Learning Ordered Binary Decision Diagrams*

Ricard Gavaldà and David Guijarro

Department of Software (LSI)
Universitat Politècnica de Catalunya
Pau Gargallo 5
08028 Barcelona, Spain
{gavalda,guijarro}@lsi.upc.es

Abstract. This note studies the learnability of ordered binary decision diagrams (obdds). We give a polynomial-time algorithm using membership and equivalence queries that finds the minimum obdd for the target respecting a given ordering. We also prove that both types of queries and the restriction to a given ordering are necessary if we want minimality in the output, unless P=NP. If learning has to occur with respect to the optimal variable ordering, polynomial-time learnability implies the approximability of two NP-hard optimization problems: the problem of finding the optimal variable ordering for a given obdd and the Optimal Linear Arrangement problem on graphs.

1 Introduction

The representation of boolean functions as ordered binary decision diagrams (obdds) has received great attention recently. This representation has nice computational properties for fixed variable ordering, such as the existence of a minimum canonical form and efficient algorithms for elementary boolean operations, satisfiability, equivalence testing, and minimization. See [5] for some of the algorithms and a survey of the uses of obdd in fields such as digital system design, combinatorial optimization, and mathematical logic.

A major problem of the obdd representation is that the size of the obdd for a function varies greatly with the variable ordering chosen. The problem of finding an ordering that minimizes the size is usually approached with heuristics. This practice has been often supported by claims that the problem is NP-hard, although a formal proof has been given only very recently ([4], see also [11]).

This note studies the learnability of boolean functions in terms of obdds using membership and equivalence queries. We present an algorithm that, given an ordering, outputs the minimum obdd with that ordering in time polynomial in the size of this obdd (call it n) and the number of variables (call it m). For simplicity, this result is first obtained via a reduction to the problem of learning deterministic finite automata. Using Schapire's algorithm for dfa [10], this yields an algorithm

* This research was supported in part by the ESPRIT Working Group NeuroCOLT (nr. 8556), by the ESPRIT Project ALCOM II (nr. 7141), and by DGICYT (project nr. PB92–0709).

making nm equivalence queries and $O(n^2m^2)$ membership queries. Then we specialize our algorithm to obdds and reduce these bounds to n and $O(n^2m + nm \log m)$, respectively.

Furthermore, we show that we need both types of queries for polynomial-time learning at all, and learning the minimum obdd with respect to *the best* ordering is not possible unless P=NP. This is a consequence of the NP-hardness result in [4] for the problem of computing an optimal ordering for a given obdd. However, it is still open whether this optimization problem admits any polynomial-time approximation algorithm. We observe that learnability of obdds implies that there is some algorithm that finds solutions for this problem within a polynomial of the optimum. The same result holds for the Optimal Linear Arrangement problem [7], whose (non)approximability remains as an important question.

A related result on learnability of restricted branching programs is given by Raghavan and Wilkins in [9]. They show that minimum read-once branching programs are learnable in polynomial time with membership and equivalence queries. But in their context, "read-once" means that each variable is tested once in the whole branching program, while the "read-once" restriction in obdds means once along each path. Hence our results are incomparable. Learning obdds with respect to the best ordering looks like the smallest natural generalization of both their result and our result.

2 Definitions

All our *strings* are defined over the boolean alphabet $\{0,1\}$. By $|x|$ we mean the length of string x. The symbol λ denotes the empty string. To simplify our exposition, lowercase letters late in the alphabet (typically, w, x, y, and z) are used to denote strings, and those early in the alphabet (a, b, and c) to denote bits.

We represent boolean functions by means of *Binary Decision Diagrams,* in short *bdd*s and also called *branching programs.* A bdd is a directed acyclic graph with a single root and two sinks. The two sinks are labelled **acc** and **rej**, and each non-sink node is labelled with a variable v_i. Also, every non-sink node has exactly two outgoing edges, labelled 0 and 1. If D is the bdd and q one of its nodes, by $\delta_D(q,a)$ we mean the endpoint of the edge leaving q and labelled with a.

A bdd on variables v_1, v_2, $\ldots v_m$ computes a boolean function $\{0,1\}^m \to \{0,1\}$ in the natural way: For a string $a_1a_2\cdots a_m$, start at the root and, whenever in a node labelled v_i, follow the edge labelled by bit a_i. The value of the function is 1 if and only if this path ends in the **acc** node. More often, we view a bdd as accepting a language (a subset of $\{0,1\}^m$); for a bdd D, this language is denoted by $L(D)$.

The number of nodes of a bdd D is written as $|D|$ and called its *size.*

Let π be an ordering of $\{1,\ldots,m\}$, that is, a one-to-one function from $\{1,\ldots,m\}$ to itself. For a string $x = a_1a_2\ldots a_m \in \{0,1\}^m$, $\pi(x)$ is the string $a_{\pi(1)}a_{\pi(2)}\cdots a_{\pi(m)}$. This definition is extended to a language $L \subseteq \{0,1\}^m$ by $\pi(L) = \{\ \pi(x) \mid x \in L\ \}$. The identity ordering is denoted by *id.*

For an ordering π and a bdd D, we say that D is a *π-ordered bdd,* or *π–obdd,* if the labels of the nodes along any path in D are consistent with the ordering π. That is, if v_i precedes v_j in a path of D, then $\pi^{-1}(i) < \pi^{-1}(j)$. Note that this definition prevents any path from checking a variable more than once. An *ordered bdd*, or *obdd*, is one that is a π–obdd for *some* π, that is one that is consistent with some ordering.

Our definition of deterministic finite automaton (dfa) is standard. Just to fix notation, we specify an acceptor dfa M by a tuple $(Q, \{0,1\}, q_0, \delta, F)$, where Q, q_0, δ, and F are the set of states, the initial state, the transition function, and the set of final states, respectively.

Our results are proved in Angluin's model of exact learning [1, 2]. We assume that a teacher has a *target* boolean function, or equivalently, a target language $L^\star \subseteq \{0,1\}^m$. The goal of the learning algorithm is to produce an obdd accepting exactly $L^\star$ by asking queries about $L^\star$ to the teacher; it will be crucial whether this obdd must respect a predefined variable ordering or not, and we will make it clear every time.

The learning algorithm can ask the teacher two types of queries:

- *Membership* queries: Given a bit vector $x \in \{0,1\}^m$ the teacher answers YES if $x \in L^\star$ and NO otherwise.
- *Equivalence* queries: Given an obdd D, the teacher answers YES if $L(D) = L^\star$ and a pair (NO,x) where x is the *counterexample,* a string of length m in the symmetric difference of $L^\star$ and $L(D)$.

The running time of the algorithm is taken with respect to the worst-case choice of counterexamples, and is measured as a function of two parameters: m and the size of the smallest obdd accepting $L^\star$ (possibly, the smallest respecting some given ordering π if so specified). The second parameter is usually denoted by n. We say that a learning task is achievable in polynomial time meaning that some learning algorithm completes it in time polynomial in these two parameters for any target boolean function. We also say that we have min-learnability if the algorithm outputs the minimum size obdd.

We assume without loss of generality that learning algorithms know m in advance; otherwise, they ask an initial equivalence query with the empty set and define m to be the length of the counterexample received. Parameter n is initially unknown.

3 Learning Obdds with a Given Ordering

In this section we show that it is possible to learn the minimum π–obdd computing the target function when the ordering π is given.

We do this by reducing this learning problem to the well-studied problem of learning dfa. In fact, we take an *arbitrary* algorithm to learn dfa and show that, with an adequate interface to a teacher for π–obdds, it can be used to learn the minimum π–obdd.

To do this, note that there are two differences between obdds and dfa accepting a subset of $\{0,1\}^m$: one, that the dfa must check the bits of the input string in sequential order while an obdd can use another ordering (though the same in every path); two, the dfa must look at all bits while the obdd can "skip" some if they are not relevant for the result. The following definitions and lemmas deal with both problems.

Definition 1. Let D be any obdd using any ordering. Then $\pi(D)$ is the obdd obtained by replacing, in each node, a label v_i with the label $v_{\pi(i)}$.

Fact 2. *$|\pi(D)| = |D|$ and $L(\pi(D)) = \pi(L(D))$.*

Definition 3. For a dfa M, $D(M)$ is the minimum *id*-obdd accepting $L(M)$. For an *id*-obdd D, $M(D)$ is the minimum dfa accepting $L(D)$. For any language $L \subseteq \{0,1\}^m$, $M(L)$ and $D(L)$ are the minimum dfa and minimum *id*-obdd accepting L, respectively.

Lemma 4. *(i) If $L(M) \subseteq \{0,1\}^m$, then $|D(M)| \leq |M|$; (ii) $|M(D)| \leq 2(m-2)\cdot|D|$.*

Proof. For (i), note that the dfa can be turned into an equivalent *id*-obdd just labelling every node at distance i from the initial state with variable v_{i+1}; no conflicts can appear in this labelling provided that M really accepts a subset of $\{0,1\}^m$.

For (ii), the only problem is that in an *id*-obdd D there may be edges linking a node labelled with v_i to a node labelled with v_j, with $j > i+1$. To transform D into a dfa, it is enough to add a chain of $j-i-1$ states that simply skip over $j-i-1$ bits of input. There are at most $2|D|$ edges, and each edge requires at most $m-2$ intermediate new states. The rej and acc nodes become the unique sink and final states of the dfa. □

(In fact, a finer count that $m \cdot |D|$ is enough in (ii); see the evaluation of the number of membership queries in Theorem 14.)

Fact 5. *$M(D)$ and $D(M)$ can be computed in polynomial time.*

With this notation and facts we can describe the reduction of obdd learning to dfa learning.

Theorem 6. *Assume that there is an algorithm that learns dfa in polynomial time, using $\#equ(n,m)$ equivalence queries and $\#mem(n,m)$ membership queries. Then there is an algorithm that receives an ordering π as input and learns the minimum $\pi-$obdd for the target function in polynomial time, using $\#equ(2mn,m)$ equivalence queries and $\#mem(2mn,m)$ membership queries.*

Proof. Let $L^\star \subseteq \{0,1\}^m$ be the target language and A be the claimed algorithm for learning dfa. The learning algorithm for $\pi-$obdds simulates A learning $\pi^{-1}(L^\star)$ by answering queries as follows:

- Membership query "$x \in \pi^{-1}(L^\star)$?" is replaced with "$\pi(x) \in L^\star$?";
- Equivalence query "$L(M) = \pi^{-1}(L^\star)$?" is replaced with "$L(\pi(D(M))) = L^\star$?" (i.e., transform M into an equivalent *id*-obdd, then permute its node labels following π and ask the resulting obdd as an equivalence query.) By Fact 2, both queries are equivalent. If a counterexample w is obtained, give counterexample $\pi^{-1}(w)$ to A.

Note that all the time needed for these replacements is a small polynomial of the size of their arguments.

When A terminates giving some dfa M, minimize $\pi(D(M))$ and output it.

The correctness of the algorithm is clear using again Fact 2. Let us evaluate, for example, the number of equivalence queries made by this algorithm. Similar arguments work for the number of membership queries and the time complexity.

We simulate A on $\pi^{-1}(L^\star)$, so we make at most $\#equ(|M(\pi^{-1}(L^\star))|, m)$ equivalence queries. Let $D^\star$ be the minimum π−obdd accepting $L^\star$. By Lemma 4, we have that $|M(\pi^{-1}(L^\star))| \leq 2(m-2) \cdot |\pi^{-1}(D^\star)| \leq 2(m-2) \cdot |D^\star|$, hence the claimed bound. □

Several algorithms for exact learning of dfa are known. A first algorithm by Angluin [1] was improved by Schapire [10]. Schapire's algorithm uses at most n equivalence queries and $O(n^2 + n \log m)$ membership queries to learn n-state dfa if the length of the longest counterexample does not exceed m. Plugging this algorithm into Theorem 6 we obtain:

Corollary 7. *There is an algorithm that, given an ordering π, outputs the minimum π−obdd for the target function in polynomial time, using $2nm$ equivalence queries and $O(n^2m^2)$ membership queries, where n is the size of that π−obdd.*

4 Reducing the Number of Queries

In this section we present a modification of Schapire's algorithm that substantially decreases the number of queries used to learn an obdd. In particular, the number of equivalence queries of the new algorithm is bounded above by the π−obdd-size of the target concept, and independent of the number of variables. This is especially important as in many situations equivalence queries are expensive to answer. For example, in the standard transformation of an equivalence-query learner into a PAC-learner [1, 2], the sample size used by the latter is quadratic in the number of queries of the former.

We describe the algorithm assuming that the required order π is the identity. Other orders are handled by bit permutations as explained in the proof of Theorem 6.

We need some definitions and facts from Schapire's algorithm, and assume some familiarity with it. We repeat them here but we direct the reader to [1, 10] for full details.

Definition 8. An *observation table* consistent with $L^\star \subseteq \{0,1\}^\star$ is a triple (S, E, T) where $S, E \subseteq \{0,1\}^\star$ and T is a function $\{0,1\}^\star \times \{0,1\}^\star \mapsto \{0,1\}$, such that for every $x \in S \cdot (\lambda + 0 + 1)$ and every $e \in E$, $T(x, e) = 1$ iff $xe \in L^\star$.

For every string x, we define a function $row(x)$ with domain E by $row(x)(e) = 1$ if $xe \in L^\star$, 0 otherwise. We will use $row(x)$ mostly (but not only) for $x \in S$.

An observation table is *closed* if for every $w \in S\{0,1\}$ there is some $u \in S$ such that $row(w) = row(u)$.

Schapire's algorithm maintains an observation table (S, E, T) consistent with the target language. Whenever the table is closed, a dfa $M(T)$ is built out of the table and asked as an equivalence query. Each element in S is used as a representative of the state it reaches in the target dfa.

Fact 9. *The observation table maintained by Schapire's algorithm satisfies the following:*

1. *for every different w and w' in S, $row(w) \neq row(w')$.*
2. *$x \in S$ implies $|x| \leq m$.*

The difference in size between the minimum dfa and the minimum obdd for the target is in the number of states that are introduced just to skip irrelevant input bits. Inside the observation table, these states translate into *blind* rows.

Definition 10. Given an observation table (S, E, T), we say that a function $row(w)$ $(w \in S)$ is *blind* if $row(w0) = row(w1)$.

Fact 11. *Suppose that Schapire's algorithm has at some moment an observation table $O = (S, E, T)$ and that at a later moment it has table $O' = (S', E', T')$. Let row and row' be the respective row functions. Then:*

1. *for every $x, y \in S$, $row(x) \neq row(y)$ implies $row'(x) \neq row'(y)$ (distinct rows never become equal again).*
2. *for every $x \in S$ if $row(x)$ is non-blind then $row'(x)$ is also non-blind.*

This follows immediately from the fact that Schapire's algorithm never deletes any entry from its observation table.

The reduction in the number of equivalence queries is based on checking the following property, that we call *self-consistency.* The intuitive meaning of this property is the following. Suppose that a string $w \in S$ leads to a non-blind state $row(w)$ in $M(T)$. If we append more and more bits to the end of w, we may visit some blind states $row(x_1)$, $row(x_2)$, ... of $M(T)$ until another non-blind state $row(w')$ is reached again. In principle, w and every x_i may be completely different, but we are guessing that after appending i bits to the end of w we really end up in the same state of the target dfa as x_i. Self-consistency states that this guess is not obviously wrong: it is not disproven by just trying a string of i zeros and the current set of experiments E.

Definition 12. Let $O = (S, E, T)$ be a closed observation table. We say that O is *self-consistent* if the following occurs for every $w \in S$ with a non-blind row and every $a \in \{0, 1\}$: Assume that in the obdd obtained from O we have $\delta(w, a) = w'$, with w' a non-blind row in T. Note that all states between $row(w)$ and $row(w')$ are blind. Then, for every i with $1 \leq i < |w'| - |wa|$:

1. let x be the state where the evaluations of strings $wa0^i0$ and $wa0^i1$ end; then $row(wa0^i0) = row(wa0^i1) = row(x)$, and
2. for every $\forall b, b' \in \{0, 1\}$ $row(wa0^ibb') = row(xb')$.

Lemma 13. *If a closed observation table O is consistent with target concept $L^\star$ but not self-consistent, then we can find a counterexample.*

Proof. If self-consistency fails then there is an i such that at least one of the following two cases is true:

1. $row(wa0^ib) \neq row(x)$ for some $b \in \{0,1\}$: $wa0^ib$ and x reach the same state in the dfa associated to the observation table, but there is some $e \in E$ such that exactly one of $wa0^ibe$ and xe is in L^*. Hence, $wa0^ibe$ is a counterexample.
2. $row(wa0^ib) \neq row(x)$ for some $b \in \{0,1\}$: by the same argument, $wa0^ib$ is a counterexample for the current obdd.
3. $row(wa0^ibb') \neq row(xb')$ for some $b, b' \in \{0,1\}$: then, again with the same argument, $wa0^ibb'$ is a counterexample.

□

We modify Schapire's algorithm filtering unnecessary equivalence queries. More precisely, whenever Schapire's algorithm poses an equivalence query, we test whether the observation table is self-consistent. If this is not the case, by Lemma 13 we can find a counterexample using a membership query instead. This way we keep the number of equivalence queries bounded above by obdd-size. The new bound on membership queries follows from the fact that obdds accept fixed-length languages, and hence a large fraction of membership queries can be answered NO right away.

Theorem 14. *There is an algorithm that, given the ordering π, outputs the minimum $\pi-$obdd for the target function in polynomial time, using n equivalence queries and $O(n^2m+nm\log m)$ membership queries, where n is the size of that $\pi-$obdd and m is the number of variables.*

Proof. Run Schapire's algorithm. Membership queries are passed to the teacher if their length is m, others are answered NO.

Whenever Schapire's algorithm asks an equivalence query with a dfa that accepts a string not in $\{0,1\}^m$, we supply a NO answer with that string as a counterexample. So we let pass only the equivalence queries accepting subsets of $\{0,1\}^m$. Note that dfa accepting these languages are dag's where all paths from the root to a given state have the same length (except for the sink); we call this length the *level* of the state.

To solve these equivalence queries, we first check self-consistency of the hypothesis. If self-consistency fails, we get a counterexample by Lemma 13 and skip the equivalence query. Otherwise, the equivalence query is passed to the teacher.

Correctness of this algorithm is clear since Schapire's algorithm is correct. Let us bound first the number of equivalence queries it makes.

Consider two consecutive equivalence queries that we pass to the teacher. Let O and O' be their respective observation tables and let nr and nr' be the corresponding number of non-blind rows. We argue first that $nr < nr'$; then, using that Schapire's algorithm never surpasses the minimum number of states, the number of equivalence queries is bounded by the number of non-blind states that of the minimum target dfa, which, in turn, coincides with the number of states of the minimum $\pi-$obdd.

Suppose for the sake of contradiction that $nr \geq nr'$, using Fact 11 we know that $nr > nr'$ is impossible (a non-blind row remains non-blind forever). This leaves $nr = nr'$ as the only case.

The only possibility now is that, since the obdds associated to O and O' will have the same number of states, the transition function changes. Let u, u' and wa be the strings involved in one of these transition function changes, i.e., w, u, and

u' are in S, $row(w)$, $row(u)$ and $row(u')$ are non-blind, $row(u) \neq row(u')$, and wa leads to u in O but to u' in O'.

There are three cases:

1. $|u| > |u'|$. For $i = |u'| - |wa|$, self-consistency of O' implies that $row'(wa0^i) = row'(u')$ and this implies that $row(wa0^i) = row(u')$, but $i < |u| - |wa|$, so self-consistency implies that $row(wa0^i0) = row(wa0^i1)$. But if $row(u')$ is non-blind then $|u'| = m$ and $|u| > m$ which contradicts Fact 9.
2. $|u| = |u'|$. A similar argumentation yields $row(u) = row(u')$ which is a contradiction with the assumption that u and u' were involved in a transition function change.
3. $|u| < |u'|$. For $i = |u| - |wa|$, self-consistency of O together with the fact that $row(u)$ is non-blind, implies that $row(wa0^i0) \neq row(wa0^i1)$. But this contradicts self-consistency in O', namely that $row'(wa0^i0) = row'(wa0^i1)$.

Now we have proved $nr < nr'$ for any consecutive closed and self-consistent observation tables O and O', and hence that the number of equivalence queries does not exceed n.

For the membership queries: Schapire's algorithm makes membership queries at two moments: after a counterexample is found (either with an equivalence query or the use of Lemma 13), in order to find an experiment that distinguishes two previously equal rows; and all other membership queries, used to fill new entries (rows or columns) of the observation table. Additionally, we make membership queries to test self-consistency.

In our case, the number of the first type of queries is $O(nm \log m)$, because our target dfa has at most $2mn$ states, and finding the right experiment for the counterexample costs only $\log m$ membership queries, as described in [10].

For the other kind of membership queries: Define S_i and E_i to be the subsets of of the final S and E containing only strings of length i. Note that a string is introduced in E_{m-i} only to distinguish two rows indexed by strings of length i that looked equal before; therefore $|E_{m-i}| \leq |S_i|$. Only strings of length m are queried, and for every $w \in S$ there are rows indexed by w, $w0$, and $w1$. So the number of queries used to fill the table is at most

$$3 \cdot \sum_{i=0}^{m} |S_i| \cdot |E_{m-i}| \leq 3 \cdot \sum_{i=0}^{m} |S_i|^2.$$

To bound $|S_i|$, observe that there are two types of states in level i of the target dfa: some non-blind states corresponding to nodes in the target obdd, and some blind states corresponding to edges that cross level i. Let a and b be the number of each. Now, to have a nodes at level i there must be at least $a - 1$ distinct nodes in previous levels; for b edges to cross this level, there must be at least b distinct nodes in previous levels. Therefore $a + \max\{a - 1, b\} \leq n$, which implies $|S_i| = a + b \leq n$.

Hence, the number of membership queries used to fill the observation table is at most $3 \cdot \sum_{i=0}^{m} |S_i|^2 \leq 3(m+1)n^2$.

Finally, to bound the number of queries used to check self-consistency, let R_i be the set of non-blind rows at level i of the final hypothesis. For each $w \in R_i$, we make at most $4m|E_{m-i}|$ queries, so the number of queries for this purpose is

$$\sum_{i=0}^{m} |R_i| \cdot 4m|E_{m-i}| \leq 4m \sum_{i=0}^{m} |R_i| \cdot |S_i| \leq 4m \cdot n \cdot \sum_{i=0}^{m} |R_i| = 4n^2 m.$$

This concludes the proof. □

5 Negative Results

Using adversary arguments, one can show that both types of queries, membership and equivalence, are necessary for learning at all.

Theorem 15. *Obdds are not learnable with membership queries alone or with equivalence queries alone, even with respect to any fixed ordering.*

Proof. For membership queries alone, the standard adversary argument using the class of all singleton sets shows that exponentially many membership queries are needed.

For equivalence queries alone, we observe that a learning algorithm for obdds and any fixed ordering yields an algorithm that learns dfa accepting fixed-length languages, and recall that no such algorithm exists as shown by Angluin [3]. Strictly speaking, it is not enough to appeal to the proof in [3]: we should show that the class of witness dfa provided by Angluin is still hard to learn under any permutation of input bits.

Instead, we reduce the problem as follows: to learn a target $L^\star \subseteq \{0,1\}^m$ run the learner for π–obdds over $\pi^{-1}(L)$, and transform its output first into an $id - obdd$ (by renaming variables), then into a dfa. This yields the minimum dfa in polynomial time. □

Furthermore, if no order is given to the learner and we want it to find the best ordering, learning is computationally hard.

Theorem 16. *There is no polynomial-time algorithm that learns minimum-size obdds with membership and equivalence queries, unless $P = NP$.*

Proof. Assume that there is such a learning algorithm. Then, given an obdd, we can find in polynomial time the variable ordering that minimizes its size by running the learning algorithm; the theorem follows as this problem is NP-complete [4].

The process is as follows: given an obdd O, run the learning algorithm; membership queries are solved by evaluating them over O; equivalence queries are be solved in polynomial time using the algorithm of Fortune, Hopcroft, and Schmidt [6], which works even if the two obdds use different variable orderings. □

With the same argument we can show: suppose we have a polynomial-time learning algorithm that does not necessarily output the optimum obdd, but is guaranteed to output a polynomial approximation to it (this is the standard definition of learning). Then, we can use it to build an approximation algorithm for the obdd minimization problem whose performance ratio is polynomial.

Actually, the implication is true for another NP-hard optimization problem called Optimal Linear Arrangement or OLA. This result is essentially the observation that the reduction from OLA to obdd-minimization in [4] is approximation-preserving.

Definition 17. [7] The *Optimal Linear Arrangement* problem (OLA) is defined as follows:

INSTANCE: Undirected graph $G = (V, E)$, positive integer K.

QUESTION: Is there a one-to-one function $f : V \mapsto \{1, 2, \ldots, |V|\}$ (a linear arrangement) such that $cost_G(f) = \sum_{(u,v)\in E} |f(u) - f(v)| \leq K$?

Theorem 18. *If boolean functions are learnable in terms of obdds (with respect to the best ordering, but not necessarily in minimal form) then OLA can be approximated within a polynomial.*

Proof. We show that the reduction from OLA to finding optimal variable orderings in obdds is approximation-preserving up to a polynomial.

Given an undirected graph $G = (V, E)$, let n and m be $|V|$ and $|E|$ respectively. We deal w.l.o.g. with the case where the graph is connected and hence $m \geq n - 1$.

The reduction in [4] produces from G an obdd o with the following properties:

1. o has more than n variables, but the optimum ordering always belongs to a set of easily described orderings with $n!$ elements, called "blockwise orderings" in [4].
2. For every variable ordering that is not blockwise we can easily find a blockwise one that gives equal or smaller size.
3. There is a bijection between the set of possible arrangements of G and blockwise orderings of the variables in o. Hence, every possible arrangement $f : V \mapsto \{1, 2, \ldots, |V|\}$ of G gives an obdd o_f equivalent to o.
4. Given G and f, it is easy to compute o_f and vice-versa.
5. There is a constant c such that, for every f, the following is satisfied:
$$|o_f| = cn^2m^2 + cost_G(f).$$
6. Therefore, if $f^\star$ is the cheapest arrangement for G and $o^\star$ the smallest obdd equivalent to o, we have
$$|o^\star| = |o_{f^\star}| = cn^2m^2 + cost_G(f^\star).$$

Now assume that there is an algorithm that minimizes obdds up to some polynomial p. Run this algorithm on the input graph G and obtain an obdd o'. By property 2 we can assume that o' is blockwise-ordered, that is, it is $o_{f'}$ for an arrangement f'. Compute f' and output it as an approximation to the optimal linear arrangement.

Indeed, using properties 5 and 6 we have

$$cost_G(f') = |o'| - cn^2m^2 \leq p(|o^\star|) - cn^2m^2 = p(cost_G(f^\star) + cn^2m^2) - cn^2m^2.$$

Since any linear arrangement has cost at least $m \geq n - 1$, this is within some polynomial of $cost_G(f^\star)$. □

Acknowledgments

We thank Christoph Meinel for pointing us to the work [4], Ingo Wegener for providing us with a copy, Hans Ulrich Simon and an anonymous referee for very helpful comments, and Josep Díaz for posing the question of learning obdds.

References

1. D. Angluin: "Learning regular sets from queries and counterexamples". *Information and Computation* **75** (1987), 87–106.
2. D. Angluin: "Queries and concept learning". *Machine Learning* **2** (1988), 319–342.
3. D. Angluin: "Negative results for equivalence queries" *Machine Learning* **5** (1990), 121–150.
4. B. Bollig and I. Wegener: *Improving the variable ordering of OBDDs is NP-complete.* Technical Report # 542, Universität Dortmund (1994).
5. R.E. Bryant: "Symbolic boolean manipulation with ordered binary decision diagrams". *ACM Computing Surveys* **24** (1992), 293–318.
6. S. Fortune, J. Hopcroft, and E. Schmidt: "The complexity of equivalence and containment for free single variable program schemes". *Proc. 5th Intl. Colloquium on Automata, Languages, and Programming.* Springer-Verlag Lecture Notes in Computer Science 62 (1978), 227–240.
7. M. Garey and D. Johnson: *Computers and intractability: a guide to the theory of NP-completeness.* Freeman 1979.
8. J. Gergov and C. Meinel: "On the complexity of analysis and manipulation of Boolean functions in terms of decision graphs". *Information Processing Letters* **50** (1994), 317-322.
9. V. Raghavan and D. Wilkins: "Learning μ-branching programs with queries". *Proc. 6th COLT* (1993), 27–36.
10. R.E. Schapire: *The Design and Analysis of Efficient Learning Algorithms.* MIT Press, 1992.
11. S. Tani, K. Hamaguchi, and S. Yajima: "The complexity of the optimal variable ordering problems for shared binary decision diagrams". *Proc. 4th Intl. Symposium ISAAC'93.* Springer Verlag Lecture Notes in Computer Science 762 (1993), 389–398.