Learning Read-Once Formulas Using Membership Queries

Lisa Hellerstein[1] and Marek Karpinski[2]

TR-89-021

April, 1989

Abstract

In this paper we examine the problem of exact learning (and inferring) of read-once formulas (also called μ-formulas or boolean trees) using membership queries. The power of membership queries in learning various classes of formulas was studied by Angluin [A]. Valiant proved that, using three powerful oracles, read-once formulas can be learned in polynomial time [V]. Pitt and Valiant proved that if RP $\neq$ NP, read-once formulas cannot be learned by example in polynomial time [PV, KLPV]. We show that given explicitly a boolean formula f defining a read-once function, if $RP \neq NP$, then there does not exist a polynomial time algorithm for inferring an equivalent read-once formula. An easy argument on the cardinality of the set of all (read-once) 1-term DNF formulas implies an exponential lower bound on the number of membership queries necessary to learn *read-once* formulas. Angluin showed that it takes time $2^{\Omega(n)}$ to learn *monotone* n-term DNF formulas using membership queries [A]. We prove that, surprisingly, it is possible to learn *monotone read-once* formulas in polynomial time using membership queries. We present an algorithm that runs in time $O(n^3)$ and makes $O(n^3)$ queries to the oracle. It is based on a combinatorial characterization of read-once formulas developed by Karchmer et. al. [KLNSW]. We also use the combinatorial characterization to prove two other results. We show that read-once formulas can be learned in polynomial time using only one of the three oracles used in Valiant's polynomial time algorithm. In addition, we show that given an arbitrary boolean formula f, the problem of deciding whether f defines a read-once function is complete in the class D^P under randomized NC^1-reductions. The main results of this paper can also be interpreted in terms of efficient input oracle algorithms for boolean function interpolation (cf. [KUW], [GKS]).

---

1 Computer Science Division, University of California, Berkeley, hstein@ernie.berkeley.edu. Supported by NSF Grant CCR-8411954.

2 International Computer Science Institute, Berkeley, California, marek@icsi.berkeley.edu, on leave from the University of Bonn. Research partially supported by DFG Grant KA 673/2-1, and by SERC Grant GR-E 68297.

1. Introduction

Let $g:\{0,1\}^V \rightarrow \{0,1\}$ and let $|V|=n$. g is called a *read-once function* if g can be written as a *read-once formula,* that is, as a Boolean formula (with AND, OR, NOT as a basis) in which every variable in V appears at most once. Read-once formulas are also called μ-formulas. Every read-once formula can be represented as a tree in which the internal nodes of the tree contain AND and OR in alternating levels, and the leaves contain the variables (possibly in negated form). We consider two read-once formulas to be identical if their tree representations are isomorphic. There is a unique read-once formula for every read-once function.

How hard is it to learn read-once formulas? Valiant defined three powerful oracles and showed that there exists an algorithm for exactly learning read-once formulas that makes $O(n^3)$ calls to the oracles [V]. Pitt and Valiant proved that if RP $\neq$ NP, it is not possible to learn read-once formulas by example in polynomial time [PV,KLPV]. Their result also applies to learning monotone read-once formulas.

In this paper we consider the complexity of exact learning of both monotone read-once formulas and arbitrary (not necessarily monotone) read-once formulas using a membership oracle. A membership oracle is an oracle which takes as input an assignment to the variables of the formula, and outputs the value of the formula on that assignment. The power of the membership oracle in learning various classes of formulas was studied by Angluin [A]. We note that the results of this paper can also be interpreted in terms of efficient input oracle algorithms for boolean function interpolation (cf. [KUW], [GKS]). The algorithm developed here gives the first efficient interpolation algorithm for a monotone read-once boolean function given by a (black box) input oracle (for the corresponding results using field extensions and (XOR, AND) formulas cf. [GKS]).

In Section 2 we note that it takes time $\Omega(2^n)$ to learn arbitrary *read-once* formulas using membership queries. We also show that given explicitly a boolean formula f defining a read-once function, if $RP \neq NP$, then there does not exist a polynomial time algorithm for inferring an equivalent read-once formula. Angluin proved that it takes time $2^{\Omega(n)}$ to learn *monotone* n-term DNF formulas using membership queries [A]. We prove that, in contrast, *monotone read-once* formulas can be learned in polynomial time using membership queries. In Section 5 we present an algorithm for learning monotone read-once formulas using membership queries that takes time $O(n^3)$ and makes $O(n^3)$ queries.

Our algorithm is based on a combinatorial characterization of read-once formulas developed by Karchmer, Linial, Saks, and Wigderson [KLNSW]. In Section 4 we discuss an algorithm for learning arbitrary read-once formulas using a *relevant possibility oracle* that is easily derived from the characterization. The algorithm is an improvement over Valiant's polynomial algorithm because it uses only one of the three oracles that Valiant uses, and it makes fewer oracle calls. In Section 2, we use the characterization to

show that given an arbitrary boolean formula f, the problem of deciding whether f defines a read-one function is complete in the class D^P under randomized NC^1-reductions.

2. Learning Equivalent Read-Once Formulas From Explicitly Given Formulas

An easy argument on the cardinality of the set of all (read-once) 1-term DNF formulas implies an $\Omega(2^n)$ lower bound on the number of membership queries necessary to learn arbitrary read-once formulas. The computational difficulty of inferring an equivalent read-once formula from an explicitly given boolean formula is characterized by the following theorem.

Theorem 1: Given a boolean formula f defining a read-once function, if $RP \neq NP$, then there does not exist a polynomial time algorithm for inferring an equivalent read-once formula.

Proof: A boolean formula with a unique satisfying assignment can be rewritten as a read-once formula consisting of one term corresponding to the satisfying assignment. Valiant and Vazirani proved that if $RP \neq NP$, there is no algorithm for finding solutions to instances of SAT having unique solutions [VV]. □

We define now the complexity class D^P (cf. [PY]).

$$D^P = \{ L_1 - L_2 \mid L_1, L_2 \in NP \}$$

The proof of the next theorem relies on the development of Section 3. For the definition of randomized NC-reductions, see [C], [KR].

Theorem 2: Given an arbitrary boolean formula f, the problem of deciding whether f defines a read-once function is complete for D^P under randomized NC^1-reductions.

Proof: Suppose f is a boolean formula on the variables $x_1,\ldots,x_n$. Suppose that f either defines a constant function 0, or f has a unique satisfying assignment. Construct a new formula $F = f + y*x_1 + y*x_2 + \cdots + y*x_n$ for $y \notin \{ x_1,\ldots,x_n \}$ (where + and * represent AND and OR). We prove that f is satisfiable iff F does not define a read-once function.

Case 1: $f = 0$, F is read-once.

Case 2: f has the unique satisfying assignment $x_1 = x_2 = \cdots = x_n = 1$. In this case, $\{ x_1, x_2, \ldots, x_n \}$ is a minterm of F, but $\{ x_1, x_2, \ldots, x_n \}$ is not a maximal clique in the minterm graph of F. Hence F is not read-once. (cf. Section 3)

Case 3: f has a unique satisfying assignment different from Case 2. In this case, the satisfying assignment contains a zero at a certain variable x_k. From the construction of F, any equivalent boolean formula must contain the variable x_k in negated and non-negated form. Therefore F is not a read-once function.

The general problem of checking satisfiability of the function f (UNIQUE SAT restricted to 0 or 1 solutions) is complete for D^P using randomized NC^1-reductions ([VV]). □

3. Preliminaries

In a read-once formula, each variable in the formula appears exactly once, either negated, or not negated. In the following section, we describe properties of monotone read-once functions. The properties can be applied to an arbitrary read-once function f by replacing every variable that appears in negated form in the read-once formula for f with a new variable that represents the negation of the original variable.

One difficulty in learning arbitrary read-once formulas is to determine for each variable whether it appears in the read-once formula in its negated, or non-negated form. If this can be determined, the problem can usually be reduced to the monotone case (depending on which oracles are used).

Let $f : 2^V \rightarrow \{ 0,1 \}$.

A subset S of negated and non-negated variables of V (no variable occuring both in negated and non-negated form) is called a *minterm* of f if setting all the non-negated variables of S to 1 and setting the negated variables of S to 0 forces the value of f to 1, and this property does not hold for any $S' \subseteq S$. A subset T of negated and non-negated variables of V is called a *maxterm* of f if setting all the negated variables of T to 1 and setting all the non-negated variables of T to 0 forces the value of f to 0, and this property does not hold for any $T' \subset T$. If f is monotone, every minterm is a subset of V, and every maxterm is a subset of V.

Assume f is a monotone read-once function.

Let V' denote the subset of variables of V on which the value of f depends. We define the *minterm graph* of f to be the graph whose vertex set is V', and whose edge set consists of all $\{ x,y \} \subseteq V'$ such that there exists a minterm S of f such that $\{ x,y \} \subseteq S$. We define the *maxterm graph* of f to be the graph whose vertex set is V', and whose edge set consists of all $\{ x,y \} \subseteq V'$ such that there exists a maxterm T of f such that $\{ x,y \} \subseteq T$.

P4 is the chordless path on four vertices. A graph is said to be *P4-free* if it does not

contain P4 as an induced subgraph.

Karchmer, Linial, Newman, Saks, and Wigderson, developed a combinatorial characterization of read-once functions. Their results include the following.

Lemma 1 [KLNSW]: If f is a monotone read-once function, and G is its minterm graph, then the minterms of f are the sets of vertices composing the maximal cliques of G, and the maxterms of f are the sets of vertices composing the maximal independent sets of G. Furthermore, G is P4-free.

Lemma 1 can be proved by induction on the read-once formula for f.

The following proposition follows directly from Lemma 1.

Proposition 1: If f is a monotone read-once function, and V' is the set of variables on which f depends, then for every $\{x,y\} \subseteq V'$, either there exists a minterm S of f such that $\{x,y\} \subseteq S$, or there exists a maxterm T of f such that $\{x,y\} \subseteq T$.

Theorem 3 [KLNSW]: f is read-once iff for all minterms S of f, and for all maxterms T of f, $|S \cap T|=1$.

We will use only the easier direction of this theorem, that is, if f is read-once, then $|S \cap T|=1$. Like Lemma 1, this direction of the theorem can be proved by induction on the read-once formula for f.

Karchmer et. al. also showed that given the minterm graph of f, it is easy to construct the read-once formula for f. Their construction makes use of the following lemma.

Lemma 2 [S]: If a P4-free graph containing more than one vertex is connected, then its complement is disconnected.

The construction of the read-once formula for f is recursive. Let G be the minterm graph of f. If G contains one vertex, then the construction is trivial. Assume G contains more than one vertex. Suppose G is disconnected. Let $V'_1, V'_2, \cdots V'_q$, be the sets of vertices in the q connected components of G. For $i \in \{1,2,...,q\}$, let f_i be the monotone boolean function whose minterm set consists of all minterms S of f such that $S \subset V'_i$. f_i is the function induced from f by setting all variables in $V'-V'_i$ to 0. Since f is expressible by a read-once formula, f_i is expressible by a read-once formula. Recursively construct read-once formulas $F_1,...,F_q$ for $f_1,...,f_q$. $F_1+F_2+\cdots+F_q$ is a read-once formula for f.

Suppose now that G is connected. Then by Lemma 2 the complement of G is disconnected. By Lemma 1 the complement of G is the maxterm graph of f. Therefore, the

maxterm graph of f is disconnected. A dual construction to the one above produces a read-once formula for f.

It follows from the above construction that given the minterm graph for a read-once function f, the read-once formula for f can be constructed in time $O(n^3)$ by repeatedly applying depth first search to find connected components.

4. Learning Read-Once Formulas with a Relevant Possibility Oracle

Relevant possibility oracles are one of the three types of oracles used in Valiant's algorithm for learning read-once formulas ([V]). A relevant possibility oracle takes as input a subset S' of negated and non-negated variables of f, and outputs 1 iff there exists a minterm S of f such that $S' \subseteq S$.

It follows from the previous section that if F is a read-once monotone formula, f is the function it represents, and we have enough information to construct the minterm graph of f, then we have enough information to construct F. To construct the minterm graph, we need the following information. For every variable x in V, we must know whether f depends on x (i.e. is x a vertex in the minterm graph?). For every pair of variables $\{x,y\}$ of V, such that f depends on x and y, we must know whether x and y appear together in a minterm of f (i.e. is there an edge between x and y in the minterm graph?). Each of these questions can be answered with a query to a relevant possibility oracle. In the case where F is not necessarily monotone, one can use the relevant possibility oracle to determine whether a variable appears in its negated or non-negated form in F. This implies that the minterm graph can be constructed in time $O(n^2)$. The read-once formula can then be constructed in time $O(n^3)$. This gives us the following theorem, which improves Valiant's result [V].

Theorem 4: There exists an algorithm for learning read-once formulas with a relevant possibility oracle that runs in time $O(n^3)$ and makes $O(n^2)$ oracle calls.

5. Learning Monotone Read-Once Formulas Using Membership Queries

In this section we present two polynomial time algorithms for learning *monotone* read-once formulas using membership queries. The first algorithm is presented for its conceptual transparency. The second algorithm improves its complexity to time $O(n^3)$ by introducing a more efficient termination condition. (The reader is advised to read Algorithm 1 first.)

Our algorithms work by constructing an exact minterm graph. Constructing a minterm graph using membership queries is much more difficult than constructing it using a relevant possibility oracle.

Let F be a monotone read-once formula expressing a function f of the variables V. We use the following proposition, which follows from Lemma 1.

Proposition 2: To construct the minterm graph of f, it is sufficient to determine a subset MIN of the minterms of f, and a subset MAX of the maxterm of f, with the following two properties:

Property 1: Every variable on which f depends (i.e. every variable that appears in F) is contained in a minterm in MIN, or a maxterm in MAX.

Property 2: Every pair of variables $\{x,y\}$ on which f depends is contained either in a common minterm in MIN, or a common maxterm in MAX.

5.1. The algorithms

We present two algorithms to construct MIN and MAX. Given MIN and MAX, we can construct the minterm graph of f and use it to determine F.

Let *Query*(A) denote the output of the membership oracle on the string in which all members of A are set to 1, and all members of $V-A$ are set to 0. Our algorithms make repeated use of the following procedure to find a minterm contained in W, provided that one exists.

```
Findmin(W)
{  S := W
   for each w ∈ W {
      if Query(S - {w}) = 1 then
            S:= S - {w} }
   return(S)
}
```

A dual procedure, Findmax(W), finds a maxterm in W provided one exists.

We present Algorithm 1 for finding MIN and MAX together with brief comments as to what each step is supposed to accomplish. Following the presentation of Algorithm 1, we discuss the correctness of the individual steps of the algorithm, and give a bound on the number of oracle queries performed by the algorithm. We then prove that upon termination of Algorithm 1, MIN and MAX have Properties 1 and 2.

Algorithm 1:

```
1. MIN := ∅          /initialize/
   MAX := ∅
```

2. if Query(V) = 0 then terminate /check if f is identically 0/

3. MIN := MIN $\cup$ {Findmin(V)} /put one minterm into MIN/

4. CONTINUE := true.
  while CONTINUE = TRUE do {
    if there exists an x $\in$ V such that x is contained in a maxterm T of MAX,
      but not in a minterm of MIN, then {
4a. MIN := MIN $\cup$ {Findmin((V - T) $\cup$ {x})}
      } /adds to MIN a minterm containing x/
    else if there exists an x in V such that x is contained in a minterm S of MIN
    but not in a maxterm of MAX, then {
4b. MAX := MAX $\cup$ {Findmax((V - S) $\cup$ {x})}
      } /adds to MAX a maxterm containing x/
    else if there exist {x,y} in V such that
      x is contained in a minterm S_1 of MIN, and a maxterm T_1 of MAX,
      and y is contained in a minterm S_2 of MIN, and a maxterm T_2 of MAX,
      but x and y do not occur in a common minterm of MIN or maxterm of MAX
      then {
4c. if Query((V - ($T_1 \cup T_2$)) $\cup$ {x,y}) = 1 then
        MIN := MIN $\cup$ {Findmin((V - ($T_1 \cup T_2$)) $\cup$ { x,y })}
          /adds to MIN a minterm containing x and y/
        else MAX := MAX $\cup$ {Findmax((V - ($S_1 \cup S_2$)) $\cup$ {x,y})}
      } /adds to MAX a maxterm containing x and y/
4d. else CONTINUE := FALSE
  }

Steps 1 through 3 of Algorithm 1 are straightforward. In Step 4a, x is contained in a maxterm T of MAX, but not in a minterm of MIN. The purpose of Step 4a is to add a minterm of f containing x to MIN. There must exist a minterm of f containing x, because every variable upon which f depends is contained in some minterm of f. Let S' be a minterm of f containing x. f is read-once, so by Theorem 3, $|S' \cap T|=1$. Therefore, the only element shared by S' and T is x. Because S' is a minterm of f, if we set all the variables of S' to 1, we force the value of f to be 1. Certainly, if we set the variables of S' to 1, and then set any (not already set) variables in $V-T$ to 1 the value of f is still forced to be 1. Therefore, $S' \cup (V-T)=(V-T) \cup \{x\}$ contains a minterm of f. Note that the only member of T contained in $(V-T) \cup \{x\}$ is x. If S is a minterm of f contained in $(V-T) \cup \{x\}$, then by Theorem 3 $|S \cap T|=1$, and therefore $S \cap T=\{x\}$. Thus any minterm of f contained in $(V-T) \cup \{x\}$ must contain x. It follows that in Step 4a a minterm containing x is added to MIN. Similarly, in Step 4b a maxterm containing x is added to MAX.

In Step 4c we are trying either to add a minterm to MIN that contains x and y, or to

add a maxterm to MAX that contains x and y. This step uses the same methods as Step 4a. By Proposition 1, x and y occur either in a common minterm of f, or in a common maxterm of f. We first show that if there exists a minterm of f containing x and y, then the if-statement in Step 4c evaluates to true. We then show that if the if-statement evaluates to true, a minterm of f containing x and y is added to MIN. Therefore, the if-statement evaluates to true iff there exists a minterm of f containing x and y.

Suppose there exists a minterm S' of f containing x and y. By Theorem 3, $|S' \cap T_1|=1$ and $|S' \cap T_2|=1$. Therefore, $S' \cap T_1=\{x\}$, and $S' \cap T_2=\{y\}$. Because S' is a minterm of f, setting all the variables of S' to 1 forces the value of f to 1. f is monotone, so setting all the variables of S' to 1, and then setting any unassigned variables in $V-(T_1 \cup T_2)$ to 1 also forces the value of f to 1. Therefore, setting all elements of $V-(T_1 \cup T_2) \cup \{x,y\}$ to 1, and all remaining elements of f to 0 makes the value of f equal 1. It follows that if f has a minterm containing x and y, *Query* $((V-(T_1 \cup T_2)) \cup \{x,y\})=1$ and the if-statement in Step 4c evaluates to true.

Whenever the if-statement in Step 4c evaluates to true, $(V-(T_1 \cup T_2)) \cup \{x,y\}$ must contain a minterm of f. The only elements of T_1 and T_2 contained in $(V-(T_1 \cup T_2)) \cup \{x,y\}$ are x and y. Every minterm of f must intersect T_1 and T_2 in exactly one element. Therefore, every minterm contained in $(V-(T_1 \cup T_2)) \cup \{x,y\}$ must contain x and y. It follows that when the if-statement in Step 4c evaluates to true, a minterm of f containing x and y is added to MIN.

If the if-statement in Step 4c evaluates to false, x and y are not contained in a common minterm of f. By Proposition 1, x and y are contained in a common maxterm of f. A similar argument to the one above proves that a maxterm containing x and y is added to MAX in this case.

We have now verified that the individual steps of Algorithm 1 perform as claimed. It remains to prove that upon termination of the algorithm, MIN and MAX actually have Properties 1 and 2. Let V' denote the set of variables that appear in minterms in MIN or maxterms in MAX. Steps 4a and 4b guarantee that if x is in V', then x appears in both a minterm of MIN and a maxterm of MAX. Step 4c guarantees that if x and y are variables in V', then x and y either appear together in a minterm of MIN, or in a maxterm of MAX. Therefore, Properties 1 and 2 hold for MIN and MAX provided that V' is the set of variables on which f depends.

Let G be the minterm graph of f. Let G' be the subgraph of G induced by V'. We will show that $G'=G$, and hence V' is the set of variables on which f depends. Suppose $G' \neq G$. Then G, G', MAX and MIN have the following properties. The properties follow from Lemma 1, and the correctness of Step 4 of the algorithm.

1'. The minterms in MIN are maximal cliques of G.

2'. The maxterms in MAX are maximal independent sets of G.
3'. G' is a proper induced subgraph of G.
4'. G is P4-free.
5'. The minterms in MIN are maximal cliques of G', and cover the vertices and edges of G'.
6'. The maxterms in MAX are maximal independent sets of G', and cover the vertices and non-edges of G'.

In the following lemma we show a contradiction. We prove that if G', G, MAX, and MIN have Properties 3' through 6', then they cannot have both properties 1' and 2'. Therefore, our assumption that $G' \neq G$ must be false. It follows that $G'=G$, and therefore Properties 1 and 2 hold for MIN and MAX upon termination of the algorithm.

Lemma 3: If G is a P4-free graph, and G' is a proper induced subgraph of G, then if MIN is a set of maximal cliques of G' covering the vertices and edges of G', and MAX is a set of maximal independent sets of G' covering the vertices and non-edges of G', then either MIN contains a clique that is not a maximal clique of G, or MAX contains an independent set that is not a maximal independent set of G.

Proof: By induction on the number of vertices in G. In the base case, G contains two vertices. G' must consist of one vertex. MIN must contain one maximal clique consisting of that one vertex, and MAX must contain one maximal independent set consisting of that one vertex. If G has an edge, then MIN contains a clique that is not maximal in G. If G doesn't have an edge, then MAX contains an independent set that is not maximal in G. Therefore, the theorem is true when G has two vertices.

Inductive Step: Let $k>2$. Assume the theorem is true provided that G has fewer than k vertices. Show it is true when G has k vertices.

Assume the theorem is false for some P4-free graph G with k vertices. Let G' be a subgraph of G for which the theorem is false. Let MAX and MIN be the associated sets. The cliques of MIN cover the vertices and edges of G'. The independent sets of MAX cover the vertices and edges of G'. It follows from our assumption the the theorem is false for G, G', MAX, and MIN, that MAX consists of maximal independent sets of G, and MIN consists of maximal cliques of G. Let V be the vertices of G. Let V' be the vertices of G'. Let x be a vertex of $V-V'$.

Consider the subgraph of G induced by $V' \cup \{ x \}$. Call it G''. G is P4-free, and hence G' and G'' are P4-free. The cliques of MIN are maximal in G'', and the independent sets of MAX are maximal in G''. If $|V' \cup \{ x \}| < k$, then G', G'', MIN, and MAX contradict the inductive assumption that the theorem is true when the graph has fewer than k vertices. Therefore, $|V' \cup \{ x \}| = k$, and hence $V' \cup \{ x \} = V$ and $G''=G$.

By Lemma 2, either G' is disconnected, or the complement of G' is disconnected.

Case 1: Suppose G' is disconnected, and G is also disconnected. Let C denote the vertices in the connected component of G that contains x. Let G_1 denote the subgraph of G induced by C. Let G'_1 denote the subgraph of G induced by $C-\{x\}$. $|C|<|V|=k$. Let $MAX'=\{T\bigcap(C-\{x\})|T\in MAX\}$. Let $MIN'=\{S|S\in MIN, S\subseteq C-\{x\}\}$. The independent sets in MAX cover all the vertices and non-edges of G', and hence MAX' covers all the vertices and non-edges of G'_1. The cliques in MIN' cover all the vertices and edges of G'_1. MIN contains maximal cliques of G' and therefore the cliques in MIN' are maximal in both G_1 and G'_1. The independent sets of MAX are maximal in both G' and G. Let T be an independent set in MAX. Because G is disconnected, $T=T_1\bigcup T_2$, where T_1 is a maximal independent set of G_1, and T_2 is a maximal independent set the subgraph of G induced by $V-C$. G' does not contain x, and therefore $T_1\subseteq C-\{x\}$. Hence, $T\bigcap(C-\{x\})=T_1$, and $T\bigcap(C-\{x\})$ is a maximal independent set of both G_1 and G'_1. It follows that the independent sets in MAX' are maximal in both G_1 and G'_1. G_1, G'_1, MIN', and MAX' contradict the inductive assumption that the theorem is true for graphs with fewer than k vertices.

Case 1a: Suppose G' is disconnected and G is connected. Then, in G, x is adjacent to a vertex in each of the connected components of G'. It follows from the fact that G is P4-free that x must be connected to all the vertices in G'. Therefore, every maximal clique of G contains x. Let S be a clique in MIN. S is maximal in G'. S does not contain $\{x\}$. Therefore, S is not maximal in G. This contradicts the assumption that the cliques in MIN are maximal in both G and G'.

Case 2: A dual argument proves the inductive step when the complement of G' is disconnected. □

Each call to Findmin or Findmax makes $O(n)$ calls to the membership oracle. The loop in Step 4 is executed at most $n+\binom{n}{2}$ times--once for each $x\in V$, and once for each pair $\{x,y\}\subseteq V$. Therefore, Algorithm 1 makes $O(n^3)$ queries, and runs in polynomial time.

We now present Algorithm 2 for producing MIN and MAX, which runs in time $O(n^3)$. The main difference between Algorithms 1 and 2 is that in Algorithm 2, we may (intentionally) produce more minterms and maxterms than actually necessary to determine the minterm graph. In particular, in Step 4c of Algorithm 2 (in contrast to Step 4c of Algorithm 1), we may produce a minterm or a maxterm containing {x,y} even if such a minterm or maxterm already exists in MIN or MAX.

Algorithm 2

```
1. MIN := ∅                    /initialize/
   MAX := ∅
2. if Query(V) = 0 then terminate        /check if f is identically 0/
3. S := Findmin(V)
   MIN := MIN ∪ {S}   /put a minterm into MIN/
   OLDSET := ∅     /The following invariants are maintained for OLDSET:
                   For all x in OLDSET, there exists
                    a minterm in MIN containing x, and a maxterm in MAX
                    containing x.  For all pairs of variables {x,y} in
                    OLDSET, there exists either a minterm in MIN
                    or a maxterm in MAX containing {x,y}/
   GROUNDSET := S      /GROUNDSET consists of all variables x such that
                  there exists a minterm in MIN containing x, or a
                  maxterm in MAX containing x (or both)/
   VARSET := S
4. while (VARSET ≠ ∅) do {
     while (VARSET ≠ ∅) do {
        choose some x in VARSET
4a.     if x is contained in a maxterm T of MAX, but not in a minterm of MIN, then {
            S := Findmin((V - T) ∪ {x})   /S is a minterm containing x/
            MIN := MIN ∪ {S}
            VARSET := VARSET ∪ S
            GROUNDSET := GROUNDSET ∪ S
        }
4b.     else if x is contained in a minterm S of MIN, but not in a maxterm of MAX, then {
            T := Findmax((V - S) ∪ {x})   /T is a maxterm containing x/
            MAX := MAX ∪ {T}
            VARSET := VARSET ∪ T
            GROUNDSET := GROUNDSET ∪ T
        }
        VARSET := VARSET - {x}
     }   /upon termination of this loop, all variables in GROUNDSET are contained
           in a minterm of MIN, and in a maxterm of MAX/
     NEWSET := GROUNDSET - OLDSET
     for all pairs {x,y} such that x ∈ NEWSET, y ∈ NEWSET ∪ OLDSET do {
         Let T1 and T2 be maxterms in MAX containing x and y respectively.
         Let S1 and S2 be minterms in MIN containing x and y respectively.
4c.      if Query((V - (T1∪T2)) ∪ {x,y}) = 1 then {
                  /Find a minterm containing x and y/
             S := Findmin((V - (T1∪T2)) ∪ { x,y })
             MIN := MIN ∪ {S}
```

```
            VARSET := VARSET ∪ S
            GROUNDSET := GROUNDSET ∪ S
        }
        else {
                /Find a maxterm containg x and y/
            T := Findmax((V - (S1∪S2)) ∪ {x,y})
            MAX := MAX ∪ {T}
            VARSET := VARSET ∪ T
            GROUNDSET := GROUNDSET ∪ T
        }
    }
    OLDSET := OLDSET ∪ NEWSET;
}
```

Note that the proof of the correctness of Algorithm 1 is not affected by the presence of "redundant" minterms and maxterms, and therefore the proof can be applied to Algorithm 2.

Each variable $x \in V$ contributes at most one minterm or maxterm to the variables in VARSET, and each pair of variable $\{x,y\} \subseteq V$ contributes at most one minterm or maxterm to the variables in VARSET. Each minterm and maxterm contain $O(n)$ variables, and so the total number of variables added to VARSET over the course of the algorithm is $O(n^3)$. Therefore, the while loop in Step 4 is executed at most $O(n^3)$ times. Each variable $x \in V$ is responsible for at most one call to Findmin or Findmax, and each pair of variables $\{x,y\} \subseteq V$ is responsible for at most one call to Findmin or Findmax. Each call to Findmin or Findmax makes $O(n)$ membership queries, and takes time $O(n)$. Therefore, the calls to Findmin and Findmax take a total of $O(n^3)$ time. Theorem 5 follows.

Theorem 5: There exists an algorithm for learning monotone read-once formulas using membership queries that runs in time $O(n^3)$ and makes $O(n^3)$ queries.

6. Extensions and open problems

Using an extension of the algorithm in Section 5, we are able to prove the following:

Theorem 6: There exists an algorithm for learning the number of satisfying assignments of a monotone read-once formula using a membership oracle that runs in time $O(n^3)$ and makes $O(n^3)$ queries.

Proof (Sketch): Apply the learning algorithm of Section 5 to produce a boolean tree corresponding to a read-once formula. This takes time $O(n^3)$. Then, starting from the leaves of the tree, for every node in the tree, compute the number of satisfying

assignments to the formula rooted at that node. This procedure can be carried out in $O(n^2)$ time. □

One open question is whether it is possible to speed up our algorithms by using randomness and parallelism. This question connects in an interesting way to the problem of finding upper and lower bounds on random and parallel algorithms for learning a minterm of a monotone read-once function using membership queries. Lower bounds on random and parallel algorithms for learning a minterm of an arbitrary monotone function using membership queries can be derived from the [KUW] lower bounds for independence system oracles.

7. Summary

The results of this paper, combined with [*PV*] and [*KLPV*], characterize the computational difficulty of learning read-once formulas as follows (Table 1).

Learning Protocol	By Example	Relevant Possibility Oracle	Membership Oracle
Arbitrary Read-Once Formulas	No (if NP ≠ RP)	Yes ($O(n^3)$ time, $O(n^2)$ queries)	No
Monotone Read-Once Formulas	No (if NP ≠ RP)	Yes ($O(n^3)$ time, $O(n^2)$ queries)	Yes ($O(n^3)$ time, $O(n^3)$ queries)

Table 1: Existence of Polynomial Time Learning Algorithms

References

[A] Angluin, D., *Queries and Concept Learning* Machine Learning 2 (1988), pp. 319-342.

[C] Cook, S.A., *A Taxonomy of Problems with Fast Parallel Algorithms*, Information and Control 64 (1985), pp. 2-22.

[GKS] Grigoriev, D.Yu., Karpinski, M., and Singer, M. F., *Fast Parallel Algorithms for Sparse Multivariate Polynomial Interpolation Over Finite Fields*, Research Report No. 8523-C5, University of Bonn (1988), submitted to SIAM J. Comput., 1988.

[KLNSW] Karchmer, M., Linial, N., Newman, I., Saks, M., and Wigderson, A., *Combinatorial Characterization of Read Once Formulae*, Presented at the Joint French-Israeli Binational Symposium on Combinatorics and Algorithms (1988). Submitted to a special issue of Discrete Math.

[KR] Karp, R.M., Ramachandran, V., *A Survey of Parallel Algorithms for Shared Memory Machines*, Research Report No. UCB/CSD 88/407, University of California,

Berkeley (1988). To appear in Handbook of Theoretical Computer Science, North Holland (1989).

[KUW] Karp, R.M., Upfal, E., and Wigderson A., *Are Search and Decision Problems Computationally Equivalent?*, Proc. 17th ACM STOC (1985), pp. 464-475

[KLPV] Kearns, M., Li, M., Pitt, L., Valiant, L.G., *On the learnability of Boolean Formulae*, Proc. 19th ACM STOC (1987), pp. 285-295.

[PV] Pitt, L., and Valiant, L.G., *Computational Limitations on Learning from Examples*, J. ACM 35 (1985), pp. 965-984.

[S] Seinsche, D., *On a Property of the Class of n-Colorable Graphs*, Journal of Combinatorial Theory (B) 16 (1974), pp. 191-193.

[PY] Papadimitriou, C.H. and Yannakakis, M., *The Complexity of Facets (and some Facets of Complexity)*, JCSS 28 (1984), pp. 244-259.

[V] Valiant, L.G., *A Theory of the Learnable*, Communications of the ACM 27 (1984), pp. 1134-1142.

[VV] Valiant, L.G., and Vazirani, V.V., *NP is as Easy as Detecting Unique Solutions*, Proc. 17th ACM STOC (1985), pp. 458-463.