Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information and Computation 201 (2005) 178–198
Information and Computation
www.elsevier.com/locate/ic

An efficient query learning algorithm for ordered binary decision diagrams ☆

Atsuyoshi Nakamura [1]

Division of Computer Science, Graduate School of Information Science and Technology, Hokkaido University, Sapporo 060-0814, Japan

Received 7 July 2003; revised 24 September 2004
Available online 28 July 2005

Abstract

In this paper, we propose a new algorithm that exactly learns ordered binary decision diagrams (OBDDs) with a given variable ordering via equivalence and membership queries. Our algorithm uses at most n equivalence queries and at most $2n(\lceil \log_2 m \rceil + 3n)$ membership queries, where n is the number of nodes in the target-reduced OBDD and m is the number of variables. The upper bound on the number of membership queries is smaller by a factor of $O(m)$ compared with that for the previous best known algorithm proposed by R. Gavaldà and D. Guijarro [Learning Ordered Binary Decision Diagrams, Proceedings of the 6th International Workshop on Algorithmic Learning Theory, 1995, pp. 228–238].

Keywords: Ordered binary decision diagram; Branching program; Query learning; Exact learning; DFA

1. Introduction

An ordered binary decision diagram (OBDD) is known to be a useful representation of a boolean function because of its easiness for performing boolean manipulations [2]. OBDDs are now being

☆ A preliminary version of this paper appeared in the Proceedings of the Eighth International Workshop on Algorithmic Learning Theory, October 1997.

E-mail address: atsu@main.ist.hokudai.ac.jp.

[1] Most of the work presented herein was performed while the author was with NEC Corporation.

0890-5401/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2005.05.003

extensively studied in several fields, including digital-system design [11], combinatorial optimization [9], mathematical logic [6], and artificial intelligence [12]. There have also been many studies on OBDDs in the field of learning theory: query learnability of μ-branching programs [15], query learnability of OBDDs with a given variable ordering [8], PAC learnability of bounded-width branching programs [7,5], and query learnability of bounded-width branching programs [3,13]. Here, we consider query learnability of OBDDs with a given ordering, which was studied by Gavaldà and Guijarro [8].

The algorithm proposed by Gavaldá and Guijarro learns OBDDs with a given ordering via equivalence and membership queries. Their algorithm is derived from a well-known algorithm for learning deterministic finite automata (DFAs) proposed by Angluin [1] and its improved version developed by Rivest and Schapire [14]. However, the DFA representation of an OBDD requires many states needed to skip irrelevant bits. The algorithm proposed by Gavaldá and Guijarro saves equivalence queries needed to find those states but still identifies all of those states, and because of this uses a number of membership queries that depends *linearly* on the number of variables in the worst case (see Section 4.1). In this paper, we propose an algorithm that does not identify any of those states explicitly. Our algorithm learns an arbitrary OBDD with a given variable ordering using at most n equivalence queries and at most $O(n(\log m + n))$ membership queries, where n is the number of nodes in the target-reduced OBDD and m is the number of variables. The upper bound on the number of membership queries for our algorithm is smaller by a factor of m than that for Gavaldà and Guijarro's algorithm, $O(mn(\log m + n))$.

Our algorithm can be regarded as an extension of the algorithm Learn-Automaton [10], which learns DFAs using *classification trees* instead of *observation tables* [1,14]. Our classification trees, however, have special internal nodes called *twin-test nodes* and a special leaf node labeled μ, which are necessary for testing that a given string reaches one of the states needed to skip irrelevant bits in the DFA representation of a target OBDD without revealing which state it is.

There is an interesting application studied by Birkendorf and Simon [4], where our algorithm may be useful. They proposed a technique of using query learning algorithms in a heuristic strategy for a class of NP-hard combinatorial optimization problems. Their technique can be applied to the problem of minimizing an OBDD relative to a given domain. Our algorithm provides an efficient implementation of their technique for this minimization problem.

The rest of this paper is organized as follows. First, we describe the definitions of OBDDs and the query learning model in Section 2. In Section 3, the algorithm Learn-Automaton, which learns DFAs and basically uses the same data structure as that used by our algorithm for OBDDs, is described. In Section 4, it is first shown that a direct adaptation of the modified Learn-Automaton for OBDDs needs at least $mn^2/32$ membership queries in the worst case, and then the data structure of our algorithm is explained. The new learning algorithm is presented in Section 5. A proof of its correctness and an analysis of its worst-case query complexity are given in Section 6. Some experimental results are presented in Section 7. This paper is concluded in Section 8.

2. Preliminaries

A binary decision diagram (BDD), also called a branching program, is a directed acyclic graph with one root node and two sink nodes, one labeled 0 and the other labeled 1. Each internal node

is labeled with a boolean variable and has two outgoing edges, one labeled 0 and the other labeled 1. By an 'ordered' BDD (OBDD), it is meant that labeled variable sequences for any paths from the root to one of the sink nodes must be consistent with a certain variable ordering π. An OBDD represents a boolean function according to the following interpretation: the value for a given assignment $x_1 = a_1, \ldots, x_m = a_m$ is obtained by starting from its root node, selecting the a_i-labeled outgoing edge at a node labeled x_i and regarding the label of the sink node reached finally as the value of the function. It is well known that every boolean function can be uniquely represented by an OBDD in the reduced (minimum) form for a given variable ordering [2].

The learning model we employ is the *query learning model* proposed by Angluin [1]. The goal of a learning algorithm in this model is to identify a target function f using *equivalence queries* and *membership queries*. An *equivalence query* asks if the unknown target function f is equivalent to a hypothesis h; if so, 'YES' is returned as a reply, and if not, a counterexample e ($f(e) \neq h(e)$) is returned as a reply. A *membership query* asks for the value $f(a)$ of the target function f for an assignment a.

3. Kearns and Vazirani's Algorithm for DFAs

The basic structure of our algorithm is the same as that of the DFA-learning algorithm *Learn-Automaton* [10]. Learn-Automaton makes use of a *classification tree* instead of an *observation table* used by Angluin's algorithm [1]. A classification tree is preferable to an observation table from the viewpoint that using a classification tree a state reached by a string is determined only by going along a path depending on the answers for a set of membership queries while searching the row of the same values is necessary for an observation table. In this paper, an efficient OBDD-learning algorithm that uses extended classification trees is presented. In this section, the algorithm Learn-Automaton and its modifications needed for efficiency are explained to help the reader understand our algorithm, which is based on the modified version of Learn-Automaton.

For any state of any DFA, there is a string that leads to that state starting from the initial state. Kearns and Vazirani called such a string an *access string*. For any pair of access strings s and s' of two distinct states, there is a string d such that sd reaches an accepting state and $s'd$ reaches a rejecting state, or vice versa. Kearns and Vazirani called such a string a *distinguishing string* for s and s'. A *classification tree* has internal nodes labeled with a distinguishing string and leaf nodes labeled with an access string of a distinct state. Each internal node has two outgoing edges, one labeled 0 (representing "reject") and the other labeled 1 (representing "accept"). Using this tree, any string s can be classified into one of the states as follows. Start from the root node, ask a membership query for the string sd at the current internal node labeled d, and follow the edge labeled with the answer for that membership query, that is, follow the 0-labeled edge if the answer is reject, and follow the 1-labeled edge otherwise. Repeat the same procedure until one of the leaf nodes, whose label s' is an access string of a state, is reached. This means that, starting from the initial state, s finally reaches a state with the access string s' in the DFA represented by the classification tree. Using a classification tree, the DFA represented by the tree can be constructed as follows. Create one state for each leaf node of the tree. For each state s, classify $s0$ and $s1$ into s_0 and s_1, respectively, by the tree using membership queries, and add directed edges (s, s_0) labeled 0 and (s, s_1) labeled 1, which represent a transition function. Note that we abuse notation and use access strings as states

they reach. Learn-Automaton updates a classification tree by repeating the following procedure: construct a hypothesis DFA, obtain a counterexample by asking an equivalence query for it, find an access string of a new state and update the classification tree.

3.1. A few modifications for efficiency

For efficiency, the following modifications are necessary for Learn-Automaton:

(1) Maintain the current hypothesis and update it instead of constructing every time from the beginning.
(2) Apply the binary search method of Rivest and Schapire to finding an access string of new state from a counterexample.

Since the states are represented by their access strings, it is natural that the states of the maintained hypothesis DFA should be labeled with their access strings. We call such a DFA *DFA with access strings*.

The modified version of Learn-Automaton works as follows. Assume that the counterexample 011 is returned to the equivalence query for the hypothesis represented by the first DFA in Fig. 1. The algorithm becomes aware that the string 01 reaches a new state instead of the state labeled 1 and that the distinguishing string 1 separates the two states. Then, the algorithm replaces the leaf node labeled 1 in the first classification tree in Fig. 1 with an internal node labeled 1 having two child nodes, one labeled 1 and the other labeled 01. The node labeled 01 is added to the first DFA with access strings in Fig. 1. Membership queries for the new dis-

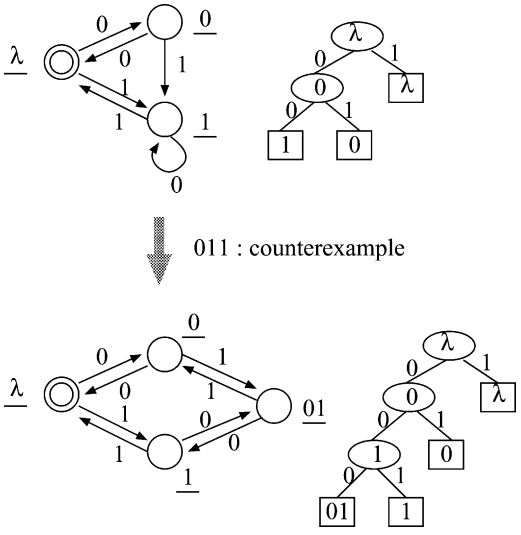


Fig. 1. DFAs with access strings and classification trees maintained by the modified version of Learn-Automaton: the underlined strings in the DFAs are access strings that serve as the unique IDs of each state and are used as leaf labels in the corresponding classification tree.

tinguishing string 1 are asked for all edges directed to the state 1 and the answer determines whether the edge direction should be changed to the state 01 or not. The directions of two outgoing edges from the state 01 are decided by the classification tree and the answers for the membership queries asked at its internal nodes.

It can be easily shown that in the worst case the modified version of Learn-Automaton needs n equivalence queries and at most $O(n(\log m + n))$ membership queries, where n is the number of states in the target DFA and m is the length of the longest counterexample the algorithm received. These numbers are the same as those needed by Rivest and Schapire's improved version of Angluin's algorithm [14].

4. Data structure for OBDDs

An OBDD represents a function f from $\{0,1\}^m$ to $\{0,1\}$, as explained in Section 2, where m is the number of variables. This can be seen as an accepter of the language $\{a_1 \ldots a_m : f(a_1, \ldots, a_m) = 1\}$. Since there is a straightforward conversion from an OBDD with an ordering π to a DFA that accepts the same language, the language class accepted by OBDDs with an ordering π is a subclass of the class of the regular sets. Thus, any learning algorithms for DFAs can be used to learn the language class representable by OBDDs. However, there are many functions whose DFA representation must have redundancy compared to the OBDD representation: a function representable by an OBDD having n nodes may need mn nodes even in its most compact DFA representation. Gavaldá and Guijarro considered that this redundancy comes from the states needed to skip irrelevant input bits and tried to cut down the cost needed for learning those states from Rivest and Schapire's algorithm. Although their modification of the algorithm succeeded in reducing the number of equivalence queries, it is still a kind of *direct adaptations* of DFA-learning algorithms, which identify all of the states needed to skip irrelevant input bits. In this section, after showing a barrier that direct adaptations cannot break through, a data structure used to break through the barrier is described.

4.1. Direct adaptations

From the fact that the number of states is at most mn, the number of membership queries needed by Rivest and Schapire's algorithm and modified Learn-Automaton is $O(m^2n^2)$. Using the fact that the answers are 0 for all of the membership queries for strings whose length is not m, the upper bound on the number of membership queries can be reduced to $O(mn(\log m + n))$ [8]. However, obtaining an upper bound less than linear dependency on m appears to be difficult if methods that identify all of the states needed to skip irrelevant input bits are used.

Theorem 1. *For any positive multiple n of* 4*, there exists a natural number* m_0 *such that for any number* $m \geqslant m_0$ *of variables, there exists an OBDD with n nodes that requires at least* $mn^2/32$ *different membership queries in the worst case in order to be identified by the modified version of Learn-Automaton presented in Section 3.*

Proof. See Appendix Appendix B. □

Remark 2. Rivest and Schapire's algorithm also needs $mn^2/32$ membership queries in the case considered in the proof of Theorem 1 (Appendix Appendix B) because the entries in the observation table corresponding to T'_k for $k = n/2, n/2+1, \ldots, m-n/2$ must be filled. Note that the order of finding the states by their algorithm is almost the same as the order of finding the states by the modified Learn-Automaton, and closedness checks of an observation table can find only one state with access string 1^m.

4.2. Data structure used in our Algorithm

To break through the barrier of linear dependency on the number of variables with respect to the number of membership queries, we introduce special internal nodes called *twin-test nodes* and a special leaf node labeled μ into classification trees. Using these nodes, we can test that a given string reaches one of the states needed to skip irrelevant bits in the DFA representation of a target OBDD without revealing which state it is. Thus, it is not necessary to identify all of those states, and as a result the number of membership queries is reduced.

In this section, the data structure used in our algorithm, *OBDDs with access strings* and a modified version of classification trees, is explained.

Assume that a variable ordering is $\pi: x_1 < x_2 < \cdots < x_m$. An *OBDD with access strings*, hereafter abbreviated as *OBDDAS*, is different from a normal OBDD in the following points:

- Its root node is always a node labeled x_1, which may be a dummy node having only one outgoing edge.
- The Label of an edge is a binary string. Its length is $|i-j|$ for the edge between nodes labeled x_i and x_j, and $m+1-i$ for an edge that goes out from the node labeled x_i to a sink node. The first bits of two edges going out from the same node must be different.
- Each node has an *access string*, which is a concatenation of the label strings of the edges on one selected path from the root node to that node.

An example of an OBDDAS is shown in the leftmost part of Fig. 2. The root node of this OBDDAS is a dummy, and the access string of each node is the underlined string written beside the

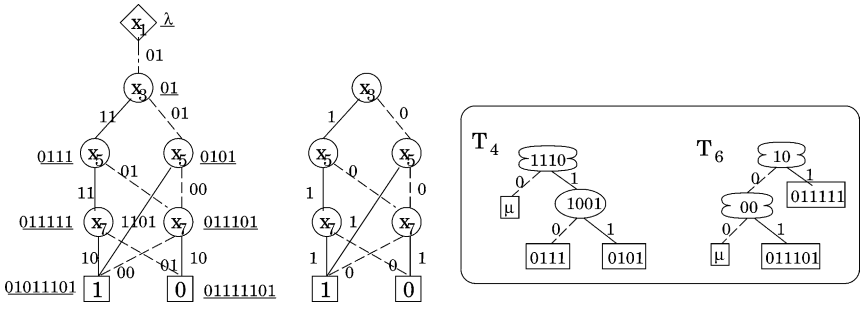


Fig. 2. From the left, an OBDDAS S, the OBDD $\mathcal{D}(S)$ and classification trees T_4 and T_6: T_4 has one twin-test node and one single-test node, and T_6 has two twin-test nodes.

node. Here, λ denotes a null string. From an OBDDAS S, we can obtain a normal OBDD, which is denoted by $\mathcal{D}(S)$, by the following operations:

- Remove the dummy root node and its outgoing edge.
- Remove the access strings of all nodes.
- Remove all bits except the first bit from the label strings of all edges.

The OBDD $\mathcal{D}(S)$ transformed from the OBDDAS S by the above operations is shown in the middle part of Fig. 2.

A classification tree decides which node in an OBDDAS a given string will reach based on the answers for the membership queries asked at each node of the tree. The OBDD version is different from the DFA version in the following respects.

- There is one classification tree T_i for each different string length i for $1 \leqslant i \leqslant m$.
- The trees T_i may have one leaf node labeled μ, which means that a given string does not reach an node at length i.
- The trees have two types of internal nodes: *single-test* nodes and *twin-test* nodes. (The trees for DFAs have single-test nodes only.)

The difference between single-test nodes and twin-test nodes is as follows: at the node labeled r, only an *experiment* for r is conducted at single-test nodes, but an *experiment* for $\dot{r}, \dot{r}$ being the string obtained by flipping the first bit of r, is also conducted at twin-test nodes.

Here, for a given string s, an *experiment* for r means asking a membership query for the assignment that corresponds to string sr, the concatenation of string s and r.

The three differences stated above are briefly explained here. Since strings of different lengths cannot reach the same node in an OBDDAS, a classification tree can be decomposed into trees for each length. In a DFA with access strings, every string reaches one of the states, but in an OBDDAS, some strings do not reach any node. Thus, a leaf node is needed for such a case. Twin-test nodes play an important role in deciding whether reachable nodes exist or not in an OBDDAS.

A precise definition of classification trees for OBDDs is given here. Let S be an OBDDAS and S-nodes$_i(S)$ be the set of access strings possessed by the non-dummy nodes in S whose length is i. Let S-nodes$(S) = \bigcup_{i=0}^{m}$ S-nodes$_i(S)$. We define a *classification tree* T_i as follows. For $1 \leqslant i < m$, the tree T_i is composed of single-test and twin-test internal nodes and of $|\text{S-nodes}_i(S)| + 1$ leaf nodes. The leaf nodes are labeled with different strings in S-nodes$_i(S) \cup \{\mu\}$. Each internal node is labeled with a string of length $m - i$ and has two child nodes connected by an edge labeled 0 and an edge labeled 1. As for T_m, the tree can be composed of a single-test node and two leaf nodes labeled with different strings in S-nodes$_m(S)$. By this T_i, a string $a \in \{0,1\}^i$ is classified into one of its leaf labels, which is denoted by $T_i(a)$, as follows. Start from the root node of T_i. If a has reached a twin-test node labeled $r \in \{0,1\}^{m-i}$, select the outgoing edge labeled 1 if[2] $D(ar) = 1$ and $D(a\dot{r}) = 0$, where D denotes a target OBDD. Otherwise, select the outgoing edge labeled 0. If a has reached a single-test

[2] We abuse notation and use representations as the functions they represent. For an OBDDAS S, S is also used as the function that $\mathcal{D}(S)$ represents.

node labeled $r \in \{0,1\}^{m-i}$, select the outgoing edge labeled $D(ar)$. Classify a into the label of the leaf reached finally. For example, T_4 in Fig. 2 is the classification tree for S-nodes$_4(S)$ of the leftmost OBDDAS S. Its root node is a twin-test node and the other internal node is a single-test node.

Beginning with a simple hypothesis, our algorithm obtains a counterexample by an equivalence query, updates its current hypothesis while satisfying certain conditions, and repeats this process until 'YES' is returned for an equivalence query. Before describing the conditions, the notation used will be explained. Let D denote a target OBDD. The set nodes(D) is the set of strings $a_1a_2\cdots a_k$ such that $k = m$ or the assignment with $x_1 = a_1, \ldots, x_k = a_k$ leads to a node labeled x_{k+1} in D. If D is reduced, $v \in$ nodes(D) if and only if $|v| = m$ or there exists a string r of length $m - |v|$ such that $D(vr) \neq D(v\dot{r})$. For $v_1, v_2 \in$ nodes(D), an equivalence relation '$\overset{D}{=}$' is defined as follows:

$$v_1 \overset{D}{=} v_2 \overset{def}{\Leftrightarrow} v_1 \text{ and } v_2 \text{ lead to the same node in } D.$$

Note that for any classification tree T_i,

$$\text{Pl. } \forall v_1, \forall v_2 \in \text{nodes}(D) \text{ with } |v_1| = |v_2| = i \ \ [v_1 \overset{D}{=} v_2 \Rightarrow T_i(v_1) = T_i(v_2)]$$

holds. For an OBDDAS S, S-edges(S) is the set of edges (u, v) in S, where u and v are the access strings of the end nodes of an edge. For an edge (u, v), $l(u, v)$ is the label of the edge. For a string a, pre(a, i) is the prefix string of a with length i.

The conditions which all the intermediate OBDDASs and classification trees of our algorithm satisfy are C1, C2, and C3 given in the following lemma.

Lemma 3. *For a reduced OBDD D, assume that an OBDDAS S and classification trees T_i for all $i = 1, \ldots, m$ satisfy the following conditions C1, C2, and C3.*

C1. (1) S-nodes(S) $\subseteq$ nodes(D),
(2) $\forall v \in$ S-nodes$_m(S)[S(v) = D(v)]$, and
(3) $\forall v_1, \forall v_2 \in$ S-nodes$(S)[v_1 \neq v_2 \Rightarrow v_1 \overset{D}{\neq} v_2]$.
C2. (1) $\forall v \in$ S-nodes$(S)[T_{|v|}(v) = v]$, and
(2) $\forall i \in \{1, \ldots, m\}, \forall a \in \{0,1\}^i [a \notin$ nodes$(D) \Rightarrow T_i(a) = \mu]$.
C3. For all $(u, v) \in$ S-edges(S),
(1) $T_{|v|}(u \cdot l(u, v)) = v$, and
(2) $|u| < \forall j < |v|$, $T_j(u \cdot pre(l(u, v), j - |u|)) = \mu$.

Then, $\mathcal{D}(S) = D$ if the cardinality of S-nodes(S) is exactly the number of nodes in D.

Proof. For $v \in$ S-nodes(S), let $N(v)$ denote the node in D to which the assignment made by assigning the ith bit of string v to x_i leads. Mapping N from S-nodes(S) to the set of nodes in D is well-defined by C1(1), one-to-one by C1(3), and onto by the assumption that |S-nodes(S)| is equal to the number of nodes in D. The label of v in S and the label of $N(v)$ in D are $x_{|v|+1}$ if $|v| \neq m$, and they also coincide even when $|v| = m$ by C1(2). Hence, to show $\mathcal{D}(S) = D$, we only have to prove that for any $v_1, v_2 \in$ S-nodes(S), if there exists an edge $(N(v_1), N(v_2))$ labeled b in D, $(v_1, v_2) \in$ S-edges(S) and the first bit of $l(v_1, v_2)$ is b.

Assume that there exists an edge $(N(v_1), N(v_2))$ labeled b in D for arbitrary $v_1, v_2 \in$ S-edges(S). Let (v_1, v) be the edge in S which goes out from node v_1 and is labeled by a string with the first bit b. Assume that $|v| < |v_2|$. Since $v_1 \cdot l(v_1, v) \notin$ nodes(D), $T_{|v|}(v_1 \cdot l(v_1, v)) = \mu$ by C2(2), so $(v_1, v) \notin$S-edges(S) by C3(1), which is a contradiction. Hence, $|v| \geqslant |v_2|$. Assume that $|v| > |v_2|$. Since $v_1 \cdot \text{pre}(l(v_1, v), |v_2| - |v_1|) \overset{D}{=} v_2$, $T_{|v_2|}(v_1 \cdot \text{pre}(l(v_1, v), |v_2| - |v_1|)) = v_2$ by C2(1) and P1, which contradicts C3(2). Therefore, $|v| = |v_2|$. Since $v_1 \cdot l(v_1, v) \overset{D}{=} v_2$, $T_{|v_2|}(v_1 \cdot l(v_1, v)) = v_2$ by C2(1) and P1. On the other hand, $T_{|v_2|}(v_1 \cdot l(v_1, v)) = v$ by C3(1). Hence, $v = v_2$. □

5. Algorithm

In this section, we will explain our algorithm QLearn-π-OBDD, which always outputs the reduced OBDD of any target function with the ordering π using equivalence queries and membership queries.

The algorithm is shown in Fig. 3. First, QLearn-π-OBDD asks equivalence queries (EQs) for two trivial OBDDs, denoted by **1** and **0**, the OBDDs being composed of only one sink node. If 'YES' is returned to one of these two queries, then the algorithm outputs the hypothesis used by the query and stops. Otherwise, the algorithm makes an initial OBDDAS and classification trees by the procedure Initial-Hypothesis from the two counterexamples e_0 and e_1 with $D(e_0) = 0$ and $D(e_1) = 1$, where D is a target function.

The procedure Initial-Hypothesis works as follows. Let $\text{cro}(e_0, e_1, i)$ denote a string of length m constructed by concatenating $\text{pre}(e_0, m - i)$ and $\text{suf}(e_1, i)$, where $\text{pre}(e_0, m - i)$ is the prefix of e_0 with length $m - i$ and $\text{suf}(e_1, i)$ is the suffix of e_1 with length i. Since $e_0 = \text{cro}(e_0, e_1, 0)$ and $e_1 = \text{cro}(e_0, e_1, m)$, there exists i with $0 < i \leqslant m$ such that $D(\text{cro}(e_0, e_1, i - 1)) = 0$ and $D(\text{cro}(e_0, e_1, i)) = 1$, and such an i can be found by a binary search using $\lceil \log_2 m \rceil$ membership queries. In the procedure, an initial OBDDAS S^0 and initial classification trees T_j^0 for $j = 1, \ldots, m$ as shown in Fig. 4 are constructed. Note that S^0 and $\{T_1^0, \ldots, T_m^0\}$ satisfy the conditions C1, C2, and C3 in Lemma 3.

```
Algorithm QLearn-π-OBDD()
Output: the reduced OBDD of a target function with the ordering π
begin
 1  (Ans, e_0) := EQ(1)
 2  if Ans = YES then output 1 and stop
 3  (Ans, e_1) := EQ(0)
 4  if Ans = YES then output 0 and stop
 5  (S, T_1, T_2, ..., T_m) := Initial-Hypothesis(e_0, e_1)
 6  (Ans, e) := EQ(D(S))
 7  while Ans = NO do
 8     (S, T_1, T_2, ..., T_m) := Update-Hypothesis(S, T_1, T_2, ..., T_m, e)
 9     (Ans, e) := EQ(D(S))
10  enddo
11  output D(S)
end
```

Fig. 3. Learning algorithm of OBDDs with the ordering π.

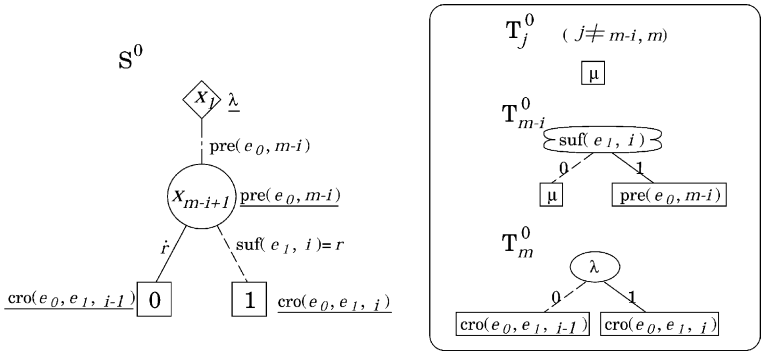


Fig. 4. Initial OBDDAS S^0 and initial classification trees T_j^0 for $j = 1, \ldots, m$: T_{m-i}^0 has one twin-test node, T_m^0 has one single-test node, and the other T_j^0s are composed of only one leaf node labeled μ.

Procedure Update-Hypothesis$(S, T_1, T_2, ..., T_m, e)$
Output: the updated OBDDAS S and classification trees $T_1, T_2, ..., T_m$
begin
1 $(p_1, p_2, ..., p_k) :=$ the sequence of access strings of nodes in S
passed by the counterexample e in the passing order
2 $i :=$ the index i such that $1 \leq i < k$ and
$D(e) = D(p_i \cdot \mathrm{suf}(e, m - |p_i|)) \neq D(p_{i+1} \cdot \mathrm{suf}(e, m - |p_{i+1}|))$
3 **if** $D(p_i \cdot l(p_i, p_{i+1}) \cdot \mathrm{suf}(e, m - |p_{i+1}|)) = D(e)$ **then**
4 $(S, T_1, T_2, ..., T_m) :=$ NodeSplit$(S, T_1, T_2, ..., T_m, e)$
5 **else**
6 $(S, T_1, T_2, ..., T_m) :=$ NewBranchingNode$(S, T_1, T_2, ..., T_m, e, p_i, p_{i+1})$
7 **endif**
8 return $(S, T_1, T_2, ..., T_m)$
end

Fig. 5. Procedure for updating a current hypothesis using a counter example.

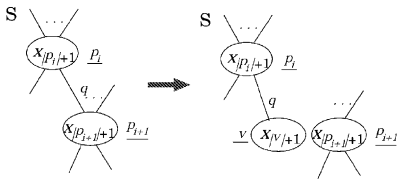
Assume that the algorithm has a current OBDDAS S and current classification trees T_i for $i = 1, \ldots, m$. The algorithm asks an equivalence query for current hypothesis OBDD $\mathcal{D}(S)$ and if a counterexample e is returned, it executes the procedure Update-Hypothesis shown in Fig. 5. This process is repeated until 'YES' is returned to the equivalence query.

Each execution of the procedure Update-Hypothesis finds one node of the target-reduced OBDD and updates the current hypothesis. Consider the path in S made by a given counterexample e. Assume that there are k nodes on the path and let p_i be the access string of the ith node on the path from the root node. Note that we abuse notation and let p_i denote a node itself as well as an access string. Since e is a counterexample for S, the leaf node p_k reached by the path is not correct, that is, $D(p_k) \neq D(e)$. Let $e_i = \mathrm{suf}(e, m - |p_i|)$. Since $p_k = p_k e_k$ and $e = p_1 e_1$, there must exist i such that $1 \leqslant i < k$ and $D(p_i e_i) = D(e) \neq D(p_{i+1} e_{i+1})$. Such i is calculated at Step 2 using membership queries. For this i, let q denote the label of the edge (p_i, p_{i+1}), that is, $q = l(p_i, p_{i+1})$. There are two cases depending on the value of $D(p_i q e_{i+1})$. When $D(p_i q e_{i+1}) = D(e) \neq D(p_{i+1} e_{i+1})$, $p_i q$ and p_{i+1} must reach different nodes, and this case is dealt with in the procedure NodeSplit (Fig. 6), where one single-test

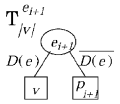
Procedure NodeSplit$(S, T_1, T_2, ..., T_m, e, p_i, p_{i+1})$
/* Called when $D(e) = D(p_i q e_{i+1}) \neq D(p_{i+1} e_{i+1})$,
where e_{i+1} and q are defined at Step 1 */
Output: the updated OBDDAS S and classification trees $T_1, T_2, ..., T_m$
begin

1 $e_{i+1} := \mathrm{suf}(e, m - |p_{i+1}|)$, $q := l(p_i, p_{i+1})$, $v := p_i q$

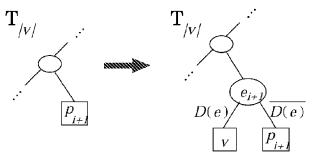
2 Add a node v labeled $x_{|v|+1}$ and an edge (p_i, v) labeled q to S, and remove the edge (p_i, p_{i+1}) from S.



3 Create a tree $T^{e_{i+1}}_{|v|}$ that is composed of a single-test root node labeled e_{i+1} and its two child nodes labeled p_{i+1} and v which are connected by edges labeled $\overline{D(e)}$ and $D(e)$, respectively. Here, $\overline{D(e)}$ is the label opposite to $D(e)$.



4 Replace leaf node labeled p_{i+1} of the tree $T_{|v|}$ with the tree $T^{e_{i+1}}_{|v|}$.



5 Update the OBDDAS S as follows: for all $(v_1, p_{i+1}) \in$ S-edges(S), classify the string $v_1 \cdot l(v_1, p_{i+1})$ into v or p_{i+1} by $T^{e_{i+1}}_{|v|}$. If $v_1 \cdot l(v_1, p_{i+1})$ is classified into v, remove the edge (v_1, p_{i+1}) and add an edge (v_1, v) labeled with the same string as the removed one.

6 $t :=$ the label of the last twin-test node in $T_{|v|}$ on the path from the root to the v-labeled leaf

7 Add new edges $(v, T_{|v|+j_1}(v \cdot \mathrm{pre}(t, j_1)))$ labeled $\mathrm{pre}(t, j_1)$ and $(v, T_{|v|+j_2}(v \cdot \mathrm{pre}(\dot{t}, j_2)))$ labeled $\mathrm{pre}(\dot{t}, j_2)$ to S, where j_1 and j_2 are the minimum natural numbers such that $T_{|v|+j_1}(v \cdot \mathrm{pre}(t, j_1)) \neq \mu$ and $T_{|v|+j_2}(v \cdot \mathrm{pre}(\dot{t}, j_2)) \neq \mu$, respectively. Here, $\dot{t}$ denotes the string obtained by flipping the first bit of t.

8 return $(S, T_1, T_2, ..., T_m)$

end

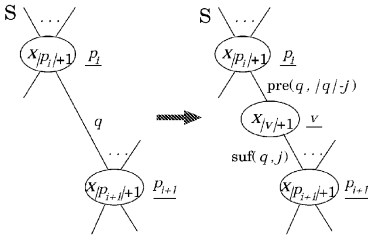
Fig. 6. Procedure for the case in which a node of a current OBDDAS is split.

node is added to one of the classification trees. When $D(p_i q e_{i+1}) = D(p_{i+1} e_{i+1}) \neq D(e)$, there must exist a node between p_i and p_{i+1} from which the path for $p_i e_i$ and the path for $p_i q e_{i+1}$ branch, and this case is dealt with in the procedure NewBranchingNode (Fig. 7), where one twin-test node is added to one of the classification trees. Both procedures add a new node v to the current OBDDAS (Step 2 in Fig. 6, Step 5 in Fig. 7), update $T_{|v|}$ (Step 4 in Fig. 6, Step 13 in Fig. 7), change all edges that must enter v (Step 5 in Fig. 6, Step 14 in Fig. 7) and add edges going out from v (Step 7 in Fig. 6, Step 15 in Fig. 7).

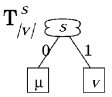
Some examples are given here for a better understanding of the above two cases. Let D in Fig. 8 be the target OBDD to learn. Note that the solid lines represent 1-labeled edges and the broken lines

Procedure NewBranchingNode($S, T_1, T_2, ..., T_m, e, p_i, p_{i+1}$)
/* Called when $D(e) = D(p_i e_i) \neq D(p_i q e_{i+1})$,
where e_i, q and e_{i+1} are defined at Step 1 and Step 2 */
Output: the updated OBDDAS S and classification trees $T_1, T_2, ..., T_m$
begin

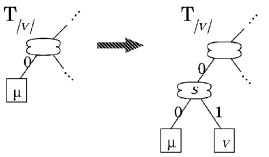
1 $q := l(p_i, p_{i+1})$
2 $j :=$ the index j with $1 \leq j \leq |q|$ such that $D(e) = D(p_i \cdot \mathrm{cro}(q, f, j) \cdot e_{i+1}) \neq D(p_i \cdot \mathrm{cro}(q, f, j-1) \cdot e_{i+1})$, where $e_{i+1} = \mathrm{suf}(e, m - |p_{i+1}|)$ and $f = \mathrm{pre}(e_i, |q|)$ for $e_i = \mathrm{suf}(e, m - |p_i|)$
3 $v := p_i \cdot \mathrm{pre}(q, |q| - j)$
4 **if** $j \neq |q|$ **then**
5 Add a node v labeled $x_{|v|+1}$ and edges (p_i, v) labeled $\mathrm{pre}(q, |q| - j)$ and (v, p_{i+1}) labeled $\mathrm{suf}(q, j)$ to S, and remove the edge (p_i, p_{i+1}) from S.



6 **else**
7 Do nothing. (p_i is an access string for a dummy node.)
8 **endif**
9 $r := \mathrm{suf}(e, m - |v|)$
10 **if** $D(e) = 1$ **then** $s := r$
11 **else** $s := \dot{r}$
12 Create a tree $T^s_{|v|}$ that is composed of a twin-test node labeled s and its two child nodes labeled μ and v which are connected by edges labeled 0 and 1, respectively.



13 Replace the leaf node labeled μ of the tree $T_{|v|}$ with the tree $T^s_{|v|}$.



14 Update the OBDDAS S as follows: for all $(v_1, v_2) \in$ S-edges(S) with $|v_1| < |v| < |v_2|$, classify the string $v_1 g$ into v or μ by $T^s_{|v|}$, where $g = \mathrm{pre}(l(v_1, v_2), |v| - |v_1|)$. If $v_1 g$ is classified into v, remove the edge (v_1, v_2) and add an edge (v_1, v) labeled g.
15 Add a new edge $(v, T_{|v|+j}(v \cdot \mathrm{pre}(r, j)))$ labeled $\mathrm{pre}(r, j)$ to S, where j is the minimum natural number such that $T_{|v|+j}(v \cdot \mathrm{pre}(r, j)) \neq \mu$.
16 return $(S, T_1, T_2, ..., T_m)$
end

Fig. 7. Procedure for the case in which a new node is added on an edge.

represent 0-labeled edges. Assume that our algorithm constructs OBDDAS S_1 in Fig. 8 after receiving several counterexamples.[3] Consider the case that the counterexample 00000011 is returned to the equivalence query for $\mathcal{D}(S_1)$. In this case, the nodes in S_1 on the path made by the counterexample

[3] Actually, the following sequence of four counterexamples makes the algorithm construct S_1: 01101111, 10100110, 10001011, 01101001.

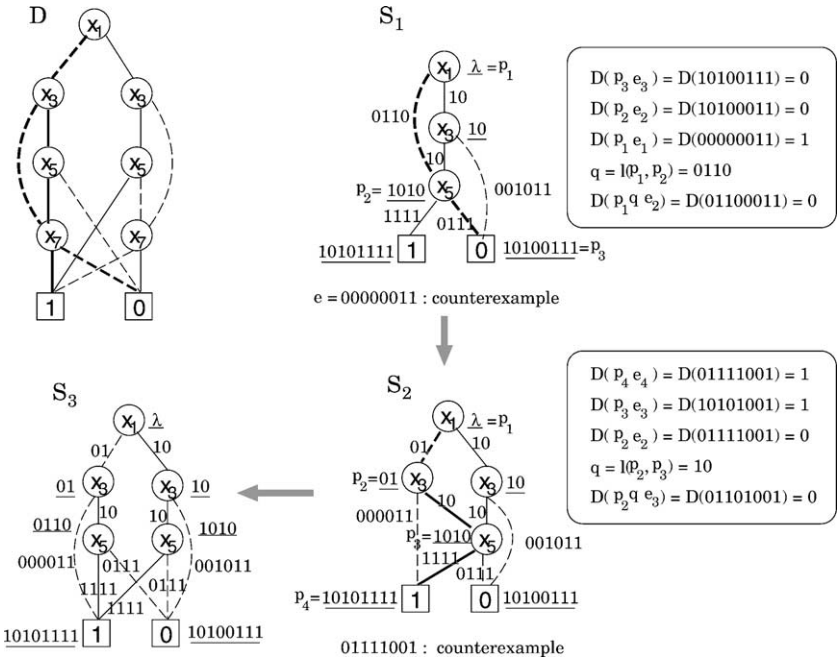


Fig. 8. Examples of the cases dealt with in the procedures NodeSplit and NewBranchingNode.

are $p_1 = \lambda$, $p_2 = 1010$ and $p_3 = 10100111$. In the procedure Update-Hypothesis, this case is dealt with in the procedure NewBranchingNode because $D(p_1e_1) \neq D(p_2e_2)$ and $D(p_1qe_2) = D(p_2e_2)$. In the procedure NewBranchingNode, S_1 is updated to S_2 in Fig. 8. Next, consider the case that the counterexample 01111001 is returned to the equivalence query for $\mathcal{D}(S_2)$. In this case, the nodes on the path made by the counterexample in S_2 are $p_1 = \lambda$, $p_2 = 01$, $p_3 = 1010$ and $p_4 = 10101111$. In the procedure Update-Hypothesis, this case is dealt with in the procedure NodeSplit because $D(p_2e_2) \neq D(p_3e_3)$ and $D(p_2qe_3) \neq D(p_3e_3)$. In the procedure NodeSplit, S_2 is updated to S_3 in Fig. 8.

6. Correctness and efficiency

Lemma 4. *For a target OBDD D, assume that an OBDDAS S and classification trees T_i for $i = 1, \ldots, m$ satisfy C1, C2, and C3 in Lemma 3 and that S has two sink nodes. Let e be a counterexample of D for $\mathcal{D}(S)$. Let $(S', T'_1, \ldots, T'_m)$ denote the output of the procedure Update-Hypothesis for $(S, T_1, \ldots, T_m, e)$. Then, $(S', T'_1, \ldots, T'_m)$ satisfies C1, C2, and C3, and $|S\text{-nodes}(S')| = |S\text{-nodes}(S)| + 1$.*

Proof.

(1) When Update-Hypothesis executes the procedure Node-Split.

In this case, one new internal node v is added to S at Step 2, so $|\text{S-nodes}(S')| = |\text{S-nodes}(S)| + 1$.

First, we show that S' satisfies condition C1. $T_{|v|}(v) = T_{|v|}(p_i \cdot l(p_i, p_{i+1})) = p_{i+1}$ by C3(1), and this fact implies $v \in \text{nodes}(D)$ by C2(2). Thus, S' satisfies C1(1). Let us show S' satisfies C1(3). Assume that $v \overset{D}{=} u \in \text{S-nodes}_{|p_{i+1}|}(S) - \{p_{i+1}\}$. Then, $T_{|p_{i+1}|}(v) = T_{|p_{i+1}|}(u) = u$ by C2(1), which contradicts the fact

implied by C3(1) that $T_{|p_{i+1}|}(v) = T_{|p_{i+1}|}(p_i \cdot l(p_i, p_{i+1})) = p_{i+1}$. Thus, $v \notin \text{S-nodes}_{|p_{i+1}|}(S) - \{p_{i+1}\}$. It also holds that $v \overset{D}{\neq} p_{i+1}$ because $D(ve_{i+1}) \neq D(p_{i+1}e_{i+1})$. Therefore, S' satisfies C1(3). By the assumption that S already has two sink nodes, v can not be a sink node if S' satisfies C1(1) and C1(3). Then, $\text{S-nodes}_m(S') = \text{S-nodes}_m(S)$, so C1(2) is satisfied.

Next, we prove that $(S', T'_1, \ldots, T'_m)$ satisfies condition C2. Since $T'_h = T_h$ for all $h \in \{1, \ldots, m\} - \{|v|\}$, we only have to check $T'_{|v|}$. For $u \in \text{S-nodes}_{|v|}(S) - \{p_{i+1}\}$, $T'_{|v|}(u) = u$ because $T_{|v|}(u) = u$ and $T'_{|v|}$ is made from $T_{|v|}$ by replacing the p_{i+1}-labeled leaf with the tree $T_{|v|}^{e_{i+1}}$, which is composed of one single-test internal node labeled e_{i+1}, the leaf labeled v and the leaf labeled p_{i+1}. $T_{|v|}(v) = T_{|v|}(p_{i+1}) = p_{i+1}$ by C3(1) and C2(1), so both v and p_{i+1} reach the root node of $T_{|v|}^{e_{i+1}}$ in $T'_{|v|}$. Thus, $T'(v) = T_{|v|}^{e_{i+1}}(v) = v$ and $T'_{|v|}(p_{i+1}) = T_{|v|}^{e_{i+1}}(p_{i+1}) = p_{i+1}$ by the definition of $T_{|v|}^{e_{i+1}}$. For all $a \in \{0,1\}^{|v|}$ s.t. $a \notin \text{nodes}(D)$, $T_{|v|}(a) = \mu$ by C2(2), so $T'_{|v|}(a) = \mu$ because a reaches the μ-labeled leaf in $T_{|v|}$, which is not modified in $T'_{|v|}$.

Finally, we show that $(S', T'_1, \ldots, T'_m)$ satisfies condition C3. First, consider the edge (p_i, v) added at Step 2. This edge satisfies C3(1) because $T'_{|v|}(p_i \cdot l(p_i, v)) = T'_{|v|}(v) = v$ by C2(1). This edge also satisfies C3(2) because for j with $|p_i| < j < |v|$,

$$T'_j(p_i \cdot \text{pre}(l(p_i, v), j - |p_i|)) = T_j(p_i \cdot \text{pre}(l(p_i, p_{i+1}), j - |p_i|)) = \mu.$$

Next, consider the edges (v_1, v) added at Step 5. $T_{|v|}(v_1 \cdot l(v_1, p_{i+1})) = p_{i+1}$ by C3(1), so $v_1 \cdot l(v_1, p_{i+1})$ reaches the root of $T_{|v|}^{e_{i+1}}$ in $T'_{|v|}$. Since $T_{|v|}^{e_{i+1}}(v_1 \cdot l(v_1, p_{i+1})) = v$, $T'_{|v|}(v_1 \cdot l(v_1, v)) = v$. Thus, the edges (v_1, v) satisfy C3(1). These edges also satisfy C3(2) because for all j with $|v_1| < j < |v|$,

$$T'_j(v_1 \cdot \text{pre}(l(v_1, v), j - |v_1|)) = T_j(v_1 \cdot \text{pre}(l(v_1, p_{i+1}), j - |v_1|)) = \mu.$$

The edges added at Step 7 are made so as to satisfy C3.

The edges in $\text{S-edges}(S) \cap \text{S-edges}(S')$ is shown to satisfy C3 as follows. Since $T'_{|a|}(a) = T_{|a|}(a)$ for all a satisfying $T_{|a|}(a) \neq p_{i+1}$, C3(2) is trivially satisfied by them for $T'_1, \ldots, T'_m$ and the edges (v_1, v_2) in S that do not satisfy C3(1) for $T'_1, \ldots, T'_m$ are only those for which $v_2 = p_{i+1}$ and $T'_{|v|}(v_1 \cdot l(v_1, p_{i+1})) = T_{|v|}^{e_{i+1}}(v_1 \cdot l(v_1, p_{i+1})) = v$. However, all such edges (v_1, v_2) are removed at Step 2 and Step 5, so C3(1) is also satisfied by all of the edges in $\text{S-edges}(S) \cap \text{S-edges}(S')$.

(2) When Update-Hypothesis executes the procedure NewBranchingNode

In this case, one new internal node v is added to S at Step 5, so $|\text{S-nodes}(S')| = |\text{S-nodes}(S)| + 1$.

First, we show that S' satisfies condition C1. Since v is not a sink node, $\text{S-nodes}_m(S') = \text{S-nodes}_m(S)$. Thus, C1(2) is satisfied by S'. Since $D(vr) = D(p_i \cdot \text{cro}(q, f, j) \cdot e_{i+1}) \neq D(p_i \cdot \text{cro}(q, f, j-1) \cdot e_{i+1}) = D(v\dot{r})$, $v \in \text{nodes}(D)$, which means that S' satisfies C1(1). To show that S' satisfies C1(3), we assume that $\exists u \in \text{S-nodes}(S)[u \overset{D}{=} v]$. Then, $u \overset{D}{=} v$ implies $T_{|v|}(v) = T_{|v|}(u)$, and furthermore $T_{|v|}(v) = u$ by C2(1). However, $T_{|v|}(v) = T_{|v|}(p_i \cdot \text{pre}(l(p_i, p_{i+1}), |v| - |p_i|)) = \mu$ by C3(2), which contradicts $T_{|v|}(v) = u$.

Next, we prove that $(S', T'_1, \ldots, T'_m)$ satisfies condition C2. Since $T'_h = T_h$ for all $h \in \{1, \ldots, m\} - \{|v|\}$, we only have to check $T'_{|v|}$. For $u \in \text{S-nodes}(S)$ with $|u| = |v|$, $T'_{|v|}(u) = u$ because $T_{|v|}(u) = u$ and $T'_{|v|}$ is made from $T_{|v|}$ by replacing the μ-labeled leaf with the tree $T^s_{|v|}$, which is composed of one twin-test internal node labeled s, the v-labeled leaf and the μ-labeled leaf. For the new node

v, $T_{|v|}(v) = \mu$ by C3(2), so v reaches the root node of $T^s_{|v|}$ in $T'_{|v|}$. Since $T^s_{|v|}(v) = v$, $T'_{|v|}(v) = v$. For all $a \in \{0,1\}^{|v|}$ s.t. $a \notin \text{nodes}(D)$, $T_{|v|}(a) = \mu$ by C2(2), so a also reaches the root node of $T^s_{|v|}$ in $T'_{|v|}$. Since $D(as) = D(a\dot{s})$, a reaches the μ-labeled leaf, that is, $T'_{|v|}(a) = \mu$.

Finally, we show that $(S', T'_1, \ldots, T'_m)$ satisfies condition C3. First, consider the edges (p_i, v), (v, p_{i+1}) added at Step 5. These two edges satisfy C3(1) because $T'_{|v|}(p_i \cdot l(p_i, v)) = T'_{|v|}(v) = v$ and $T'_{|p_{i+1}|}(v \cdot l(v, p_{i+1})) = T_{|p_{i+1}|}(p_i \cdot l(p_i, p_{i+1})) = p_{i+1}$. When $|p_i| < j_1 < |v| < j_2 < |p_{i+1}|$,

$$T'_{j_1}(p_i \cdot \text{pre}(l(p_i, v), j_1 - |p_i|)) = T_{j_1}(p_i \cdot \text{pre}(l(p_i, p_{i+1}), j_1 - |p_i|)) = \mu$$

and

$$T'_{j_2}(v \cdot \text{pre}(l(v, p_{i+1}), j_2 - |v|)) = T_{j_2}(p_i \cdot \text{pre}(l(p_i, p_{i+1}), j_2 - |p_i|)) = \mu.$$

Thus, these two edges also satisfy C3(2).

Next, consider the edges (v_1, v) added at Step 14. $T_{|v|}(v_1 \cdot \text{pre}(l(v_1, v_2), |v| - |v_1|)) = \mu$ by C3(2) and the μ-labeled leaf is replaced with $T^s_{|v|}$ in $T'_{|v|}$, so $T'_{|v|}(v_1 \cdot l(v_1, v)) = T^s_{|v|}(v_1 \cdot \text{pre}(l(v_1, v_2), |v| - |v_1|)) = v$. Furthermore, for $|v_1| < j < |v|$,

$$T'_j(v_1 \cdot \text{pre}(l(v_1, v), j - |v_1|)) = T_j(v_1 \cdot \text{pre}(l(v_1, v_2), j - |v_1|)) = \mu.$$

Thus, all of the edges (v_1, v) satisfy C3.

The edge added at Step 15 is made so as to satisfy C3.

The edges in S-edges(S) $\cap$ S-edges(S') is shown to satisfy C3 as follows. Since $T'_{|a|}(a) = T_{|a|}(a)$ for all a satisfying $T_{|a|}(a) \neq \mu$ or $|a| \neq |v|$, C3(1) is trivially satisfied by them for $T'_1, \ldots, T'_m$ and the edges (v_1, v_2) in S that do not satisfy C3(2) for $T'_1, \ldots, T'_m$ are only those for which $|v_1| < |v| < |v_2|$ and $T'_{|v|}(v_1 \cdot \text{pre}(l(v_1, v_2), |v| - |v_1|)) = v$. However, all such edges (v_1, v_2) are removed at Step 5 and Step 14, so C3(2) is also satisfied by all of the edges in S-edges(S) $\cap$ S-edges(S'). $\square$

Theorem 5. *For an arbitrary target OBDD D in reduced form with ordering π, algorithm QLearn-π-OBDD exactly learns D using at most n equivalence queries and at most $2n(\lceil \log_2 m \rceil + 3n)$ membership queries, where $m(\geqslant 1)$ is the number of variables and n is the number of nodes in D.*

Proof. We assume that D has two sink nodes because the case with one sink node is trivial. First, we prove that QLearn-π-OBDD outputs D with at most n equivalence queries. It is trivial that S^0 and $T^0_1, \ldots, T^0_m$ satisfy the conditions C1, C2, and C3 in Lemma 3. By Lemma 4, the number of non-dummy nodes of the OBDDAS S maintained by the algorithm increases by one for every execution of the procedure Update-Hypothesis, which updates S and classification trees $T_1, \ldots, T_m$ while satisfying C1, C2, and C3. Thus, |S-nodes(S)| reaches just the number of nodes in D after executing the procedure Update-Hypothesis $n - 3$ times. Then, Lemma 3 guarantees that $\mathcal{D}(S) = D$. Therefore, the nth equivalence query by QLearn-π-OBDD is answered with 'YES' because three equivalence queries are asked before the first execution of Update-Hypothesis and one equivalence query is asked after each execution of Update-Hypothesis.

Next, we consider the number of membership queries. To construct S^0 and $T^0_1, \ldots, T^0_m$, the algorithm uses $\lceil \log_2 m \rceil$ membership queries. Let us consider how many membership queries are asked in one execution of Update-Hypothesis.

(1) When Update-Hypothesis executes the procedure NodeSplit
At Step 2 of Update-Hypothesis, at most $\lceil \log_2 m \rceil$ membership queries are asked. At Step 5 of NodeSplit, at most n membership queries are asked because at most one membership query is asked for each node in S. In each execution of Step 7 of NodeSplit, at most $4n$ membership queries are asked because at most two membership queries are asked for each internal node of $T_1, \ldots, T_m$ and the number of internal nodes is at most n. In this case, the algorithm asks at most $\lceil \log_2 m \rceil + 5n$ membership queries in total.
(2) When Update-Hypothesis executes the procedure NewBranchingNode.
At Step 2 of Update-Hypothesis and Step 2 of NewBranchingNode, at most $2\lceil \log_2 m \rceil$ membership queries are asked by using a binary search. At Step 14 of NewBranchingNode, at most $4n$ membership queries are asked because at most two membership queries are asked for each edge in S. At step 15 of NewBranchingNode, at most $2n$ membership queries are asked for the reason described above. In this case, the algorithm asks at most $2\lceil \log_2 m \rceil + 6n$ membership queries in total.

Thus, QLearn-π-OBDD asks at most $2n(\lceil \log_2 m \rceil + 3n)$ membership queries. □

Assuming that all of the operations on strings of length m need at most $O(m)$ steps, it can be easily shown that the running time is at most $O(nm(\log m + n))$, that is, a factor of $O(m)$ larger than the number of queries.

7. Experiments

From the results obtained by worst-case analysis in Sections 4.1 and 6, the number of membership queries asked by our algorithm is dependent on the number of variables at most logarithmically, while the number of those asked by a direct adaptation of the modified Learn-Automaton is dependent on the number of variables at least linearly. In this section, we show experimental results which indicate that a similar performance difference between two algorithms appears to exist even for target OBDDs generated at random according to a certain natural distribution.

For a fixed n and for various ms, some reduced OBDDs with n nodes each labeled with one variable selected from m variables were generated by the OBDD generation procedure described in Appendix A, and the number of membership queries asked by the algorithms was experimentally investigated.

In the first experiment, n was fixed to 100 and m was set to $100, 200, \ldots, 900$ or 1000. For each m, 10 OBDDs were generated and each algorithm was executed 5 times for each OBDD. Then the results were obtained by averaging the number of membership queries over these 50 runs for each m. Note that the number of queries asked by the algorithms possibly varies even for the same OBDD because of the introduction of randomness to the selection of counterexamples.

We ran two algorithms, QLearn-π-OBDD and a direct adaptation of the modified Learn-Automaton, which we call DFALearn-π-OBDD here. Note that DFALearn-π-OBDD used the fact that the value of a target function is 0 for all strings of which length is not m to reduce the number of membership queries.

The results are shown in Fig. 9. The difference in the performance of the two algorithms is clear, but with respect to the number of queries as a function of the number of variables, the increasing

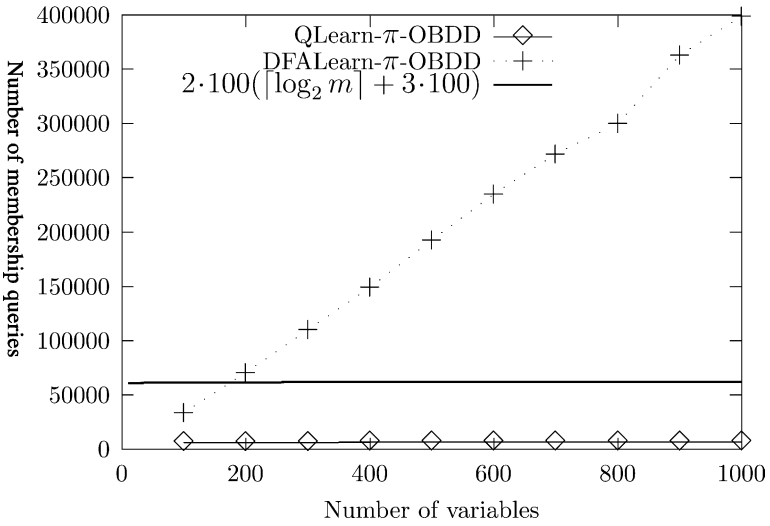


Fig. 9. Average number of membership queries asked by QLearn-π-OBDD and DFALearn-π-OBDD.

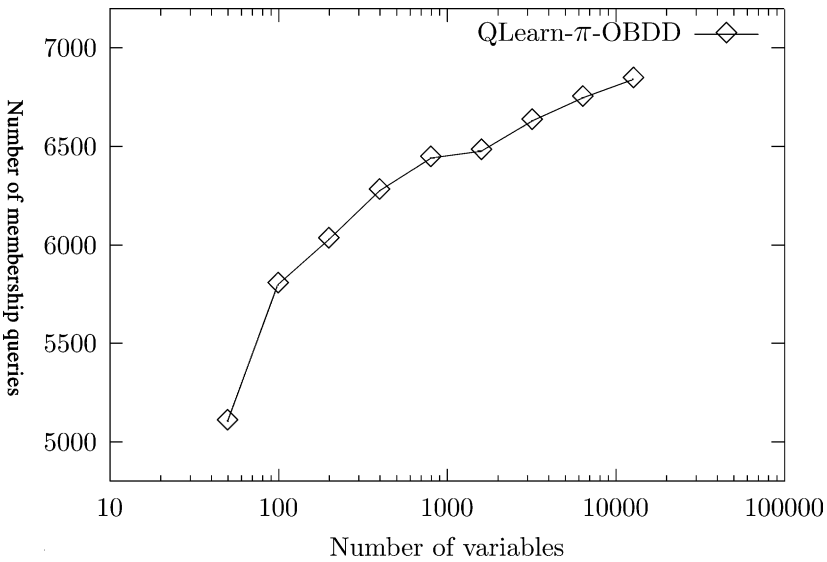


Fig. 10. Average number of membership queries requested by QLearn-π-OBDD.

rate is too small to estimate the order of magnitude for QLearn-π-OBDD. Another experiment in which m was set to $50, 100, 200, \ldots, 6400$ or 12800 while n was fixed to 100 was therefore carried out.

The results are shown in Fig. 10, where a log scale is used on the axis of the number of variables. According to the results of these experiments, the number of membership queries asked by QLearn-π-OBDD appears to increase at most logarithmically, while the number of membership queries asked by DFALearn-π-OBDD appears to increase at least linearly.

8. Conclusions

We proposed a new algorithm for query learning of OBDDs with a given variable ordering, proved its correctness and analyzed its query complexity. Compared to the previous best known algorithm proposed by Gavaldà and Guijarro, our algorithm has the same upper bound on the number of equivalence queries and an upper bound on the number of membership queries that is smaller by a factor of $O(m)$, where m is the number of variables. Our algorithm makes use of classification trees as the DFA learning algorithm proposed by Kearns and Vazirani does, but our classification trees have special internal nodes called *twin-test nodes* and special leaf nodes labeled μ. Through introduction of these nodes, our algorithm can identify a target OBDD without identifying any of the states needed to skip irrelevant bits in its DFA-representation. As far as such algorithms that identify all of those states are considered, the barrier of linear dependency on m for the number of membership queries appears not to be broken through from the results presented in Section 4.1. In our experiments, a similar performance difference was observed even for target OBDDs generated at random according to a certain natural distribution.

Acknowledgments

We thank Dr. Naoki Abe of IBM for his helpful comments and advice. We also thank the editor and the anonymous referees for their suggestions which helped us in improving the quality of the paper.

Appendix A. OBDD generation procedure

Input:
m: number of variables
n: number of nodes
Output:
D: an OBDD having n nodes with a variable ordering $x_1 < \cdots < x_m$

(1) For each one of $n-2$ internal nodes, select a label variable from $\{x_1, \ldots, x_m\}$ at random. Reorder nodes so that that the labels $x_{i_1}, \ldots, x_{i_{n-2}}$ of nodes $v_1, \ldots, v_{n-2}$, respectively, satisfy that $i_1 \leqslant \ldots \leqslant i_{n-2}$. Let v_{n-1} and v_n denote the two sink nodes.
(2) For $j = 1$ to $n-2$, do the following procedure.
  (a) If $j > 1$ and v_j has no incoming nodes, redo from the beginning.
  (b) Set k to 0 or 1 at random.
  (c) Add a k-labeled outgoing edge from v_j as follows.
    Let $h_0 = \min\{h > j : i_h > i_j\}$ and $h_1 = \max\{h \geqslant h_0 : i_h = i_{h_0}\}$. For $h_0 \leqslant h \leqslant h_1$, if there is a node v_h that does not have an incoming edge yet, then select one such h at random. Otherwise, select h from $\{h_0, \ldots, n\}$ at random. Add an edge (v_j, v_h).
  (d) Add the other outgoing edge from v_j similarly, but reselect h when the same h is selected.

(3) Decide the labels of the two sink nodes at random.
(4) Transform the current OBDD into the unique OBDD D in the reduced form.
(5) If the number of nodes in D is smaller than n, redo from the beginning.

Appendix B. Proof of Theorem 1

Consider an OBDD shown in the left part of Fig. B.1. The right part in Fig. B.1 is its minimum DFA representation. For identifying this DFA with $m \geqslant 2n$, the modified Learn-Automaton asks at least $mn^2/32$ different membership queries, which is shown as follows.

Consider the case in which the algorithm receives counterexamples $1^m (= e_0), 1^{2m+1}$ and e_i for $i = 1, 2, \ldots, n/2 - 1$ in this order, where e_i is $0^i 1^{m-n/2} 0^{n/2-i}$ when i is an even number and e_i is $0^i 10^{m-n/2-2} 10^{n/2-i}$ when i is an odd number. Here, a^i for any character a denotes a length-i string that is composed of character a only. Note that Learn-Automaton does not always create a hypothesis DFA consistent with given counterexamples, and it is assumed that in such cases the same counterexample is returned for those equivalence queries. Then, the classification tree T shown in Fig. B.2 is constructed, where $\mathrm{pre}(e_i, k)$ and $\mathrm{suf}(e_i, k)$ are the prefix and suffix of e_i with length k, respectively. Considering the fact that the length of a string must be m for being accepted, the classification tree for strings with length $k = 1, \ldots, m-1$ can be seen as T'_k shown in Fig. B.2. Define A as set $\{(i, j, k) : 0 \leqslant i, j \leqslant n/2 - 1, n/2 \leqslant k \leqslant m - n/2, i - j \text{ is odd}\}$. Then for $(i_0, j_0, k_0), (i_1, j_1, k_1) \in A$, $P_{i_0,k_0} S_{j_0 k_0} \neq P_{i_1,k_1} S_{j_1 k_1}$ when $(i_0, j_0, k_0) \neq (i_1, j_1, k_1)$. To construct the part of T corresponding to T'_k for each $k = n/2, n/2 + 1, \ldots, m - n/2$, at least $n^2/16$ membership queries are needed because $\lfloor (i+1)/2 \rfloor + 1$ membership queries for strings $P_{ik} S_{jk}$ $(j \leqslant i + 1, (i, j, k) \in A)$ are needed for each leaf with access string $P_{i,k}$ for $i = 0, 1, \ldots, n/2 - 2$. Thus, totally, at least $(m - n + 1) \times (n^2/16) \geqslant mn^2/32$ different membership queries are asked by the modified Learn-Automaton. □

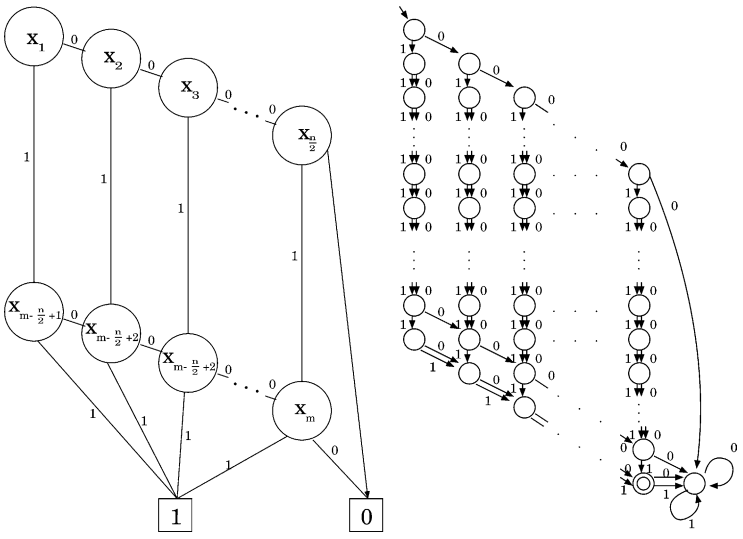


Fig. B.1. An OBDD that requires many membership queries to be identified by the modified Learn-Automaton.

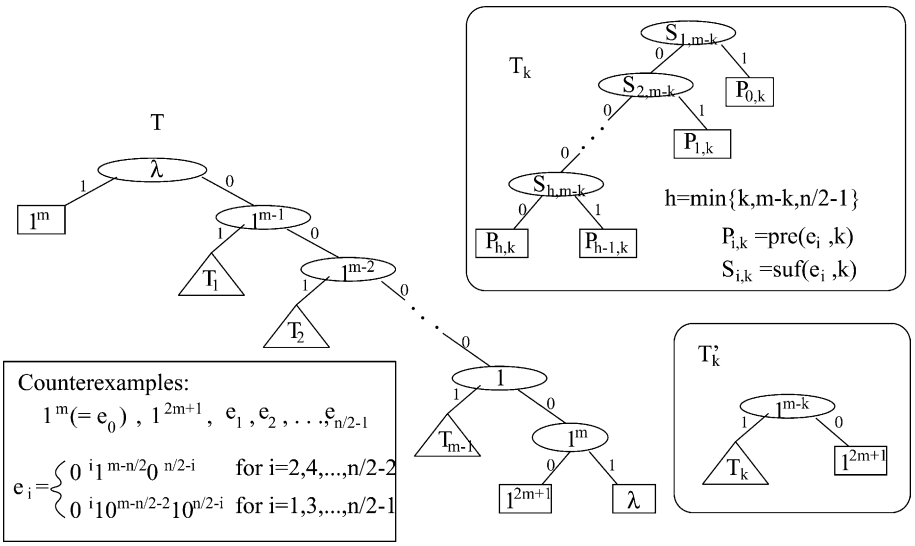


Fig. B.2. Classification tree constructed by Learn-Automaton.

References

[1] D. Angluin, Learning regular sets from queries and counterexamples, Information and Computation 75 (1987) 87–106.

[2] R. Bryant, Symbolic boolean manipulation with ordered binary-decision diagrams, ACM Computing Surveys 24 (3) (1992) 293–318.

[3] F. Bergadano, N. Bshouty, C. Tamon, S. Varricchio, On learning branching programs and small circuits, in: Proceedings of the 3rd European Conference on Computational Learning Theory, Lecture Notes in Artificial Intelligence, vol. 1208, 1997, pp. 150–161.

[4] A. Birkendorf, H. Simon, Using computational learning strategies as a tool for combinatorial optimization, Annals of Mathematics and Artificial Intelligence 22 (1998) 237–257.

[5] N. Bshouty, C. Tamon, D. Wilson, On learning width two branching programs, in: Proceedings of the 9th Annual Conference on Computational Learning Theory, 1996, pp. 224–227.

[6] J.R. Burch, E.M. Clarke, K. McMillan, Symbolic model checking: 10^{20} states and beyond, in: Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science, 1990, pp. 428–439.

[7] F. Ergün, S. Kumar, R. Rubinfeld, On learning bounded-width branching programs, in: Proceedings of the 9th Annual Conference on Computational Learning Theory, 1995, pp. 361–368.

[8] R. Gavaldà, D. Guijarro, Learning ordered binary decision diagrams, in: Proceedings of the 6th International Workshop on Algorithmic Learning Theory, 1995, pp. 228–238.

[9] S.-W. Jeong, F. Somenzi, A new algorithm for the binate covering problem and its application to the minimization of Boolean relations, in: Proceedings of IEEE International Conference on Computer-Aided Design (ICCAD-92), 1992, pp. 417–420.

[10] M. Kearns, U. Vazirani, An Introduction to Computational Learning Theory, The MIT Press, Cambridge, MA, 1994.

[11] J.C. Madre, J.P. Billon, Proving circuit correctness using formal comparison between expected and extracted behavior, in: Proceedings of the 25th ACM/IEEE Design Automation Conference, 1988, pp.205–210.

[12] J.C. Madre, O. Coudert, A logically complete reasoning maintenance system based on a logical constraint solver, in: Proceedings of the 12th International Joint Conference on Artificial Intelligence, 1988, pp. 94–299.

[13] A. Nakamura, Query learning of bounded-width OBDDs, Theoretical Computer Science 241 (2000) 83–114.

[14] R. Rivest, R. Schapire, Inference of finite automata using homing sequences, Information and Computation 103 (1993) 299–347.
[15] V. Raghavan, D. Wilkins, Learning μ-branching programs with queries, in: Proceedings of the 7th Annual Conference on Computational Learning Theory, 1993, pp. 27–36.