. Our learning problem, which is called *attribute efficient learning*, is to identify a target function f "efficiently" with respect to the number of essential variables of f. Here we only allow membership queries; that is, we can ask the value of the target function f on chosen input values.

Littlestone [8] obtained several results concerning attribute-efficient learning in the mistake-bound model, among them optimal results for the class of disjunctions. Bshouty and Hellerstein [3] have also studied attribute-efficient learning in other learning models. We obtain more precise and often optimal results for disjunction, parity, and threshold functions. These functions are the most important basic gates in circuit theory and, therefore, the most fundamental functions. This fact motivates us to consider these special cases.

From the practical point of view, our results have applications on integrated circuits testing. Suppose that a chip does not work due to faulty wiring and we want to test whether each wire in the chip is connected. We would like to efficiently identify which wires are disconnected. In this case, we know the base function in the chip, and we test it by repeatedly asking the output of the function on chosen inputs. Then, how can we choose the test inputs, and how many do we need? In this case, we face the considered problem. This particular application motivates us to investigate the case of the large

number of essential variables (for example, $n - O(1)$). Interestingly, when the number of the essential variables is large, the complexity of the problem is completely different depending on the kind of the function.

A complete identification strategy can be described as a binary decision tree. Each inner node describes a query and the outgoing 0-edge (resp. 1-edge) leads to the next query if the last query has been answered by 0 (resp. 1), and each leaf corresponds to a function in F. For any target function in F, there should be one path from the root to a leaf that is consistent with the value of the function. The cost of the strategy is equal to the depth of the corresponding binary decision tree. Hence, lower bound techniques can be adopted from the theory on binary decision trees [12, 1, 15].

The simplest lower bound is the information theoretic lower bound $\lceil \log |F| \rceil$ for given class F. In fact, for many problems, we show that this lower bound is almost optimal. Sometimes we prove better lower bounds by adversary arguments. The upper bounds will be stated by showing the algorithms. All algorithms are deterministic and constructive except the following two algorithms: In the case of the class based on the parity function, where the number of essential variables is given, we will construct a randomized efficient algorithm, and also show the existence of a deterministic efficient algorithm. We only consider query complexity and not time complexity, however, all algorithms giving the upper bounds are also time efficient except the nonconstructive one.

In Section 2 we introduce some notation, state basic lower bounds, and summarize the results. In the Sections 3, 4, and 5 we consider the three different classes of functions and present often optimal results.

2 Preliminaries and basic lower bounds

A Boolean concept class F over $\{0,1\}^n$ is a set of Boolean concepts $f \subseteq \{0,1\}^n$. Since we identify f with its characteristic function $\chi_f : \{0,1\}^n \to \{0,1\}$, we treat the concept f as a Boolean function. We will try to find out which function $f \in F$ is chosen by a teacher, an adversary, or by chance. For this reason we will ask *queries* $a \in \{0,1\}^n$, say $a = (a_1, a_2, \cdots, a_n)$, and the teacher answers with the correct value $f(a)$. In this paper, we restrict ourselves to consider the Boolean concept class F that contains each function

f_S such that;

- f_S is characterized by a set $S \subseteq \{1,\ldots,n\}$ containing *essential variables*;
- the elements in $\{1,\ldots,n\} - S$ are called *nonessential variables*;
- f_S is based on one function f that is either disjunction, parity, or threshold function; and
- f_S outputs $f(s_1, s_2, \cdots, s_n)$, where $s_i = a_i$ if $i \in S$, and s_i is the constant 0 otherwise.

The queries can be chosen adaptively, i.e., the i-th query may depend on the answers to the first $i-1$ queries. After having asked $a^1,\ldots,a^i$, a concept g is *consistent* with f_S if $f_S(a^1) = g(a^1),\ldots,f_S(a^i) = g(a^i)$. We say that f_S has been *identified* if a unique concept $f_S \in F$ is consistent with the given answers.

All queries have the same cost. The cost of an identification strategy for $f_S \in F$ is the number of queries needed to identify f_S. The cost of an identification strategy for the concept class F is the maximum of the cost to identify f_S for all $f_S \in F$ with the considered identification strategy.

In the following we sometimes identify a query $a = (a_1, a_2, \cdots, a_n)$ with the query set $A = \{i \mid a_i = 1\}$. A query set leading to the answer 1 is called a *winner*. For each i with $1 \leq i \leq n$, we denote the query set $\{i\}$ of cardinality 1 by E_i.

The classes OR and $\mathsf{OR}(k)$ are based on the disjunction of n variables. We denote the disjunction function characterized by a set $S \subseteq \{1,\ldots,n\}$ by OR_S. Then, the class $\mathsf{OR}(k)$ consists of all OR_S where $S \subseteq \{1,\ldots,n\}$ and $\mid S \mid = k$, and the class OR is $\bigcup_{k=0}^{n} \mathsf{OR}(k)$. In other papers the class $\mathsf{OR}(k)$ is called 1-DNF of size k (over n variables). We use the shorter notation $\mathsf{OR}(k)$. In the same way we obtain the classes PAR and $\mathsf{PAR}(k)$ for the parity function which decides whether the number of ones in the input is odd or even. The threshold function on n variables and threshold t decides whether the input contains at least t ones. Then we consider the classes THR, $\mathsf{THR}(k)$, and $\mathsf{THR}_t(k)$, i.e., we distinguish whether we consider all $S \subseteq \{1,\ldots,n\}$ or all $S \subseteq \{1,\ldots,n\}$ of cardinality k and we distinguish whether the threshold t is unknown or known. In the theory of neural nets and also in learning theory more general threshold functions are considered.

For given weights $w_1, \ldots, w_n$ and a threshold t the output of the corresponding threshold function is 1 iff $w_1x_1 + \cdots + w_nx_n \geq t$. We restrict the base threshold function to the case $w_1 = \cdots = w_n = 1$ (and, nonessential variables will be forced to be the constant 0 on an instance).

Using the notations defined above, we summarize our results in Table 1. It is worth mentioning that we will show that the complexity of identification of $\mathsf{PAR}(k)$ is equal to the complexity of identification of $\mathsf{PAR}(n-k)$, and the complexity of identification of $\mathsf{THR}_t(k)$ is equal to the complexity of identification of $\mathsf{THR}_{k-t+1}(k)$. The results are interesting compared with the case of the class $\mathsf{OR}(k)$ with $k > \frac{n}{2}$ in the table.

In this paper, we will not deal with the classes AND and $\mathsf{AND}(k)$ defined on conjunction functions. By de Morgan's rules it easily follows that all results for disjunctions also hold for conjunctions, if the nonessential variables for conjunctions are replaced by ones.

Since the queries have two possible answers, $\lceil \log |F| \rceil$ queries are necessary for the class F. For the classes OR and PAR this leads to the lower bound n which is optimal, since these classes can be identified by asking $E_1, E_2, \ldots, E_n$. Theorem 1 contains some other information theoretic lower bounds.

Theorem 1 The following lower bounds for $\mathsf{OR}(k)$, $\mathsf{PAR}(k)$, $\mathsf{THR}_t(k)$, $\mathsf{THR}(k)$, and THR hold:

(i) $\lceil \log \binom{n}{k} \rceil$ for $\mathsf{OR}(k)$, $\mathsf{PAR}(k)$, and $\mathsf{THR}_t(k)$, if $k \geq t > 0$. This bound is larger than $k \log \frac{n}{k}$ and $(n-k) \log \left(\frac{n}{n-k}\right)$.

(ii) $\lceil \log \left(\binom{n}{k} \cdot k + 2\right) \rceil$ for $\mathsf{THR}(k)$.

(iii) $n - 1 + \lceil \log(n+1) \rceil$ for THR.

Proof. The classes $\mathsf{OR}(k)$, $\mathsf{PAR}(k)$, and $\mathsf{THR}_t(k)$ contain $\binom{n}{k}$ functions. The estimations of $\log \binom{n}{k}$ are standard calculations. For $\mathsf{THR}(k)$ we have the possible threshold values $t \in \{1, \ldots, k\}$ each leading to $\binom{n}{k}$ functions. For different t we obtain different functions. Moreover, we obtain for $t = 0$ the constant function 1 and for $t > k$ the constant function 0. Thus for the class THR the number of different functions equals

$$\sum_{1 \leq k \leq n} k \binom{n}{k} + 2 = n2^{n-1} + 2.$$

Classes	Conditions	Lower bounds	Upper bounds
OR	—	n	n
OR(k)	$k < \frac{n}{2}$	$\lceil \log \binom{n}{k} \rceil$	$k \lceil \log \frac{n}{k} \rceil + 2k - 2$
	$k \geq \frac{n}{2}$	$n-1$	$n-1$
PAR	—	n	n
PAR(k)	$k = 1$	$\lceil \log n \rceil$	$\lceil \log n \rceil$
	$k = 2$	$\lceil \log(n^2 - n) \rceil - 1$	$3 \lceil \log n \rceil - 3$
	$k = 3$	$\lceil \log \binom{n}{3} \rceil$	$4 \lceil \log n \rceil - 4$
	—	$\lceil \log \binom{n}{k} \rceil$	$k \lceil \log \frac{n}{k} \rceil + EO(k)$ (Randomized)
			$O\left(k \lceil \log \frac{n}{k} \rceil\right)$ (Deterministic, not constructive)
			$O\left(k^3 \log n\right)$ (Deterministic, constructive)
THR	—	$n - 1 + \lceil \log(n+1) \rceil$	$n - 1 + \lceil \log(n+2) \rceil$
THR(k)	—	$\lceil \log \left(\binom{n}{k} k + 2\right) \rceil$	$2(k-1) \log \frac{n-1}{k-1} + 4k - 4 + \lceil \log(n+2) \rceil$
THR$_t$(k)	—	$\lceil \log \binom{n}{k} \rceil$	$(k-1) \log \frac{n-1}{k-1} + 3k - 3 + \lceil \log(n-k+1) \rceil$
	$t \leq \frac{n}{2}$, $k = n-1$	$\max\left(\lceil \log n \rceil, \left\lceil \frac{n-1}{t} \right\rceil\right)$	$\lceil \frac{n}{t} \rceil + \lceil \log t \rceil - 1$
	$t = \frac{n}{2}$, $k \geq \frac{n}{2}$	—	$O\left((n-k) \log \frac{n}{n-k}\right)$

Remark: $\binom{n}{k} \geq \left(\frac{n}{k}\right)^k, \left(\frac{n}{n-k}\right)^{n-k}$

Table 1: Summary of results

Hence we have $\lceil \log(n2^{n-1}+2) \rceil = \left\lceil \log 2^{n-1}\left(n+2^{-(n-2)}\right)\right\rceil = n-1+\lceil \log(n+1)\rceil$. We remark that the last equality is because of ceiling. ∎

3 Disjunctions

We present two algorithms for the class $\mathsf{OR}(k)$. The second one needs less queries but the first one does not use the knowledge of k. The first one, stated in the proof of Theorem 2, will be used for the identification of $\mathsf{THR}(k)$ in Section 5.

Theorem 2 There exists an algorithm which identifies $\mathsf{OR}(k)$ with $2k\,(\log \frac{n}{k}+2)$ queries without knowing k (if $k=0$, the algorithm needs one query).

Proof. We start with the query $\{1,\ldots,n\}$. Each winner (a query with answer 1) is split into two disjoint sets of almost the same size and these sets are used as query sets. Without knowing k the algorithm does not produce more than k winners in each phase. The number of queries in the i-th phase can be bounded by $\min\{2^{i-1}, 2k\}$ and after $\lceil \log n \rceil + 1$ phases all query sets have size 1. That is, the number of queries in the i-th phase is bounded by 2^{i-1} when $1 \le i \le \lceil \log k \rceil$, and by $2k$ when $\lceil \log k \rceil + 1 \le i \le \lceil \log n \rceil + 1$. Hence, the total number of queries is bounded by

$$\begin{aligned} 2^0+\cdots+2^{\lceil \log k\rceil -1}+(\lceil \log n\rceil - \lceil \log k\rceil +1)2k &\le 2\cdot 2^{\lceil \log k\rceil -1} - 2k\lceil \log k\rceil + 2k(\log n+1) \\ &\le 2k - 2k\log k + 2k(\log n+1) \\ &= 2k\left(\log\frac{n}{k}+2\right). \end{aligned}$$

∎

Theorem 3 The class $\mathsf{OR}(k)$ can be identified with at most $\min\{n-1, k\lceil \log \frac{n}{k}\rceil + 2k - 2\}$ queries.

Proof. The upper bound $n-1$ follows from the strategy to ask $E_1, ..., E_{n-1}$. For the other upper bound we partition the set of n elements into k sets of size at most $\lceil \frac{n}{k} \rceil$. We ask these k sets as queries. We assume that a set S is a winner. Then, we can find an essential element x by binary search with at most $\lceil \log \lceil \frac{n}{k} \rceil \rceil = \lceil \log \frac{n}{k} \rceil$ queries. Then we test whether $S - \{x\}$ is still a winner and continue in this way. For each of the k essential elements

the binary search can be done with at most $\lceil \log \frac{n}{k} \rceil$ queries. Moreover, we ask k queries in the beginning. For each essential element x we have found we ask an additional query (the query $S - \{x\}$ above). We can save this additional query for the last two essential elements. After having found the last essential element we are done. If we have found the last but one essential element, we know whether there is another set which is a winner. Thus, we can determine whether the corresponding set $S - \{x\}$ is a winner without making a query. ∎

The second upper bound beats the trivial upper bound $n-1$ only if $k = n/4$ or $k \leq n/5$. If $k = o(n)$, the upper bound in Theorem 3 is $k \log \frac{n}{k} + o(k \log \frac{n}{k})$ while the lower bound is larger than $k \log \frac{n}{k}$. Hence, our algorithm is almost optimal. Moreover, the upper bound of $\mathsf{OR}(k)$ given by the algorithm in Theorem 3 is asymptotically optimal if $0 \leq k < \frac{n}{2}$. However, the lower bound given in Theorem 1 is much smaller than the upper bound for large k. We prove a better lower bound, and show that the upper bound of $\mathsf{OR}(k)$ given by the algorithm in Theorem 3 is optimal even if $\frac{n}{2} \leq k \leq n-1$.

Theorem 4 If $\frac{n}{2} \leq k \leq n-1$, $n-1$ queries are necessary to identify $\mathsf{OR}(k)$.

Proof. The claim is proved with an adversary argument. An adversary answers the queries of the identification algorithm consistently and tries to force the identification algorithm to ask many queries. Our aim is to describe an adversary which can force the identification algorithm to ask $n-1$ queries. In order to simplify the analysis the adversary gives sometimes more information to the identification algorithm. If the identification algorithm nevertheless has to ask $n-1$ queries, we are done.

First we consider the class $\mathsf{OR}(n-1)$. Query sets of size larger than 1 are useless, since the answer 1 is known in advance. Hence, only E_is are used as queries and the adversary may answer the first $n-2$ queries by 1. Then the identification algorithm still has to ask a query.

We restrict ourselves w.l.o.g. to strategies which do not include identified elements (elements which are known to be essential or nonessential) into further query sets. The adversary strategy is the following. If $k = n-1$, we follow the strategy described above. Otherwise, the adversary does the following. As long as the query sets are disjoint and at

most $n-k-1$ query sets of size 1 are asked, the adversary answers 0 for the query sets of size 1 and answers 1 for larger query sets. The answers 1 are allowed, since $k \geq \frac{n}{2}$. There are two cases to consider.

In the first case the identification algorithm has asked $n-k-1$ query sets of size 1. Then the identification algorithm has obtained the knowledge of $n-k-1$ nonessential elements by $n-k-1$ queries. The answers to these queries do not include more information. Hence, the identification algorithm is faced with the problem $\mathsf{OR}(k)$ on $k+1$ variables. As we have shown, the identification algorithm needs k additional queries for this purpose and we are done.

In the second case the identification algorithm asks a query A which is not disjoint from a previous query B. By our assumptions $|B| \geq 2$. The adversary answers the query by 1. Furthermore, the adversary gives the following information for free. He chooses an element $i \in A \cap B$ and an element $j \in B - \{i\}$. Then he tells the identification algorithm that i is essential and j is nonessential. This is consistent with the previous answers. Now the answers to the queries A and B do not contain further information. The identification algorithm has obtained the knowledge of one essential element and one nonessential element by 2 queries. Hence, the identification algorithm is faced with the problem $\mathsf{OR}(k-1)$ on $n-2$ variables. For this problem we prove with the same strategy that $n-3$ queries are necessary. ∎

4 Parity functions

For small k a small number of queries is sufficient to identify $\mathsf{OR}(k)$ while the maximum number of $n-1$ queries is necessary to identify $\mathsf{OR}(n-k)$. For parity functions we prove that the complexity of the identification of $\mathsf{PAR}(k)$ is equal to the complexity of the identification of $\mathsf{PAR}(n-k)$. What is the reason for this difference?

Parity (or sum mod 2 denoted by $\oplus$) is, like disjunction, a binary, associative, and commutative operation with 0 as neutral element. But $(\{0,1\}, \oplus)$ is an additive group while the equation $1 \vee x = 0$ has no solution.

Proposition 5 For given n, the complexity of the identification of $\mathsf{PAR}(k)$ is equal to the

complexity of the identification of $\mathsf{PAR}(n-k)$.

Proof. Let g_1 be the parity function on k of the n variables and g_2 be the parity function on the $n-k$ other variables. This defines a one-to-one mapping between the classes $\mathsf{PAR}(k)$ and $\mathsf{PAR}(n-k)$. For a query set S let $a_1(S)$ and $a_2(S)$ be the answers given for g_1 and g_2 resp. Then $a_1(S) \oplus a_2(S) \equiv |S| \mod 2$. Hence, we can compute $a_1(S)$ from $a_2(S)$ and vice versa. A strategy for $\mathsf{PAR}(k)$ which identifies g_1 identifies with the corresponding answers g_2, if it is used for $\mathsf{PAR}(n-k)$. ∎

Because of Proposition 5 we assume w.l.o.g. that $k \leq \frac{n}{2}$. For the OR-classes, one essential variable can be identified with binary search using $\lceil \log n \rceil$ queries. For the PAR-classes the same holds, if the number of essential elements in a given set is odd. If we partition the set of elements into two parts, one has an odd number of essential elements and the other one has an even number. With one query we can find out which is the "odd part" and can continue with this set. On the other hand, when the number of the essential elements in a given set is even, the problem turns to be more difficult. In this case, we can not easily determine even whether the set contains essential elements or not.

We get good upper bounds (see Theorem 1) for $k \in \{1,2,3\}$.

Proposition 6 (i) $\mathsf{PAR}(1)$ can be identified with $\lceil \log n \rceil$ queries.

(ii) $\mathsf{PAR}(2)$ can be identified with $3\lceil \log n \rceil - 3$ queries.

(iii) $\mathsf{PAR}(3)$ can be identified with $4\lceil \log n \rceil - 4$ queries.

Proof. The first result follows from the binary search technique sketched above. For the second claim we use a so-called *separating system* that consists of the following sets S_m (see e.g. [15] for detail): For each m with $0 \leq m \leq \lceil \log n \rceil - 1$, S_m consists of the numbers i whose m-th bit in the binary representation equals 1. Let i and j be integers with $0 \leq i < j \leq n$. Then, since $i \neq j$, their binary representations are different from each other, and there exists a set S_m where $i \in S_m$ and $j \notin S_m$ or $i \notin S_m$ and $j \in S_m$. First we query the sets of the separating system. For the essential elements i and j, the corresponding set S_m should lead to the answer 1. Since we can save the last query for the separating system, we make at most $\lceil \log n \rceil - 1$ queries in this step. Then we may start

binary searches on S_m and its complement, sets of at most $\lceil \frac{n}{2} \rceil$ elements. Each binary searches requires at most $\log \lceil \frac{n}{2} \rceil = \lceil \log n \rceil - 1$ queries. The third result is obvious; we find the first essential element with binary search, and apply the second result (even for a set of $n-1$ elements). We save one query, since the first query can be chosen as a set belonging to the separating system in the second round. ∎

How can we find for larger k a set with an odd number of essential objects? First we consider randomized efficient identification algorithms. Next, we show the existence of the deterministic efficient identification algorithms (the proof is not constructive). Finally, we remark that we can construct the deterministic identification algorithms using the notion of "splitters" to be defined later.

First we consider randomized identification algorithms with zero-error. Either we can guarantee that the algorithm always gives the right answer and the expected number of queries is small or we can guarantee an upper bound on the number of queries and a small upper bound on the probability that the algorithm answers "don't know". The reader not familiar with randomized algorithms may look at [10] for an introduction. Randomized algorithms in the context of query complexity have been used in [14] and [6] for randomized decision trees.

Obviously, one half of the queries leads to the answer 1. Asking randomized queries we get in expected time 2 a query set with answer 1. Then, we find two essential elements with binary search. This leads to an error-free randomized algorithm with an expected number of $k\,(\lceil \log n \rceil + 1)$ queries. In order to meet the lower bound we should decrease the size of the query sets.

Theorem 7 For $\mathsf{PAR}(k)$ there is a randomized error-free identification algorithm with an expected number of $k \log \frac{n}{k} + EO(k)$ queries. If the number of queries is bounded by $O\left(k \log \frac{n}{k}\right)$, the probability of not having identified the unknown function can be bounded by a function tending with respect to $k \log \frac{n}{k}$ exponentially fast to 0.

Proof. The randomized algorithm repeats the following steps until all essential elements have been found: Without loss of generality, we assume that the algorithm has k' unidentified essential elements in a set of size n' with $0 < k' < n'$. First, the algorithm chooses

at random and independently r elements, where $r = \left\lceil \frac{n'}{k'} \right\rceil$. The choices are made with replacement, i.e., the algorithm may choose an element more than once. Next the algorithm collects the chosen elements into a query set (of size at most r), and makes a query. Then, the query set contains exactly one essential element with probability at least

$$\binom{r}{1} \frac{k'}{n'} \left(1 - \frac{k'}{n'}\right)^{r-1} = \left\lceil \frac{n'}{k'} \right\rceil \frac{k'}{n'} \left(1 - \frac{k'}{n'}\right)^{\left\lceil \frac{n'}{k'} \right\rceil - 1} \geq \left(1 - \frac{k'}{n'}\right)^{\frac{n'}{k'}} > \frac{1}{e},$$

where e is the Eulerian constant $2.718\cdots$. Hence, the probability of success to find a set containing an odd number of essential elements is bounded below by $\frac{1}{3}$. For a successful query we perform the binary search only on the small query set and not on the large complement. For the binary searches we are done in any case with at most

$$\begin{aligned} \sum_{1 \leq k' \leq k} \left\lceil \log \frac{n'}{k'} \right\rceil &= \sum_{1 \leq k' \leq k} \left\lceil \log \frac{n - k + k'}{k'} \right\rceil \leq \sum_{1 \leq k' \leq k} \log \frac{n}{k'} + O(k) \\ &= k \log n - \log(k!) + O(k) = k \log n - (k \log k - O(k)) + O(k) \\ &= k \log \frac{n}{k} + O(k) \end{aligned}$$

queries. The expected number of queries to find the next set with an odd number of essential elements is bounded by 3 leading to an additional expected number of $3k$ queries.

For the second claim we get smaller upper bounds than for general error-free randomized algorithms. We have independent trials each with a probability of success at least $\frac{1}{3}$. We need k successes and may perform $O\left(k \log \frac{n}{k}\right)$ trials. Now the probability of less than k successes can be estimated using the Chernoff bound. ∎

Now we look for deterministic identification algorithms. The main idea is to "recycle" query sets, i. e. to try to use them more than once. A random query set S contains approximately half of the essential elements. The set is good if it contains an odd number of essential elements. If the number of essential elements in S is even, we try other sets. If we find the first essential element i contained in S, we know without any further query that $S - \{i\}$ contains an odd number of essential elements. These ideas are made precise in the next proposition and lead to the proof of the existence of the deterministic algorithm.

Proposition 8 There exists a deterministic algorithm that identifies the class $\mathsf{PAR}(k)$ with at most $\lceil \log \binom{n}{k} \rceil + k \lceil \log n \rceil + 2$ queries.

Proof. If $k = \Omega(n)$, the result is obvious, since $n-1$ queries of E_is suffice. We assume that $k \leq \frac{n}{3}$. Then,

$$\binom{n}{k-1} \leq \frac{1}{2}\binom{n}{k}$$

and

$$\binom{n}{k} + \binom{n}{k-1} + \binom{n}{k-2} + \cdots + \binom{n}{1} \leq 2\binom{n}{k}.$$

We choose $m = \lceil \log \binom{n}{k} \rceil + 2$ random sets $R_1, \ldots, R_m$. Each random set is constructed as follows; for each i and j with $1 \leq i \leq n$ and $1 \leq j \leq m$, i is in R_j with probability $\frac{1}{2}$. Let $S \subseteq \{1, \ldots, n\}$ be one set where $1 \leq \mid S \mid \leq k$. The probability that $S \cap R_j$ has odd cardinality equals $\frac{1}{2}$.

The probability that all random sets are bad for S, i.e. $S \cap R_1$, ..., $S \cap R_m$ all have even cardinality, equals $\left(\frac{1}{2}\right)^m \leq \frac{1}{4}\binom{n}{k}^{-1}$. Therefore, the probability that for at least one of the less than $2\binom{n}{k}$ sets S all random sets are bad is bounded above by $\frac{1}{2}$. Hence, there exist m query sets $T_1, \ldots, T_m$ such that for each $S \subseteq \{1, \ldots, n\}$ with $1 \leq \mid S \mid \leq k$ there exists a set T_j such that $\mid S \cap T_j \mid$ is odd.

The sets $T_1, \ldots, T_m$ are used as query sets. Let S be the unknown set of essential elements. Then $\mid S \cap T_{i(1)} \mid$ is odd for some $i(1)$ and we find an essential element $j(1)$ with binary search on $T_{i(1)}$. Let $S_1 := S - \{j(1)\}$. Then $\mid S_1 \cap T_{i(2)} \mid$ is odd for some $i(2)$ and we find an essential element $j(2)$ with binary search on $T_{i(2)} - \{j(1)\}$ and so on. Here we have used the fact that we have implicit knowledge about the query set $T_i - \{j(1)\}$ if we have asked the query T_i and know that $j(1)$ is essential.

Altogether we have asked m queries $T_1, \ldots, T_m$ and need $k \lceil \log n \rceil$ further queries for the binary searches. ∎

The term $\lceil \log \binom{n}{k} \rceil$ is equal to the information theoretic lower bound. In order to obtain an asymptotically optimal algorithm the term $k \log n$ should be replaced by $O\left(k \log \frac{n}{k}\right)$. If $k = O\left(n^{1-\varepsilon}\right)$, nothing has to be done. Hence, we have to consider functions like $k = \frac{n}{\log^2 n}$ or $k = \frac{n}{\log \log n}$.

Theorem 9 There exists a deterministic algorithm that identifies the class PAR(k) with at most $O\left(k \log \frac{n}{k}\right)$ queries.

Proof. If $k \leq n/2$ and $k = \Omega(n)$, the proposed upper bound is $O(n)$ and we are done by

using the trivial strategy asking $E_1, \ldots, E_{n-1}$. Hence, we can assume that $k = \frac{n}{h(n)}$ for some function $h(n)$ tending to ∞. The first idea is to replace the random sets in the proof of Proposition 8 by random sets of size $\frac{n}{k} = h(n)$. For sets S with $k' = o(k)$ elements the intersection of S and such a random set is empty with large probability. Therefore, we work with random sets of size $\left(\frac{n}{k}\right)^2 = h(n)^2$. The cost for each binary search is increased by a factor 2 only. For sets S where $|S| \geq \frac{n}{h(n)^2}$ the probability of success for a random set is at least $\frac{1}{3}$. Hence, with $O\left(\log \binom{n}{k}\right)$ sets we can identify all but $\frac{n}{h(n)^2}$ essential elements.

In the second phase we work with sets of size $h(n)^4$ and can identify all but $\frac{n}{h(n)^4}$ essential elements. This process is continued until all essential elements can be identified.

The number of query sets besides the queries in the binary searches is bounded by

$$\begin{aligned} & O\left(\log \binom{n}{n/h(n)} + \log \binom{n}{n/h(n)^2} + \log \binom{n}{n/h(n)^4} + \cdots\right) \\ = \; & O\left(\frac{n}{h(n)} \log h(n) + \frac{n}{h(n)^2} 2 \log h(n) + \frac{n}{h(n)^4} 4 \log h(n) + \cdots\right) \\ = \; & O\left(k \log \frac{n}{k}\right). \end{aligned}$$

We still have to estimate the number of queries during the binary searches. In the first phase we find not more than $\frac{n}{h(n)}$ essential elements each with a cost of $2 \log h(n)$. In the second phase we find not more than $\frac{n}{h(n)^2}$ essential elements each with a cost of $4 \log h(n)$ and so on. Hence, the same estimation as before shows that $O\left(k \log \frac{n}{k}\right)$ queries suffice. ∎

The upper bound of Theorem 9 is asymptotically optimal for $k \leq n/2$, since it meets the lower bound of Theorem 1. Applying Proposition 5 we have proved that the complexity of identification of the concept classes $\mathsf{PAR}(k)$ is only by a constant factor larger than the simple information theoretic lower bound.

Remark. If one is interested in *applying* the identification algorithms, one has to notice that in Proposition 8 and Theorem 9 only the existence of identification algorithms with the proposed bounds has been proved. For most applications the simple randomized error-free identification algorithm described in the proof of Theorem 7 is good enough. The algorithms proving all the other bounds in this paper are explicitly described and can be applied without further considerations.

We also remark that to identify the class $\mathsf{PAR}(k)$, we can use (n, k, r, α)-splitters defined as follows. (The parameter α is not used in our application.)

Definition 10 [11] An (n,k,r,α)-*splitter* of size s is a sequence $P_1,\ldots,P_s$ of partitions of $\{1,\ldots,n\}$ into r sets such that for each set S of cardinality k at least a fraction of $\alpha > 0$ partitions put the elements of S into r different sets.

Lemma 11 An (n,k,r,α)-splitter of size s implies that $\mathsf{PAR}(k)$ can be identified with $rs + k\lceil\log n\rceil$ queries.

Proof. With rs queries of the sets in $P_1,\ldots,P_s$ we find one partition where k subsets give the answer 1. With binary search on these sets we find the k essential elements. ∎

Although the notion of splitters is quite new, their existence, efficient constructions, and applications to learning theory have been investigated [4, 13, 9, 5, 3]. These splitters lead by Lemma 11 to the *existence* of the identification algorithms for $\mathsf{PAR}(k)$ with $O(k^3\log n)$ queries and the efficient *construction* of the identification algorithms with $O(k^4\log n)$ queries [7].

5 Threshold functions

Theorem 12 The class THR can be identified with $n-1+\lceil\log(n+2)\rceil$ queries.

Proof. In the first phase we only use query sets of type $\{1,\ldots,m\}$ for some $m \leq n$. Since threshold functions are monotone, we can find with binary search the minimal $s \in \{1,\ldots,n\}$ such that the answer to the query $\{1,\ldots,s\}$ is 1. We remark that there are two special cases that such an s does not exist; the answer to the empty query is 1, and the answer to the query $\{1,2,\ldots n\}$ is 0. Hence, the binary search above takes $\lceil\log(n+2)\rceil$ queries. In the special cases we are done, since the function is the constant 1 and 0, respectively. If $s \in \{1,\ldots,n\}$, we know that s is an essential element, and the number of the essential elements in $\{1,\ldots,s\}$ equals the threshold value. The query $\{1,\ldots,s\}-\{j\}$ decides for $j \in \{1,\ldots,s-1\}$ whether j is essential and the query $\{1,\ldots,s-1,j\}$ does the same for $j \in \{s+1,\ldots,n\}$. These are $n-1$ additional queries. ∎

Theorem 13 Let $0 \leq t \leq k \leq n$. The class $\mathsf{THR}_t(k)$ can be identified with at most $(k-1)\log\frac{n-1}{k-1} + 3k - 3 + \lceil\log(n-k+1)\rceil$ queries.

Proof. The first phase is the same as in the proof of Theorem 12. Remind that we know the value of the parameters n, k, and t for this class. Thus, we can determine whether we are in the special two cases in the proof of Theorem 12 with no queries. Moreover, we know that $t \leq s \leq n-k+t$ from the knowledge of the parameters, where s is the minimal value such that the answer to the query $\{1,\ldots,s\}$ is 1. Thus, the step for finding s requires at most $\lceil \log(n-k+1) \rceil$ queries. Here we know an essential element s such that the query $\{1,\ldots,s\}$ leads to the answer 1 and the query $\{1,\ldots,s-1\}$ leads to the answer 0. Hence, $\{1,\ldots,s-1\}$ contains exactly $t-1$ essential elements and $\{s+1,\ldots,n\}$ contains the remaining $k-t$ essential elements.

We next identify $k-t$ essential elements in $\{s+1,\ldots,n\}$. To do this, we use an algorithm for $\mathsf{OR}(k-t)$. The algorithm for $\mathsf{OR}(k-t)$ works on the set $\{s+1,\ldots,n\}$. When the algorithm makes a query set in $\{s+1,\ldots,n\}$, we add the set $\{1,\ldots,s-1\}$ to the query set. Since the set $\{1,\ldots,s-1\}$ contains $t-1$ essential elements, the algorithm for $\mathsf{OR}(k-t)$ on $\{s+1,\ldots,n\}$ identifies the $k-t$ essential elements in $\{s+1,\ldots,n\}$. Finally we identify $t-1$ essential elements in $\{1,\ldots,s-1\}$. Then we use an algorithm for $\mathsf{AND}(t-1)$ on the set $\{1,\ldots,s-1\}$. When the algorithm makes a query set in $\{1,\ldots,s-1\}$, we add $\{s\}$ to the query set. Since s is an essential element, this algorithm identifies the $t-1$ essential elements in $\{1,\ldots,s-1\}$. In Section 2 we have seen that the complexity of the identification of the AND-classes is the same as for the corresponding OR-classes. By Theorem 3 the number of queries of the second and third phase is bounded by

$$\begin{aligned}
&(k-t)\left\lceil \log \frac{n-s}{k-t} \right\rceil + 2(k-t) + (t-1)\left\lceil \log \frac{s-1}{t-1} \right\rceil + 2(t-1) \\
&\leq \quad (k-t)\log \frac{n-s}{k-t} + (k-t) + (t-1)\log \frac{s-1}{t-1} + (t-1) + 2k - 2 \\
&\leq \quad (k-1)\log \frac{n-1}{k-1} + 3k - 3.
\end{aligned}$$

The last inequality is for $a = \frac{s-1}{n-1}$ and $b = \frac{t-1}{k-1}$ equivalent to

$$-b\log b - (1-b)\log(1-b) \leq -b\log a - (1-b)\log(1-a).$$

This inequality holds, since $0 \leq a, b \leq 1$, see, e.g., [1, Lemma 5.3, p.20]. ∎

Theorem 14 The class $\mathsf{THR}(k)$ can be identified with at most $2(k-1)\log \frac{n-1}{k-1} + 4k - 4 + \lceil \log(n+2) \rceil$ queries.

Proof. The first phase of the algorithm in the proof of Theorem 12 does not need the knowledge of t. Hence, we can use this phase also here.

Afterwards we can solve the problem by the application of algorithms for OR (and equivalently AND). But now the number of essential elements is not known, since it depends on the unknown parameter t. Hence, we use the algorithm of Theorem 2. Then, the number of queries for $\mathsf{THR}(k)$ is bounded by

$$\lceil \log(n+2) \rceil + 2(k-t)\log\frac{n-s}{k-t} + 4(k-t) + 2(t-1)\log\frac{s-1}{t-1} + 4(t-1).$$

Now we can use the same estimations as in the proof of Theorem 13. ∎

By simple calculations together with the lower bounds showed in Section 2, we have the following corollary.

Corollary 15 The upper bound of THR given in the algorithm in Theorem 12 is optimal if $n+1$ is not a power of 2. Otherwise, we miss the lower bound by 1. The upper bound of $\mathsf{THR}_t(k)$ given in the algorithm in Theorem 13 is asymptotically optimal within a constant factor if $k \leq \frac{n}{2}$. The upper bound of $\mathsf{THR}(k)$ given in the algorithm in Theorem 14 is asymptotically optimal within a constant factor if $k \leq \frac{n}{2}$.

For the classes $\mathsf{THR}_t(k)$ and $\mathsf{THR}(k)$, their upper bounds are asymptotically optimal for $k \leq \frac{n}{2}$. The optimal number of queries does not rely essentially on the threshold value t or even on the knowledge of t. Does this hold also for $k > \frac{n}{2}$? We only answer this question in special cases. We first observe that we get $\mathsf{OR}(k)$ from $\mathsf{THR}_t(k)$ if $t = 1$. We know that for this case we need $n-1$ queries, if $k \geq \frac{n}{2}$. Moreover, the cases $t = 1$ and $t = k$ are equivalent. This observation can be generalized.

Proposition 16 For given n, the complexity of the identification of $\mathsf{THR}_t(k)$ is equal to the complexity of the identification of $\mathsf{THR}_{k-t+1}(k)$.

Proof. Let S be the set of essential elements for the problem $\mathsf{THR}_t(k)$. A query Q leads to the answer 1 if and only if $|Q \cap S| \geq t$. If S is the set of essential elements for the problem $\mathsf{THR}_{k-t+1}(k)$, the query $Q' := \{1,\ldots,n\} - Q$ leads to the answer 1 if and only if $|Q' \cap S| \geq k-t+1$ which is equivalent to $|Q \cap S| \leq t-1$. Hence, we can simulate the query Q for $\mathsf{THR}_t(k)$ by the query Q' for $\mathsf{THR}_{k-t+1}(k)$ and vice versa. ∎

Because of Proposition 16 we may restrict ourselves to the case $t \leq n/2$. We are still left with a variety of parameter choices. If one uses threshold gates in Boolean circuits, the gates become more powerful if t is close to $n/2$. In particular, for $t = 1$ we get the "simple" OR-function and for $t = n/2$ the "complex" MAJORITY-function. Hence, one might believe that threshold functions with small t are easier to identify than threshold functions with t close to $n/2$. But the following two theorems prove that this is not true.

We first consider the case $k = n - 1$. As stated below, if $t \leq n/2$, the query complexity decreases with t. Interestingly, while $n-1$ queries are necessary for the OR-function (t=1), only $\lceil \log n \rceil$ queries are sufficient to identify the MAJORITY-function, namely the case $t = \frac{n}{2}$.

Theorem 17 Let $t \leq \frac{n}{2}$. To identify the class $\mathsf{THR}_t(n-1)$

- $\max\left(\lceil \log n \rceil, \left\lceil \frac{n-1}{t} \right\rceil\right)$ queries are necessary and
- $\lceil \frac{n}{t} \rceil + \lceil \log t \rceil - 1$ queries are sufficient.

By Proposition 16 we obtain similar results for the case $t > \frac{n}{2}$.

Proof. The lower bound $\lceil \log n \rceil$ is the simple information theoretical lower bound. For the lower bound $\left\lceil \frac{n-1}{t} \right\rceil$ a simple adversary argument works. A query set of size $t+1$ or larger is useless, since we know in advance that the answer is 1. A query set of size $t-1$ or smaller is useless, since we know in advance that the answer is 0. Hence, all query sets have size t. An adversary may answer 1 for the first $\left\lceil \frac{n-1}{t} \right\rceil - 1$ queries. Then there are still at least 2 candidates for the nonessential element and another query is necessary.

For the upper bound we construct $\lceil \frac{n}{t} \rceil$ query sets of size t such that each element is contained in at least one set. We look for a query set leading to the answer 0, since this set contains the only nonessential element. At least one of the queries is good for our purposes. Hence, it is sufficient to ask $\lceil \frac{n}{t} \rceil - 1$ queries. Since $t \leq \frac{n}{2}$, we now can use binary search on this set, if we add always as many already identified elements such that the query size becomes t. ∎

Next we consider the MAJORITY-function ($t = \frac{n}{2}$) for any k. We present an asymptotically optimal identification algorithm (for the lower bound see Theorem 1(i)). We leave it to the reader to generalize this result to other threshold values.

Theorem 18 The class $\mathsf{THR}_{\frac{n}{2}}(n-k)$ can be learned with $O\left(k \log \frac{n}{k}\right)$ queries.

Proof. If $k > \frac{n}{2}$, nothing has to be done and 0 queries are sufficient. Thus, hereafter, we assume that $k \leq \frac{n}{2}$ and the number of nonessential elements is smaller than the number of essential elements. If $k = \Theta(n)$, the upper bound follows from Theorem 12. Hence, w.l.o.g. we can assume that $k \leq \frac{n}{8}$. Moreover, we assume that k divides n. Otherwise, we may add less than k elements known to be nonessential. Additional nonessential elements have no influence on the answer to queries and can therefore be added. This does not hold for additional essential elements.

We present two algorithms. The first one meets the upper bound $O\left(k \log \frac{n}{k}\right)$ if $k = \Omega\left(\sqrt{\frac{n}{\log n}}\right)$, and so does the second one if $k = O(n^{1-\varepsilon})$ for some $\varepsilon > 0$. Since each k fulfills at least one of the conditions, the two algorithms are sufficient to prove the theorem.

The main idea is to search for the k nonessential elements. The set $\{1, \ldots, n\}$ of elements is partitioned into $B_1 = \{1, \ldots, \lceil \frac{n}{3} \rceil\}$, $B_2 = \left\{\lceil \frac{n}{3} \rceil + 1, \ldots, \left\lceil \frac{2n}{3} \right\rceil\right\}$, and $B_3 = \left\{\left\lceil \frac{2n}{3} \right\rceil + 1, \ldots, n\right\}$. We only describe how to find the at most k nonessential elements in B_3. The same approach can be used for B_1 and B_2.

Both algorithms determine (as in the proof of Theorem 12) the minimal s such that $S^* = \{1, \ldots, s\}$ contains $\frac{n}{2}$ essential elements. Since $\frac{n}{2} \leq s \leq \frac{n}{2} + k$, $\lceil \log(k+1) \rceil$ queries are sufficient for this purpose. Remark that $S^* \cap B_3 = \emptyset$ since $k \leq \frac{n}{8}$.

We will use the OR-strategy in Section 3 to search for nonessential elements. In order to apply OR-strategies we construct so-called *dummy* sets D_j with $1 \leq j \leq m$ for some m where D_j contains exactly $\frac{n}{2} - j$ essential elements. Let B be a set of size $j \leq m$ and that is disjoint from all dummy sets. Then the query $B \cup D_j$ is answered 0 iff B contains at least one nonessential element. This is equivalent to an OR-query B with respect to nonessential elements. Thus we can use the OR-strategy in Section 3 by making a query $B \cup D_{|B|}$ when we want to know whether B contains at least one nonessential element. We have to be careful. Since we only have the dummy sets $D_1, \cdots, D_m$, the OR-strategy only can ask queries B where $|B| \leq m$.

The first algorithm then asks the queries $S^* - \{i\}$, $1 \leq i \leq k + \frac{n}{k}$. The query $S^* - \{i\}$ is answered by 0 iff the element i is essential. Since the number of nonessential elements equals k, we determine at least $\frac{n}{k}$ essential elements $e_1, \ldots, e_{\frac{n}{k}} \in \{1, \ldots, k + \frac{n}{k}\}$. The

dummy set D_j is defined as $S^* - \{e_1, \ldots, e_j\}$ containing $\frac{n}{2} - j$ essential elements. Then, since $k \leq \frac{n}{8}$, $D_j \subseteq S^* \subseteq B_1 \cup B_2$, and the dummy sets are disjoint from B_3. Now we determine the nonessential elements in B_3 with an OR-strategy starting with a partition of B_3 into $\lceil \frac{k}{3} \rceil$ sets of size bounded by $\frac{n}{k}$. Each query which is a winner, is partitioned into two queries of almost the same size. The number of queries to determine the nonessential elements in B_3 is bounded by $\lceil \log(k+1) \rceil + k + \frac{n}{k}$ for the construction of the dummy sets and $O\left(k \log \frac{n}{k}\right)$ for the OR-strategy. Since $k = \Omega\left(\sqrt{\frac{n}{\log n}}\right)$, $\frac{n}{k} = O(k \log \frac{n}{k})$ and we are done in this case.

The second algorithm chooses some l where $s < l \leq \lceil \frac{2n}{3} \rceil + 1$ such that the elements $l, \ldots, n$ can be partitioned into $k+1$ sets $R_1, \ldots, R_{k+1}$ of the same size $r \leq \frac{n}{k}$. Since the number of nonessential elements is k, there exists one set R_i consisting of r essential elements. Later the algorithm will determine such an i. We try to find a set $D^* \subseteq S^*$ with exactly $\frac{n}{2} - r$ essential elements. With D^* and R_i it is easy to fulfill our task. Let $R_i = \{e_1, \ldots, e_r\}$, then $D_j := D^* \cup \{e_1, \ldots e_{r-j}\}$ for each j with $1 \leq j < r$ and $D_r := D^*$ can be used as dummy sets containing exactly $\frac{n}{2} - j$ essential elements.

The elements of R_i all are essential and we can determine the nonessential elements in the union of all R_h, $h \neq i$, with these dummy sets and an OR-strategy asking queries whose size is bounded by r. This again can be done with $O\left(k \log \frac{n}{k}\right)$ queries using the same argument in the proof of Theorem 2. We still have to describe how we can find the dummy sets or R_i and D^*.

We consider $C_0 = S^* - \{1, \ldots, r\}, \ldots, C_k = S^* - \{1, \ldots, r+k\}$ as candidates for D^*. We know that C_0 contains at least $\frac{n}{2} - r$ essential elements and that C_k contains at most $\frac{n}{2} - r$ essential elements. Hence, one of the candidates can serve as D^*. Let us assume that we ask the queries $R_1 \cup C_j, \ldots, R_{k+1} \cup C_j$. If all queries are answered by 0, even the union of R_i (with r essential elements) and C_j does not contain $\frac{n}{2}$ essential elements. Hence, C_j contains less than $\frac{n}{2} - r$ essential elements. If at least one query is answered by 1, then C_j contains at least $\frac{n}{2} - r$ essential elements. Hence, with binary search on $j \in \{0, \ldots, k\}$ and altogether $O(k \log k)$ queries we find the largest j such that one of the queries $R_1 \cup C_j, \ldots R_{k+1} \cup C_j$ is answered by 1. This implies that C_j contains at least $\frac{n}{2} - r$ essential elements while C_{j+1} contains less than $\frac{n}{2} - r$ essential elements. Since $C_j \supseteq C_{j+1}$

and $|C_j - C_{j+1}| = 1$, C_j contains exactly $\frac{n}{2} - r$ essential elements and we choose $D^* = C_j$. Moreover, let R_i be one of the sets such that the query $R_i \cup D^*$ has been answered by 1, then all r elements of R_i are essential. Hence, we have found D^* and R_i. The number of additional queries is $O(k \log k) = O\left(k \log \frac{n}{k}\right)$, since $k = O\left(n^{1-\varepsilon}\right)$. ∎

Acknowledgment

The first and second authors would like to thank Professor Etsuro Moriya for his comments on this problem, and Professor Masao Hara and Professor Sei'ich Tani for their helpful suggestions. The first author wishes to thank Professor Takumi Kasai, Professor Seinosuke Toda, and Professor Zhi-Zhong Chen for their helpful advice during the course of this work. The first author also thanks Professor Misao Nagayama for her comments on earlier drafts of this paper.

References

[1] R. Ahlswede and I. Wegener. *Search Problems*. Wiley, 1987.

[2] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

[3] N.H. Bshouty and L. Hellerstein. Attribute-efficient learning in query and mistake-bound models. In *Proc. 9th Ann. Conf. on Computational Learning Theory*, pages 235–243. ACM, 1996.

[4] J. Friedman. Constructing $O(n \log n)$ size monotone formulae for the k-th elementary symmetric polynomial of n Boolean variables. In *Proc. 25th Symp. on Foundations of Computer Science*, pages 506–515. IEEE, 1984.

[5] J. Håstad, I. Wegener, N. Wurm, and S.-Z. Yi. Optimal depth, very small size circuits for symmetric functions in AC^0. *Information and Computation*, 108:200–211, 1994.

[6] R. Heiman and A. Wigderson. Randomized vs. deterministic decision tree complexity for read-once Boolean functions. *Computational Complexity*, 1:311–329, 1991.

[7] T. Hofmeister. Personal communication. 1996.

[8] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[9] K. Mehlhorn. On the program size of perfect and universal hash functions. In *Proc. 23rd Symp. on Foundations of Computer Science*, pages 170–175. IEEE, 1982.

[10] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge, 1995.

[11] M. Naor, L.J. Schulman, and A. Srinivasan. Splitters and near-optimal derandomization. In *Proc. 36th Symp. on Foundations of Computer Science*, pages 182–191. IEEE, 1995.

[12] C. Picard. *Théorie des Questionnaires*. Gauthier-Villars, 1965.

[13] P. Ragde and A. Wigderson. Linear-size constant-depth polylog-threshold circuits. *Information Processing Letters*, 39:143–146, 1991.

[14] M. Saks and A. Wigderson. Probabilistic Boolean decision trees and the complexity of evaluating game trees. In *Proc. 27th Symp. on Foundations of Computer Science*, pages 29–38. IEEE, 1986.

[15] I. Wegener. *The Complexity of Boolean Functions*. Wiley-Teubner, 1987.