Learning Unrestricted Read-Once Formulas in Polynomial Time (Extended Abstract) [*]

Dana Angluin [†] Lisa Hellerstein [‡] Marek Karpinski [§]

1 Introduction

We present the first polynomial time algorithm that exactly learns the whole class of boolean read-once formulas using membership and equivalence queries. Previous work has shown that no polynomial time algorithm can exactly learn all read-once formulas using only membership or only equivalence queries, so this is a minimal set of types of queries for this problem. Our algorithm implies the existence of an efficient learning algorithm for read-once formulas in the distribution-free definition of Valiant, provided membership queries are available. Recent work of Kearns and Valiant implies that even predicting read-once formulas in the distribution-free setting *without* membership queries is as hard as deciding certain cryptographic predicates [11]. Thus, as in the case of dfas, membership queries appear to be crucial to the learnability of read-once formulas. Our work contributes new insight to a line of research into the power of queries in computational learning problems.

The algorithm that we present is a simple and elegant transformation of any polynomial time algorithm to learn monotone read-once formulas using membership queries into a polynomial time algorithm to learn (not necessarily monotone) read-once formulas using membership and equivalence queries. The best previous algorithms for learning monotone read-once formulas using membership queries required time $O(n^3)$ and $O(n^3)$ membership queries [4,8]. We give a new simpler algorithm for this problem that runs in the same time bound and uses only $O(n^2)$ membership queries. Using this new algorithm as a subroutine, the transformation learns the whole class of read-once formulas in time $O(n^4)$ using $O(n^3)$ membership queries and $O(n)$ equivalence queries.

[*]Most of the results reported here are contained in the technical report [5].

[†]Computer Science Dept., Yale University, P. O. Box 2158, New Haven, CT 06520. Research supported by NSF grant IRI-8718975.

[‡]Computer Science Division University of California Berkeley, CA 94720. Research supported by NSF Grant CCR-8411954, and an AT&T GRPW grant.

[§]International Computer Science Institute Berkeley, CA 94704. On leave from the University of Bonn. Research partially supported by DFG Grant KA 673/2-1, and by SERC Grant GR-E 68297.

In addition we generalize the transformation to apply to any polynomial time algorithm that exactly learns a projection-closed monotone class of boolean formulas using both membership and equivalence queries. The resulting algorithm exactly learns the unate substitution instances of the original class of formulas in polynomial time using membership and equivalence queries. One corollary of this is the existence of a polynomial time algorithm for exact learning of unate DNF formulas using membership and equivalence queries.

As another application of the analysis of minterms and maxterms of a boolean function, we present polynomial time algorithms for learning k-DNF/CNF formulas and depth-k decision trees using membership queries alone. The class of k-DNF/CNF formulas was used in [12].

2 Preliminaries

We consider boolean formulas using a finite set of variables V over the basis (AND, OR, NOT). Let $n = |V|$. By a standard transformation, each nonconstant boolean formula over V can be represented as a rooted tree whose internal nodes are labelled in alternating levels by AND and OR, and whose leaves are labelled by literals from V. A boolean formula is *monotone* if it contains no occurrences of NOT (i.e. no negative literals). A boolean formula is *unate* if no variable appears in both positive and negative literals of the formula.

A read-once formula is a boolean formula over V in which each variable occurs at most once. Read-once formulas are also known as μ-formulas and boolean trees. The tree representation of a read-once formula is unique up to tree isomorphism.

A *read-once function* is a function that can be expressed by a read-once formula. Let $f : \{0,1\}^V \rightarrow \{0,1\}$ be a read-once function over the variables V.

A *minterm* of f is a subset S of literals of V such that setting all the literals in S to 1 forces the value of f to 1, and S is minimal with respect to this property. A *maxterm* of f is a subset T of literals of V such that setting all the literals in T to 0 forces the value of f to 0, and T is minimal with respect to this property.

A membership oracle for the read-once function f answers queries of the form "What is the value of f on A?", where A is an assignment to the variables in V. The oracle answers with the value of f on A.

An equivalence oracle for a read-once function f answers queries of the form "Is g equivalent to f?", where g is an explicit read-once formula. If g represents the function f, the oracle answers "yes". Otherwise, the oracle answers "no" and gives an assignment B such that $f(B) \neq g(B)$ as a counterexample.

We say that an algorithm "exactly learns read-once formulas using membership queries" if given a membership oracle for a read-once function f, and the set V of inputs to f, the algorithm eventually halts and outputs a read-once formula expressing f. The definition of what it means for an algorithm to "exactly learn read-once formulas using membership and equivalence queries" is analogous.

3 Previous results

The learnability of read-once formulas was first studied by Valiant [15]. He proved that it is possible to exactly learn read-once formulas in polynomial time using three powerful types of queries, called relevant possibility queries, necessity queries, and relevant accompaniment queries. Hellerstein and Karpinski [8] subsequently presented a polynomial time algorithm for learning read-once formulas using only relevant possibility queries.

Pitt and Valiant [13] proved that if RP $\neq$ NP, then there is no polynomial time algorithm for learning read-once formulas from examples in the Valiant's distribution-free model (also known as the PAC model). This result depends on the requirement that the output of the learning algorithm be a read-once formula.

The prediction model permits the formulation of representation-independent hardness results. The reductions of Kearns, Li, Pitt, and Valiant [10] show that predicting the value of a monotone read-once formula on randomly chosen examples in the distribution-free model is no easier than predicting general boolean formulas. Kearns and Valiant [11] have shown that prediction of general boolean formulas in the distribution-free model is as hard as certain cryptographic problems, for example, factoring Blum integers.

These results may be interpreted as follows: read-once functions appear to be hard to learn or predict if our only information about them is examples drawn according to an arbitrary distribution. However, if other information in the form of oracles is available, then there are feasible learning algorithms. The relevant question then is: how weak an oracle is sufficient for polynomial time exact learning of the read-once functions?

For the case of monotone read-once functions, membership queries alone suffice for polynomial time exact learning, as Angluin [4] and, independently, Hellerstein and Karpinski [8] showed. In this paper, we give an improved algorithm for this problem.

However, for the whole class of read-once functions, a straightforward adversary argument using the class of 1-term DNF formulas shows that no polynomial time algorithm can exactly learn unrestricted read-once formulas using only membership queries [3]. Also, with a more complex adversary argument, Angluin has shown that there is no polynomial time algorithm for exactly learning read-once formulas using equivalence queries alone [4].

This paper settles the problem of exactly learning unrestricted read-once formulas using both membership and equivalence queries. We present an algorithm for exactly learning read-once formulas in this model that runs in time $O(n^4)$, and makes $O(n^3)$ membership queries and $O(n)$ equivalence queries. Other concept classes learnable in polynomial time with both membership and equivalence queries, but not with membership or equivalence queries alone, include deterministic finite state acceptors and monotone DNF formulas [2,3].

A polynomial time algorithm using equivalence and membership queries implies a polynomial time algorithm in the distribution-free model provided membership queries are also available [2,3]. Equivalence queries can be probabilistically simulated in the PAC model by using random examples to determine (with high probability) whether the hypothesis is approximately equal to the target concept. In this way, a learning algorithm that uses membership and equivalence queries and achieves exact identification may be transformed into one that uses randomly drawn examples and membership queries in the distribution-free model, with a moderate increase in computational cost.

4 An improved algorithm for learning monotone read-once formulas

The results of [4,8] show that there is an algorithm for learning monotone read-once formulas with membership queries alone that runs in time $O(n^3)$ and makes $O(n^3)$ membership queries. We present a new simpler algorithm that runs within the same time bound and uses only $O(n^2)$ membership queries.

Suppose f is a monotone read-once function over the variables V. Consider the tree representation of any read-once formula for f. Let V' be the set of variables that actually occur as leaf labels in this tree representation. The lowest common ancestor in the tree of any pair of distinct variables in V' is labelled either AND or OR.

Our algorithm first determines the set of variables V', and, for each pair of distinct variables in V', the label of the lowest common ancestor of the pair. From this information a monotone read-once formula representing f can be constructed in time $O(n^2)$.

The basic lemma we use is that two distinct variables in V' occur in a common minterm if and only if the label of their lowest common ancestor is AND, and they occur in a common maxterm if and only if the label of their lowest common ancestor is OR. Note that this means that the size of the intersection of a maxterm and a minterm can be at most 1. Since for any boolean function there must be a nonempty intersection between any maxterm and any minterm of the function, this shows that every minterm and maxterm of a read-once function intersect in exactly one literal. (This is the easier direction of an elegant characterization of read-once functions proved by Karchmer, Linial, Newman, Saks, and Wigderson [9].)

The subprocedures used by the algorithm are as follows. Findmin takes as input a set $Q \subseteq V$ that contains a minterm of f and outputs a minterm of f contained in Q. The method is a simple greedy search using membership queries to remove as many variables from Q as possible, while still maintaining the property that Q contains a minterm. This uses $O(n)$ membership queries.

The procedure Xmin takes as input a variable x and a maxterm T of f containing x, and ouputs a minterm of f containing x. *Xmin* consists of one call to Findmin, on the input set $(V - T) \cup \{x\}$. Since $(V - T) \cup \{x\}$ contains a minterm of f, Findmin returns a minterm S of f contained in this set. Since we must have $|S \cap T| = 1$, x must be in S. The procedure Xmax dually uses a minterm of f containing x to find a maxterm of f containing x.

The last procedure used by the algorithm, Checkedge, takes as input two variables x and y, a maxterm T_1 containing x, and a maxterm T_2 containing y. It outputs "yes" or "no" depending whether or not x and y are contained in a common minterm of f. If T_1 or T_2 contain both x and y, then x and y are contained in a common maxterm, and cannot be contained in a common minterm. Otherwise, Checkedge queries the membership oracle on the assignment setting the variables in $((V - (T_1 \cup T_2)) \cup \{x, y\}$ to 1, and the other variables to 0. In this case, x and y are contained in a common minterm if and only if the value of f on this assignment is 1.

The algorithm works by first checking whether f is a constant function. If it is not, it generates a minterm of f using Findmin. Using this minterm as a "seed", it then uses Xmin and Xmax to generate minterms and maxterms until every variable contained in a generated minterm is also contained in a generated maxterm, and vice versa. We prove that when this step of the algorithm is over, every variable in V' is in some minterm or maxterm. The algorithm then uses the generated maxterms as inputs to the procedure Checkedge to determine the labels of lowest common ancestors of pairs of elements of V', and finally constructs and outputs a monotone read-once formula representing f.

5 Learning read-once formulas with membership and equivalence queries

Theorem 1 *There exists an algorithm for learning read-once formulas using membership queries that runs in time $O(n^4)$ and makes $O(n^3)$ membership queries and $O(n)$ equivalence queries.*

Our algorithm for learning read-once formulas with membership and equivalence queries uses as a subroutine any algorithm for learning monotone read-once formulas using only membership queries. Using the new algorithm for this problem given above, we achieve the bounds stated.

It is clear that if f is a read-once formula defined on the variable set V, then if we know the signs of the variables appearing in f (i.e., whether the variables appear in positive or negative literals of f) then we can use the monotone membership query algorithm to learn f. The difficulty is to learn the signs of the variables in V.

We do so by learning appropriate projections of f. A projection of f is a formula that is obtained from f by fixing the value of some of the variables in f to constants. Each projection of f is associated with a partial assignment to the variables of V. Formally, a partial assignment P is a mapping from V to the set $\{0, 1, *\}$. The variables mapped to $*$ are left unassigned. The variables not mapped to $*$ are said to be in the *defined set* of P. Let f_P denote the projection of f induced by P. Although f_P does not in general depend on all the variables in V, we will consider f_P to be defined on V. For any assignment A to the variables of V, let A/P denote the (total) assignment that sets the variables in the defined set of P according to P, and sets the other variables according to A. Given P, one can use the membership oracle for f to simulate a membership oracle for f_P, because $f_P(A) = f(A/P)$.

Our algorithm makes use of the following lemma.

Lemma 2 *Let f be an unknown read-once formula defined on the variable set V. Let A and B be two assignments to the variables of V, such that $f(A) = 0$ and $f(B) = 1$. Then in at most $\lceil \log n \rceil$*

membership queries, it is possible to learn the sign of a variable x in f such that the setting of x differs in A and B.

The algorithm keeps a set W containing the set of variables in V whose signs it has learned so far. Initially, the algorithm does not know the signs of any variables in V, so W is empty. The algorithm also keeps a set W_{neg} containing all the variables in W occurring in negative literals of f.

The algorithm executes the following loop. At the start of the loop, it chooses an arbitrary partial assignment P whose defined set is $V - W$. The projected function f_P depends only on variables in W. Therefore, the algorithm knows the signs of all the variables on which f_P depends. The algorithm runs the membership query algorithm for monotone read-once formulas to learn f_P, simulating each query to the membership oracle for f_P by an appropriate query to the membership oracle for f. The algorithm presents f_P as input to the equivalence oracle, asking whether $f_P = f$. If the answer is "yes", then the algorithm outputs f_P and terminates. Otherwise, the answer is "no" and a counterexample B such that $f_P(B) \neq f(B)$. But $f_P(B) = f(B/P)$, so $f(B/P) \neq f(B)$. The assignments B and B/P differ only in variables in the defined set of P, that is, in variables in $V - W$. Using at most $\lceil \log n \rceil$ membership queries, the algorithm determines the sign in f of a variable x in $V - W$ The variable x is added to W, and to W_{neg} if f is in a negative literal of f. The algorithm then iterates the loop.

In at most $|V| + 1$ iterations of the loop the algorithm will terminate and output a read-once formula expressing f. Since $|V| = n$, the bounds follow.

6 Generalization of the transformation

In the previous section, we showed how to transform an algorithm for learning monotone read-once formulas using membership queries into an algorithm for learning unrestricted read-once formulas using membership and equivalence queries. In this section we generalize the transformation. As a corollary, we will show that the class of unate DNF formulas is learnable in polynomial time using membership and equivalence queries.

Let M be a class of monotone boolean formulas. M is closed under projection if for every formula $f \in M$ and every partial assignment P to the variables of f, there is a formula in M equivalent to f_P.

If f is any monotone boolean formula defined on V, let $U(f)$ denote the class of all formulas f' obtained from f by selecting a subset V' of V and replacing every occurrence of x in f by $\neg x$ for all $x \in V'$. All the elements of $U(f)$ are unate. If M is a class of monotone boolean formulas, let $U(M)$ denote the union of $U(f)$ for all $f \in M$. Note that if M is the class of monotone read-once formulas then $U(M)$ is the class of unrestricted read-once formulas. We prove the following theorem.

Theorem 3 *Let M be a class of monotone formulas closed under projection such that there is a* *polynomial time algorithm for learning M using membership and equivalence queries. Then there is also a polynomial time algorithm for learning $U(M)$ using membership and equivalence queries.*

Note that this theorem strengthens the transformation in the previous section by allowing equivalence queries to be used in the learning algorithm for M.

As in the case of read-once functions, the problem is to learn the signs of the variables in the unknown formula $f \in U(M)$. It is clear that if we know the signs of the variables, we can use the algorithm for learning M with membership and equivalence queries to learn $U(M)$ with membership and equivalence queries.

Let A^{signs} be an algorithm that learns $U(M)$ using membership queries provided it is given the signs of the variables in the unknown formula as input. As in the preceding section, we use this algorithm to attempt to learn projections of f for which we know the signs; the major difference is that we must now handle equivalence queries made by A^{signs}.

The algorithm to learn $U(M)$ works as follows. Let f be the unknown formula. Let W be the set of variables of f whose signs are known, and let W_{neg} be the set of variables $x \in W$ such that f is negative in x. Initially, W and W_{neg} are empty.

The algorithm executes the following loop at most $n+1$ times. It chooses an arbitrary partial assignment P whose defined set is $V-W$, and runs the procedure A^{signs} with input W_{neg} simulating an oracle for f_P as follows. When A^{signs} makes a query to the membership oracle for f_P on input A, the algorithm simulates that query using the membership oracle for f. When A^{signs} makes an equivalence query "Is $f_P \equiv g$?" the algorithm queries the equivalence oracle for f on input g. If the answer is "yes", then the algorithm outputs g and terminates. Otherwise, the reply is a counterexample B such that $f(B) \neq g(B)$. Using the membership oracle for f, the algorithm finds out the value of $f_P(B)$. The algorithm knows g, so it can compute $g(B)$. If $f_P(B) \neq g(B)$, then the algorithm returns "no" and the counterexample B to the equivalence query asked by A^{signs}. Otherwise, $f(B)$ does not equal $f_P(B)$, and so by Lemma 2 it can use at most $\lceil \log n \rceil$ membership queries to the oracle for f to determine the sign of a variable x not in W. In this case, the algorithm adds x to W (and to W_{neg} if its sign is negative), gives up the current simulation of A^{signs}, and iterates the loop with the new values of W and W_{neg}.

If the algorithm succeeds in answering all the queries of A^{signs} then A^{signs} returns a formula g equivalent to f_P. The algorithm makes an equivalence query with g. If the reply is "yes", then it outputs g and terminates. Otherwise, the reply is a counterexample B such that

$$f(B) \neq g(B) = f_P(B) = f(B/P),$$

so, as before, it uses B and B/P to determine the sign of a variable x not in W, adds x to W_{neg} if appropriate, and iterates the loop. This completes the description of the algorithm.

Let M denote the class of monotone DNF formulas. Then there is a polynomial time algorithm for learning M in polynomial time using membership and equivalence queries [3]. M is closed under projection and $U(M)$ is the class of unate DNF formulas. Thus, one corollary of the theorem above is the following.

Corollary 4 *There exists a polynomial time algorithm for learning unate DNF formulas using membership and equivalence queries.*

7 Learning k-DNF/CNF formulas and depth–k decision trees with membership queries

A k-DNF/CNF formula is a DNF formula whose terms all contain at most k literals, that has an equivalent CNF formula whose clauses all contain at most k literals. The class of k-DNF/CNF formulas was used in [12].

As a further application of the technique of analyzing the intersections of minterms and maxterms of boolean functions, we show an $O(2^{2k}n^{2k})$ algorithm for exactly learning k-DNF/CNF formulas using only membership queries. This algorithm can also be used to exactly learn depth–k decision trees by depth–k^2 decision trees.

Our algorithm relies on the fact that if $S_1, \ldots, S_z$ are the terms of a DNF formula f, then the maxterms of f are precisely the minimal sets of literals containing at least one variable from each of the terms.

Let f be a k-DNF/CNF formula representing the function F. Then f is a DNF formula with at most k literals per term. Also, there exists a formula g in CNF with at most k literals per clause that is equivalent to f, and therefore also represents F.

Each term of f is a superset of a minterm of F and each clause of g is a superset of a maxterm of F. Since every minterm of F intersects with every maxterm of F, every term of f intersects with every clause of g.

Let S be any set of at most k literals. If S does not contain a minterm of F, then S has an empty intersection with the set of literals T in one of the clauses of g. The value of F is 0 on any assignment setting the literals in S to 1 and the literals in T to 0. T contains at most k literals.

Thus, to discover whether S contains a minterm of F, it is sufficient to check, for all possible sets T of size at most k not intersecting S, whether the value of F is 0 on an arbitrary assignment setting the literals in S to 1 and the literals in T to 0. A k-DNF/CNF formula for f can be formed by taking the disjunction of terms made up of all sets of literals of size at most k that contain a minterm of f. We have the following theorem.

Theorem 5 *There exists an algorithm that exactly learns k-DNF/CNF formulas that runs in time $O(2^{2k}n^{2k})$ and makes $O(2^{2k}n^{2k})$ membership queries.*

Every depth–k decision tree computes a function expressible by a k-DNF/CNF formula. Several authors [6,7,14] have independently shown that any function computed by a k-DNF/CNF formula can be computed by a depth–k^2 decision tree. As a consequence we have the following.

Corollary 6 *There is an algorithm that exactly learns depth–k decision trees by depth-k^2 decision trees that runs in time $O(2^{2k}n^{2k} + 2^{k^2})$ and makes $O(2^{2k}n^{2k})$ membership queries.*

8 Open problems

A natural question is what other nontrivial classes of boolean formulas are exactly learnable in polynomial time with membership and equivalence queries. Previous results have shown that monotone DNF formulas [3] and k-term DNF formulas [1] are learnable in polynomial time in this setting. The question is open for general DNF formulas, even if we permit subset and superset queries in addition to membership and equivalence queries.

Another important open problem is to learn read-once straight line or branching programs over finite or characteristic zero fields. Also, it is an interesting question whether our algorithms for exact learning can be speeded up via randomization. We are not asking here for algorithms in the PAC model, but rather for algorithms that achieve exact identification with controllably high probability (Monte Carlo).

There is a simple counting argument to show that any algorithm that exactly learns all monotone read-once formulas in n variables using only membership queries must use $\Omega(n \log n)$ membership queries. The algorithm we give above achieves $O(n^2)$ membership queries - perhaps an improvement is possible.

9 Acknowledgements

It is a pleasure to thank Sally Floyd, Dick Karp, Lenny Pitt, Ron Rivest, and Manfred Warmuth for helpful discussions of this material.

References

[1] D. Angluin. Learning k-term DNF formulas using queries and counterexamples. Technical report, Yale University, YALE/DCS/RR-559, 1987.

[2] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

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

[4] D. Angluin. Using queries to identify μ-formulas. Technical report, Yale University, YALE/DCS/RR-694, 1989.

[5] D. Angluin, L. Hellerstein, and M. Karpinski. Learning read-once formulas with queries. U.C. Berkeley Technical Report UCB CSD 89-258 and International Computer Science Institute Technical Report TR 89-021, 1989.

[6] M. Blum and R. Impagliazzo. Generic oracles and oracle classes. In *Proc. 28th IEEE Symposium on Foundations of Computer Science*, pages 118–126. IEEE, 1987.

[7] J. Hartmanis and L. Hemachandra. One-way functions, robustness, and non-isomorphism of NP-complete sets. Technical report, Cornell University Technical Report DCS TR86-796, 1987.

[8] L. Hellerstein and M. Karpinski. Learning read-once formulas using membership queries. In *Proc. of the Second Annual Workshop on Computational Learning Theory*, pages 146–161. Morgan Kaufmann Publishers, 1989.

[9] M. Karchmer, N. Linial, I. Newman, M. Saks, and A. Wigderson. Combinatorial characterization of read once formulae. Presented at the Joint French-Israeli Binational Symposium on Combinatorics and Algorithms, 1988.

[10] M. Kearns, M. Li, L. Pitt, and L. Valiant. On the learnability of boolean formulae. In *Proc. 19th ACM Symposium on Theory of Computing*, pages 285–295. ACM, 1987.

[11] M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. In *Proc. 21st ACM Symposium on Theory of Computing*, pages 433–444. ACM, 1989.

[12] N. Linail, Y. Mansour, and N. Nisan. Constant depth circuits, Fourier transform, and learnability. In *Proc. 30th IEEE Symposium on Foundations of Computer Science*, pages 574–579. IEEE, 1989.

[13] L. Pitt and L. Valiant. Computational limitations on learning from examples. *J. ACM*, 35:965–984, 1988.

[14] E. Tardos. Query complexity, or why is it difficult to separate $NP^A \cap co-NP^A$ from P^A by a random oracle A? Manuscript., 1988.

[15] L. G. Valiant. A theory of the learnable. *C. ACM*, 27:1134–1142, 1984.