Query Learning Algorithm for Ordered Multi-Terminal Binary Decision Diagrams

Atsuyoshi Nakamura

[a]Graduate School of Information Science and Technology, Hokkaido University, Kita 14, Nishi 9, Kita-ku, Sapporo, 060-0814, Hokkaido, Japan

Abstract

We propose a query learning algorithm for ordered multi-terminal binary decision diagrams (OMTBDDs) using at most n equivalence and $2n(\lceil \log_2 m \rceil + 3n)$ membership queries by extending the algorithm for ordered binary decision diagrams (OBDDs). Tightness of our upper bounds is checked in our experiments using synthetically generated target OMTBDDs. Possibility of applying our algorithm to classification problems is also indicated in our other experiments using datasets of UCI Machine Learning Repository.

Keywords: query learning, sample complexity, OMTBDD

1. Introduction

A *binary decision diagram (BDD)* is a representation of a Boolean function, and it is known to be compact for many functions and easy to be manipulated [1]. It is a kind of directed acyclic graph (DAG) that has a root and two sinks labeled 0 and 1. Each non-sink node is labeled a Boolean variable x_i and assignment $x_i = a_i$ for the variables decides a path from the root to one of sinks by selecting a_i-labeled outgoing edge at the node labeled x_i. If the sequence of labeled variables on any path from the root to one of sinks in the passing order is consistent with some variable order, then it is called an *ordered BDD (OBDD)*. An OBDD is very popular due to its good property that any Boolean function can be represented by a unique *reduced OBDD* for any fixed variable order.

A *multi-terminal binary decision diagram (MTBDD)* [2] is an extension of BDD so as to represent a multi-valued function of Boolean variables. The structural difference is the number of sinks only; an MTBDD can have more than two sinks. An *ordered MTBDD (OMTBDD)* is an MTBDD with variable labels of non-sink nodes restricted as an OBDD. An OMTBDD is known to inherit good properties such as the unique reduced form from an OBDD.

In this paper, we extend query learning algorithm for an OBDD to that for an OMTBDD. Query learning that we adopt here is a learning using equivalence and membership queries [3]. In the study of query learning for function class $\mathcal{F}$, we develop an efficient algorithm that can identify an unknown target function f in $\mathcal{F}$ using queries allowed to ask. An equivalence query $\mathrm{EQ}(h)$ for hypothesis $h \in \mathcal{F}$ of the learner's choice is a query to ask whether $h = f$ or not and the answer to the query is 'YES' if $h = f$ and 'NO' otherwise. In the case with answer 'NO', the learner also obtains counterexample e for which $h(e) \neq f(e)$. A membership query is a query to ask the function value $f(a)$ for an assignment a of the learner's choice, and the value $f(a)$ is answered. The oracle that answers to membership queries can be realized as a blackbox function but the oracle that answers to equivalence queries cannot be realized easily. Stochastic testing $h(x) = f(x)$ for randomly sampled x is known to be enough for probably approximately correct (PAC) learning instead of identification [3].

Our query learning algorithm for an OMTBDD, called QLearn-OMTBDD, is an extension of QLearn-π-OBDD [4] that identifies a target OBDD using equivalence and membership queries. In the algorithms, we use (node) classification tree T_i to classify a partial assignment $x_1 = a_1, x_2 = a_2, \ldots, x_{i-1} = a_{i-1}$ into an internal node labeled x_i in the hypothesis OMTBDD or μ in order to judge which node the partial assignment reach or no reachable node (in the case with μ) in it. The judgment by the classification tree is done depending on the answers to the membership queries. Since the number of possible answers for membership queries increases from two to K in the case with an OMTBDD with K sinks, the structure of the trees must be modified, which forces some modifications of their update procedure. Especially, there is a case that no existing edge label corresponds to the answer to a membership query at a node in a classification tree, which never occurs for OBDDs. In such case, the trees must be updated so as to include such edge.

We prove that an arbitrary target reduced OMTBDD can be learned using at most $2n(\lceil \log_2 m \rceil + 3n)$ membership queries and at most n equivalence queries by Algorithm QLearn-OMTBDD, which are exactly the same

upper bounds for OBDDs shown in [4]. The tightness of the upper bounds on these query complexities is checked in our experiments using synthetically generated OMTBDDs. We also check applicability of our algorithm to classification problem. Our multi-terminal extension enables OMTBDDs to represent multi-class classifiers. Even real-valued features can be converted to binary features whose value represents above or below some fixed threshold. OMTBDD representations for classifiers are useful in the point that various operations with some condition-represented OMTBDDs are possible. Through our experiments using 12 real-valued feature datasets in UCI Machine Learning Repository, we show a way of constructing an OMTBDDs by query learning from a tree-based classifier using its training data that are correctly predicted by the classifier, where the classifier is used for answering to membership queries and consistency with the training data is replaced with identification by equivalence queries. As for tree-based classifiers, we use decision trees and random forests. On decision tree classifiers, significant accuracy deterioration cannot be observed except for one dataset, and the number of nodes are kept at most the same number for 5 among 12 datasets. For two datasets, the number of nodes in the learned OMTBDDs is smaller than that in the original decision trees even in the trees whose leaves are shared among those with the same label. On random forest classifiers, accuracy deterioration is observed except two datasets but significant reduction in the number of nodes is achieved for 5 of 10 datasets[1]. As for two of the 5 successfully reduced datasets, accuracy of the OMTBDD learned from a random forest is better than that of the decision tree. Our results on classification problem for these benchmark datasets indicate possibility of application of our query learning algorithm.

Related Work. Query learning is proposed by Angluin [3], and the algorithm for learning deterministic finite automata (DFAs) is one of the most famous query learning algorithms using equivalence and membership queries. In the query learning for DFAs, Kearns and Vazirani [5] used a classification tree instead of an observation table used in Angluin's algorithm. Gavaldà and Guijarro [6] extended Angluin's query learning algorithm for DFAs to the algorithm for OBDDs. Nakamura [4] reduced the number of membership queries used for learning an OBDD by a factor of $O(m)$ by using classification

---

[1]As for two datasets, epileptic seizure and magic datasets, OMTBDDs could not be learned from random forests due to their large number of nodes.

3

trees, where m is the number of variables. The ZDD-version of Nakamura's algorithm was developed by Mizumoto et al. [7].

2. Preliminaries

A *multi-terminal binary decision diagram* (MTBDD) is an extension of a binary decision diagram (BDD) that can represent a function of more than 2 values from domain $\{0,1\}^m$. Let $\{0,\ldots,K-1\}$ for $K \geq 2$ represent the set of values of K-valued functions. An MTBDD representing a K-valued function is a directed acyclic graph with one root and at most K sinks, nodes labeled $0,\ldots,K-1$. The simplest MTBDD is composed of one sink that is also the root, and it represents a constant function. Other MTBDDs have at least two sinks and one internal (non-sink) node. Each internal node is labeled a Boolean variable and has two outgoing edges, 0-labeled and 1-labeled edges. An *ordered MTBDD (OMTBDD)* denotes an MTBDD in which the sequence of variables labeling nodes on any path from the root to one of sinks must be consistent with a certain preset order. The variable order is fixed to $x_1, x_2, ..., x_m$ in this paper.

Given an assignment $x_1 = a_1, \ldots, x_m = a_m$, the value of the function represented by an OMTBDD at the assignment is calculated as follows: starting from its root, selecting the a_i-labeled outgoing edge at a node labeled x_i and taking value $j \in \{0, \ldots, K-1\}$ that is the label of the finally-reached sink. An assignment $x_1 = a_1, \ldots, x_m = a_m$ is also represented by the binary string $a_1 a_2 \ldots a_m$.

We use bold letters like $\boldsymbol{a}, \boldsymbol{b}$ to represent strings and the length of any string $\boldsymbol{a}$ is denoted as $|\boldsymbol{a}|$. The concatenated string of $\boldsymbol{a}$ and $\boldsymbol{b}$ is denoted as $\boldsymbol{a} \cdot \boldsymbol{b}$, which is sometimes abbreviated as $\boldsymbol{ab}$. For a string $\boldsymbol{a}$, $\mathrm{pre}(\boldsymbol{a}, i)$ and $\mathrm{suf}(\boldsymbol{a}, i)$ are the prefix and suffix strings of $\boldsymbol{a}$ with length i, respectively. For two length-i strings $\boldsymbol{a}, \boldsymbol{b}$ and a natural number $j < i$, $\mathrm{cro}(\boldsymbol{a}, \boldsymbol{b}, j)$ denote the length-i string constructed by concatenating $\mathrm{pre}(\boldsymbol{a}, i - j)$ and $\mathrm{suf}(\boldsymbol{b}, j)$. We abuse the notation, and the function represented by an OMTBDD D is also denoted as D, so $D(a_1, a_2, \ldots, a_m)$ is the value of D for the assignment $x_1 = a_1, \ldots, x_m = a_m$ and it is also written as $D(\boldsymbol{a})$ for $\boldsymbol{a} = a_1 a_2 \cdots a_m$.

For node N in OMTBDD D, an *access string of node N* is a string $a_1 a_2 \cdots a_{i-1}$ such that the path on D for assignment $x_1 = a_1, x_2 = a_2, \ldots, x_{i-1} = a_{i-1}$ just reaches node N. The length of the access string is $i - 1$ for nodes with labeled x_i and m for sinks. We let $\mathrm{nodes}_i(D)$ denote the set of length-$(i - 1)$ access strings for nodes in D and let $\mathrm{nodes}(D) = \bigcup_{i=1}^{m+1} \mathrm{nodes}_i(D)$.

Then, for $\boldsymbol{a}, \boldsymbol{b} \in \text{nodes}(D)$, an equivalence relation '$\stackrel{D}{=}$' is defined as follows:

$$\boldsymbol{a} \stackrel{D}{=} \boldsymbol{b} \stackrel{def}{\Leftrightarrow} \boldsymbol{a} \text{ and } \boldsymbol{b} \text{ are access strings to the same node in } D.$$

Let $[\boldsymbol{a}]$ denote the equivalence class of $\boldsymbol{a}$. We call a subset V of $\text{nodes}(D)$ a *node id set of* D if V has just one access string per each node in D. For a node id set V of D, let V_i denote the set of access strings in V whose length is $(i-1)$. Then $V = \bigcup_{i=1}^{m+1} V_i$, and $\text{nodes}_i(D)$ can be partitioned into $\{[\boldsymbol{v}] \mid \boldsymbol{v} \in V_i\}$ and $\text{nodes}(D)$ can be also partitioned into $\{[\boldsymbol{v}] \mid \boldsymbol{v} \in V\}$. We use $\boldsymbol{v} \in V$ as a node id, and let 'node $\boldsymbol{v}$' mean the node with access string $\boldsymbol{v}$.

The learning framework we consider is the *query learning* proposed by Angluin [3]. In this framework, an unknown target function f is identified using *equivalence queries* and *membership queries*. An *equivalence query* asks if f is equivalent to a hypothesis h; if so, 'YES' is returned, and if not, 'NO' and a counterexample e $(f(e) \neq h(e))$ are returned. We treat an equivalence query as a function EQ from a hypothesis h to a pair of an answer and a counterexample (Ans, e) for $\text{Ans} \in \{\text{'YES', 'NO'}\}$, and let $\text{EQ}(h)$ represent (Ans, e). A *membership query* asks for the value $f(a)$ of the target function f for an assignment a.

3. Node Classification Trees

For each $i = 1, 2, \ldots, m+1$, let $\mathcal{S}_i$ be the set of $\{0,1\}$-strings with length $i-1$. Let D be an OMTBDD and let V be a node id set of D. Then, $\mathcal{S}_i$ can be partitioned as $\mathcal{S}_i = \bigcup_{\boldsymbol{v} \in V_i}[\boldsymbol{v}] \cup \left(\mathcal{S}_i \setminus \bigcup_{\boldsymbol{v} \in V_i}[\boldsymbol{v}]\right)$, where $\mathcal{S}_i \setminus \bigcup_{\boldsymbol{v} \in V_i}[\boldsymbol{v}]$ is the set of length-$(i-1)$ strings that reach none of the nodes in D.

If an OMTBDD D and its node id set V are given, we can easily answer that a given $\boldsymbol{a} = a_1 a_2 \cdots a_{i-1} \in \mathcal{S}_i$ reaches node $\boldsymbol{v} \in V_i$ or does not reach any node $\boldsymbol{v} \in V_i$ from the path in D for assignment $x_1 = a_1, x_2 = a_2, \ldots, x_{i-1} = a_{i-1}$. Then, can we answer the same question when D is not given but we can ask membership queries for D? If D is reduced, the answer is yes. The reason is as follows.

Consider the OMTBDD $D_{\boldsymbol{v}}$ which is the subgraph reachable from node $\boldsymbol{v} \in V_i$. OMTBDD $D_{\boldsymbol{v}}$ can be seen as a K-valued function over $(x_i, x_{i+1}, \ldots, x_m) \in \{0,1\}^{m-i+1}$, that is, $D_{\boldsymbol{v}}(x_i, x_{i+1}, \ldots, x_m) = D(v_1, v_2, \ldots, v_{i-1}, x_i, x_{i+1}, \ldots, x_m)$ for $\boldsymbol{v} = v_1 v_2 \cdots v_{i-1}$. Let node $\boldsymbol{v}_a \in V$ be the node in which the a-labeled edge outgoing from node $\boldsymbol{v}$ comes. Then, $D_{\boldsymbol{v}}(0, x_{i+1}, \ldots, x_m)$ and $D_{\boldsymbol{v}}(1, x_{i+1}, \ldots, x_m)$ are functions represented by $D_{\boldsymbol{v}_0}$ and $D_{\boldsymbol{v}_1}$, respectively. If D is reduced,

5

$D_{\boldsymbol{v}}(0, x_{i+1}, \ldots, x_m)$ and $D_{\boldsymbol{v}}(1, x_{i+1}, \ldots, x_m)$ must be different because, otherwise, the further reduction is possible; node $\boldsymbol{v}$ can be removed, and its incoming edge can directly come in node $\boldsymbol{v}_0$ (or $\boldsymbol{v}_1$). Thus, there must be a string $a_{i+1}a_{i+2}\cdots a_m$ for which $D_{\boldsymbol{v}}(0, a_{i+1}, \ldots, a_m) \neq D_{\boldsymbol{v}}(1, a_{i+1}, \ldots, a_m)$. Let $\boldsymbol{r}^{(\boldsymbol{v})}$ be one of such strings $0a_{i+1}a_{i+2}\cdots a_m$ and $1a_{i+1}a_{i+2}\cdots a_m$, and let $\dot{\boldsymbol{r}}^{(\boldsymbol{v})}$ denote the string that is made by flipping the first bit of $\boldsymbol{r}^{(\boldsymbol{v})}$, that is, the other one of them. If $\boldsymbol{a} \in \mathcal{S}_i \setminus \bigcup_{\boldsymbol{v} \in V_i}[\boldsymbol{v}]$, then $D(\boldsymbol{a}\boldsymbol{r}^{(\boldsymbol{v})}) = D(\boldsymbol{a}\dot{\boldsymbol{r}}^{(\boldsymbol{v})})$ holds for all $\boldsymbol{v} \in V_i$, thus $D(\boldsymbol{a}\boldsymbol{r}^{(\boldsymbol{v})}) \neq D_{\boldsymbol{v}}(\boldsymbol{r}^{(\boldsymbol{v})})$ or $D(\boldsymbol{a}\dot{\boldsymbol{r}}^{(\boldsymbol{v})}) \neq D_{\boldsymbol{v}}(\dot{\boldsymbol{r}}^{(\boldsymbol{v})})$ holds for all $\boldsymbol{v} \in V_i$. From the above fact,

$$\boldsymbol{a} \in \mathcal{S}_i \setminus \bigcup_{\boldsymbol{v} \in V_i} [\boldsymbol{v}] \Leftrightarrow D(\boldsymbol{a}\boldsymbol{r}^{(\boldsymbol{v})}) \neq D(\boldsymbol{v}\boldsymbol{r}^{(\boldsymbol{v})}) \text{ or } D(\boldsymbol{a}\dot{\boldsymbol{r}}^{(\boldsymbol{v})}) \neq D(\boldsymbol{v}\dot{\boldsymbol{r}}^{(\boldsymbol{v})}) \text{ for all } \boldsymbol{v} \in V_i$$

(1)

holds. This means that, if we know $\boldsymbol{r}^{(\boldsymbol{v})}, D(\boldsymbol{v}\boldsymbol{r}^{(\boldsymbol{v})})$ and $D(\boldsymbol{v}\dot{\boldsymbol{r}}^{(\boldsymbol{v})})$ for all $\boldsymbol{v} \in V_i$, we can check whether $\boldsymbol{a} \in \mathcal{S}_i \setminus \bigcup_{\boldsymbol{v} \in V_i}[\boldsymbol{v}]$ holds or not by asking membership queries for $\boldsymbol{a}\boldsymbol{r}^{(v)}$ and $\boldsymbol{a}\dot{\boldsymbol{r}}^{(v)}$ for all $\boldsymbol{v} \in V_i$. Note that $D(\boldsymbol{a}\boldsymbol{r}^{(v)}) = D(\boldsymbol{v}\boldsymbol{r}^{(v)})$ and $D(\boldsymbol{a}\dot{\boldsymbol{r}}^{(v)}) = D(\boldsymbol{v}\dot{\boldsymbol{r}}^{(v)})$ might hold for $\boldsymbol{a} \in [\boldsymbol{v}']$ with $\boldsymbol{v}' \overset{D}{\neq} \boldsymbol{v}$, and $\boldsymbol{r}^{(\boldsymbol{v}')}$ for $\boldsymbol{v}' \overset{D}{\neq} \boldsymbol{v}$ can be equal to $\boldsymbol{r}^{(\boldsymbol{v})}$. Even though $D(\boldsymbol{a}\boldsymbol{r}^{(v)}) = D(\boldsymbol{v}\boldsymbol{r}^{(v)})$ and $D(\boldsymbol{a}\dot{\boldsymbol{r}}^{(v)}) = D(\boldsymbol{v}\dot{\boldsymbol{r}}^{(v)})$ holds for $\boldsymbol{a} \in [\boldsymbol{v}']$ with $\boldsymbol{v}' \overset{D}{\neq} \boldsymbol{v}$, we can know whether $\boldsymbol{a} \in [\boldsymbol{v}]$ or not by asking membership queries if D is reduced. If D is reduced, then for any two nodes $\boldsymbol{v}, \boldsymbol{v}' \in V_i$, there is a string $a_i a_{i+1}\cdots a_m$ such that $D_{\boldsymbol{v}}(a_i, a_{i+1}, \ldots, a_m) \neq D_{\boldsymbol{v}'}(a_i, a_{i+1}, \ldots, a_m)$ because, if not, $D_{\boldsymbol{v}} = D_{\boldsymbol{v}'}$ holds, then further reduction is possible; node $\boldsymbol{v}'$ can be removed and all its incoming edges can come in node $\boldsymbol{v}$. Let $r^{(\boldsymbol{v},\boldsymbol{v}')}$ denote the string $a_i a_{i+1}\cdots a_m$ with $D_{\boldsymbol{v}}(a_i, a_{i+1}, \ldots, a_m) \neq D_{\boldsymbol{v}'}(a_i, a_{i+1}, \ldots, a_m)$. Then, we can check whether $\boldsymbol{a} \in [\boldsymbol{v}]$ or not by asking membership queries at $\boldsymbol{a}\boldsymbol{r}^{(\boldsymbol{v},\boldsymbol{v}')}$ for all $\boldsymbol{v}' \overset{D}{\neq} \boldsymbol{v}$ that satisfies $D(\boldsymbol{v}'\boldsymbol{r}^{(v)}) = D(\boldsymbol{v}\boldsymbol{r}^{(v)})$ and $D(\boldsymbol{v}'\dot{\boldsymbol{r}}^{(v)}) = D(\boldsymbol{v}\dot{\boldsymbol{r}}^{(v)})$.

From the above discussion, we can construct *node classification trees* T_i ($i = 1, \ldots, m$) that classifies $\boldsymbol{a} \in \mathcal{S}_i$ into $\boldsymbol{v} \in V_i$ or μ, where μ means $\boldsymbol{a} \in \mathcal{S}_i \setminus \bigcup_{\boldsymbol{v} \in V_i}[\boldsymbol{v}]$. The leftmost figure in Figure 1 is the form of a node classification tree T_i. It is a rooted tree and composed of two types of internal nodes and leaf nodes. One type of internal nodes is a *twin-test node* and the other type is a *single-test node*. A twin-test node has label $\boldsymbol{r}^{(\boldsymbol{v})}$ for some $\boldsymbol{v} \in V_i$ and two membership queries for $\boldsymbol{a}\boldsymbol{r}^{(\boldsymbol{v})}$ and $\boldsymbol{a}\dot{\boldsymbol{r}}^{(\boldsymbol{v})}$ is asked to classify string $\boldsymbol{a} \in \mathcal{S}_i$. Each twin-test node has two outgoing edges, one is labeled $(D(\boldsymbol{v}\boldsymbol{r}^{(\boldsymbol{v})}), D(\boldsymbol{v}\dot{\boldsymbol{r}}^{(\boldsymbol{v})}))$ and coming in a single-test node or a leaf, and the other
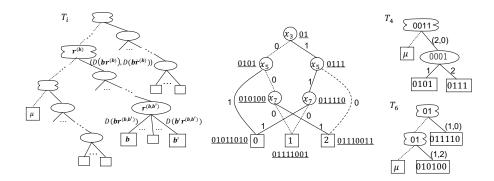
6

Figure 1: The form of a node classification tree (left) and its instance (right) for an example OMTBDD with the access strings of nodes (center).

is non-labeled and coming in a twin-test node or a leaf labeled μ. A single-test node has label $\boldsymbol{r}^{(v,v')}$ for some $\boldsymbol{v}, \boldsymbol{v'} \in V_i$ and one membership query for $\boldsymbol{ar}^{(v,v')}$ is asked to classify string $\boldsymbol{a} \in \mathcal{S}_i$. Each single-test node has at most K outgoing edges labeled $\ell \in \{0, 1, \ldots, K-1\}$ that comes in a single-test node or a leaf labeled some $\boldsymbol{v''} \in V_i$. If the edge comes in a leaf labeled $\boldsymbol{v''}$, the label of the edge must be $D(\boldsymbol{v''r}^{(v,v')})$. Tree T_i has at most $|V_i|+1$ leaves and each of them is labeled $\boldsymbol{v} \in V_i$ or μ. Distinct leaves must have different labels. If a μ-labeled leaf exists, all the nodes on the path from the root to the μ-labeled leaf must be twin-test nodes and all the twin-test nodes must appear on the path, which guarantees the satisfaction of the righthand side condition of (1).

Classification of $\boldsymbol{a} \in \mathcal{S}_i$ into $V_i \cup \{\mu\}$ can be done by using the node classification tree T_i as follows. Start from the root. At a twin-test node labeled $\boldsymbol{r}^{(v)}$, ask two membership queries for $\boldsymbol{ar}^{(v)}$ and $\boldsymbol{a\dot{r}}^{(v)}$, and select the edge labeled $(D(\boldsymbol{ar}^{(v)}), D(\boldsymbol{a\dot{r}}^{(v)}))$ if such labeled edge exists, otherwise select the non-labeled edge. At a single-test node labeled $\boldsymbol{r}^{(v,v')}$, ask one membership query for $\boldsymbol{ar}^{(v,v')}$, and select the edge labeled $D(\boldsymbol{ar}^{(v,v')})$. Label of $\boldsymbol{a}$ classified by T_i, denoted as $T_i(\boldsymbol{a})$, is the label of the leaf that is reached finally repeating the above operations.

Example 1. *Consider the OMTBDD D shown in the center of Figure 1. The underlined string beside each node is its access string in some node id set V of D. Node classification trees T_4 and T_6 of this OMTBDD are shown in the right of the figure. T_4 is composed of one twin-test node, one single-test node and three leaves including μ-labeled leaf. T_6 is composed of two*

7

twin-test nodes and three leaves. String $1100 \in \mathcal{S}_4$ *reaches node* 0101 *in* D. *Since* $D(11000011) = 2$, $D(11001011) = 0$ *and* $D(11000001) = 1$, *it is classified into* 0101 *by* T_4, *which coincides with the reached node in* D *by the assignment* $(x_1, x_2, x_3, x_4) = (1, 1, 0, 0)$. *String* $101001 \in \mathcal{S}_6$ *does not reach any nodes in* D. *Since* $D(10100101) = D(10100111) = 2$, *it is classified into* μ *by* T_6, *which also coincides with nonexistence of nodes reached by the assignment* $(x_1, x_2, x_3, x_4, x_5, x_6) = (1, 0, 1, 0, 0, 1)$.

Remark 1. *Node classification trees for an OMTBDD are different from classification trees [4] for an OBDD in edge labels. In a classification tree* T_i *for an OBDD, the label of an edge outgoing from a twin-test node labeled* $\boldsymbol{r}$ *is* 0 *or* 1, *and* 1-*labeled edge is selected for test string* $\boldsymbol{a} \in \{0, 1\}^{i-1}$ *if and only if* $D(\boldsymbol{a}\boldsymbol{r}) = 1$ *and* $D(\boldsymbol{a}\dot{\boldsymbol{r}}) = 0$. *In the case with an OMTBDD, the number of combinations of different two function values can be more than one, so we adopt label* $(j, j') \in \{0, \ldots, K-1\}^2$ *and no label instead of* 1 *and* 0, *respectively;* (j, j')-*edge is selected for* $\boldsymbol{a}$ *if and only if* $(D(\boldsymbol{a}\boldsymbol{r}), D(\boldsymbol{a}\dot{\boldsymbol{r}})) = (j, j')$. *The label of an edge outgoing from a single-test node is naturally extended from* $j \in \{0, 1\}$ *to* $j \in \{0, \ldots, K-1\}$, *and* j-*labeled edge is selected if and only if* $D(\boldsymbol{a}\boldsymbol{r}) = j$.
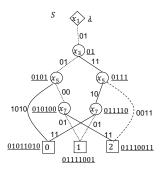
4. Algorithm

We extend algorithm QLearn-π-OBDD [4] for OBDDs to an algorithm for OMTBDDs. Starting from a simple hypothesis OBDD, QLearn-π-OBDD repeatedly asks an equivalence query for the current hypothesis OBDD and updates it using the obtained counterexample from the query until 'YES' is returned to the equivalence query. The basic structure of the extended algorithm is the same as QLearn-π-OBDD. In this section, we explain how to extend QLearn-π-OBDD.

4.1. Hypothesis Data Structure

In QLearn-π-OBDD, the current hypothesis is stored with additional information so as to be updated easily. It keeps the current hypothesis as an *OBDD with access strings (OBDDAS)* and (node) classification trees. In order to deal with more than two function values, node classification trees must be modified, and they are already extended in the previous section. In the followings, we describe extension of OBDDASs for OBDDs to those for OMTBDDs, which does not need modification except the number of sinks.

An *OMTBDD with access strings (OMTBDDAS)* S represents some OMTBDD, which is denoted by $\mathcal{D}(S)$, and stores additional information at edges and nodes. Differences from an OMTBDD are followings:

- Its root is always a node labeled x_1, which may be a dummy node having only one outgoing edge.

- Labels of edges are binary strings instead of 0 or 1. The length of the edge label string between nodes labeled x_i and x_j is $|i - j|$. If an edge goes out from the node labeled x_i to a sink, then its length is $m + 1 - i$. The first bits of two edges going out from the same node must be different.

- Each node has an id string which is a member of some node id set $V(S)$ of the represented OMTBDD $\mathcal{D}(S)$ if the node is not dummy, and λ (null string) if the node is dummy.

The represented OMTBDD $\mathcal{D}(S)$ is obtained from an OMTBDDAS S by removing the dummy root, its outgoing edge and all bits except the first bit from the label strings of all edges (and throwing away the id strings possessed by all nodes). We define $E(S)$ as the set of edges in S, which is represented as a subset of $V(S) \times V(S)$. The label string of edge $(\boldsymbol{u}, \boldsymbol{v}) \in E(S)$ in S is denoted as $l(\boldsymbol{u}, \boldsymbol{v})$.

Example 2. *An example of an OMTBDDAS S is shown in the above. The root of S is a dummy, and the id of each node is the underlined string written beside the node, which is a member of the node id set $V(S)$ of $\mathcal{D}(S)$ except λ, the node id of the dummy node. Here, λ denotes a null string. The OMTBDD $\mathcal{D}(S)$ with node id set $V(S)$ represented by the OBDDAS S is shown in the center of Figure 1.*

As QLearn-π-OBDD does[2], for an unknown target OMTBDD D, our its extension grows hypothesis OMTBDDAS S and node classification trees $T_1, T_2, \ldots, T_m$ so as to keep the following conditions CN, CT and CE. Note

---

[2]Conditions CN, CT and CE are the same as conditions C1, C2 and C3 in [4].

9

that classification by a hypothesis node classification tree is done by membership queries for not $\mathcal{D}(S)$ but D though non-μ leaf labels are strings in $V(S)$. Thus, for any classification tree T_i,

P1. $\forall \boldsymbol{v}_1, \forall \boldsymbol{v}_2 \in \mathrm{nodes}(D)$ with $|\boldsymbol{v}_1| = |\boldsymbol{v}_2| = i$ $[\boldsymbol{v}_1 \overset{D}{=} \boldsymbol{v}_2 \Rightarrow T_i(\boldsymbol{v}_1) = T_i(\boldsymbol{v}_2)]$

holds.

- CN. [Node condition]
    (1)$V(S) \subseteq \mathrm{nodes}(D)$,
    (2)$\forall \boldsymbol{v} \in V(S)_m[\mathcal{D}(S)(\boldsymbol{v}) = D(\boldsymbol{v})]$, and
    (3)$\forall \boldsymbol{v}_1, \forall \boldsymbol{v}_2 \in V(S)[\boldsymbol{v}_1 \neq \boldsymbol{v}_2 \Rightarrow \boldsymbol{v}_1 \overset{D}{\neq} \boldsymbol{v}_2]$.

- CT. [Node classification tree condition]
    (1)$\forall \boldsymbol{v} \in V(S)[T_{|\boldsymbol{v}|}(\boldsymbol{v}) = \boldsymbol{v}]$, and
    (2)$\forall i \in \{1, ..., m\}, \forall \boldsymbol{a} \in \{0,1\}^i[\boldsymbol{a} \notin \mathrm{nodes}(D) \Rightarrow T_i(\boldsymbol{a}) = \mu]$.

- CE. [Edge condition] For all $(\boldsymbol{u}, \boldsymbol{v}) \in E(S)$,
    (1)$T_{|\boldsymbol{v}|}(\boldsymbol{u} \cdot l(\boldsymbol{u}, \boldsymbol{v})) = \boldsymbol{v}$, and
    (2)$|\boldsymbol{u}| < \forall j < |\boldsymbol{v}|, T_j(\boldsymbol{u} \cdot \mathrm{pre}(l(\boldsymbol{u}, \boldsymbol{v}), j - |\boldsymbol{u}|)) = \mu$.

Condition CN is conditions for the node id set $V(S)$: (1) any element of $V(S)$ must be an access string for some node in D, (2) $\mathcal{D}(S)$ must have the same value as D for all the length-m strings in $V(S)$, and (3) any distinct strings in $V(S)$ must reach distinct nodes in D. Condition CT is conditions for hypothesis node classification trees $T_1, T_2, \ldots, T_m$: (1) any node id must be classified into itself, and (2) any non-access-string must be classified into μ. Condition CE is conditions for edges in $\mathcal{D}(S)$: for any edge, (1) the concatenated string of its from-node id and its label string must be classified into its to-node id, and (2) the concatenated string of its from-node id and any prefix of its label string must be classified into μ.

Note that hypothesis node classification trees might be incomplete. As a result, for the label $\boldsymbol{r}$ of some single test node in T_i ($i = 1, \ldots, m$), no edge labeled $D(\boldsymbol{ar})$ might exist for some $\boldsymbol{a} \in \{0,1\}^i$. In such case, we define $T_i(\boldsymbol{a})$ as the label of the last single test node that is reached by repeating edge selection operations at twin-test and single-test nodes starting from the root node.

Algorithm 1 QLearn-OMTBDD()

Output: the reduced OMTBDD of a target function
1: $(\text{Ans}, \boldsymbol{e}') \leftarrow \text{EQ}(\boldsymbol{0})$
2: **if** $\text{Ans} = \text{YES}$ **then return 0**
3: $\ell' \leftarrow D(\boldsymbol{e}')$, $(\text{Ans}, \boldsymbol{e}) \leftarrow \text{EQ}(\boldsymbol{\ell}')$
4: **if** $\text{Ans} = \text{YES}$ **then return** $\boldsymbol{\ell}'$
5: $\ell \leftarrow D(\boldsymbol{e})$, $(S, T_1, T_2, ..., T_m) \leftarrow \text{Initial-Hypothesis}((\boldsymbol{e}', \ell'), \boldsymbol{e})$
6: **loop**
7:    **if** $S(\boldsymbol{e}) = \ell$ **then**
8:       $(\text{Ans}, \boldsymbol{e}) \leftarrow \text{EQ}(\mathcal{D}(S))$
9:       **if** $\text{Ans} = \text{YES}$ **then return** $\mathcal{D}(S)$
10:      $\ell \leftarrow D(\boldsymbol{e})$
11:    $(S, T_1, T_2, ..., T_m) \leftarrow \text{Update-Hypothesis}(S, T_1, T_2, ..., T_m, \boldsymbol{e})$

Algorithm 2 Update-Hypothesis($S, T_1, T_2, ..., T_m, \boldsymbol{e}$)

Output: the updated OMTBDDAS S and node classification trees $T_1, T_2, ..., T_m$
1: $(\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_k) \leftarrow$ the id sequence of nodes in S passed by the counterexample $\boldsymbol{e}$ in the passing order.
2: $i \leftarrow$ the index i s.t. $1 \le i < k$ and $D(\boldsymbol{p}_i\text{suf}(\boldsymbol{e}, m - |\boldsymbol{p}_i|)) \ne D(\boldsymbol{p}_{i+1}\text{suf}(\boldsymbol{e}, m - |\boldsymbol{p}_{i+1}|)) = D(\boldsymbol{p}_k)$
3: **if** $D(\boldsymbol{p}_i l(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})\text{suf}(\boldsymbol{e}, m - |\boldsymbol{p}_{i+1}|)) \ne D(\boldsymbol{p}_k)$ **then**
4:    $(S, T_1, T_2, ..., T_m) \leftarrow \text{NodeSplit}(S, T_1, T_2, ..., T_m, \boldsymbol{e})$
5: **else**
6:    $(S, T_1, T_2, ..., T_m) \leftarrow \text{NewBranchingNode}(S, T_1, T_2, ..., T_m, \boldsymbol{e}, \boldsymbol{p}_i, \boldsymbol{p}_{i+1})$
7: **return** $(S, T_1, T_2, ..., T_m)$
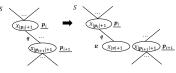

4.2. Pseudocode

Our OMTBDD-version extension of algorithm QLearn-π-OBDD is called QLearn-OMTBDD, and its pseudocodes are shown in Algorithm 1-5. Main extensions are followings:

- It is not easy to find all the sink nodes initially as QLearn-π-OBDD does, so the extended algorithm finds only two sink nodes initially and add necessary sink nodes at Line 10 of AddEdge (Algorithm 5) later.

- When the algorithm adds one single-test node to some node classification tree, it seems inefficient to find and add all its child leaves at that time as QLearn-π-OBDD does, so the extended algorithm adds only two child leaves to the new single-test node at Line 3 of NodeSplit
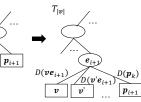

Algorithm 3 NodeSplit$(S, T_1, T_2, ..., T_m, \boldsymbol{e}, \boldsymbol{p}_i, \boldsymbol{p}_{i+1})$

Output: the updated OMTBDDAS S and node classification trees $T_1, T_2, ..., T_m$

1: $\boldsymbol{e}_{i+1} \leftarrow \mathrm{suf}(\boldsymbol{e}, m - |\boldsymbol{p}_{i+1}|)$, $\boldsymbol{v} \leftarrow \boldsymbol{p}_i l(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$

2: Add a node $\boldsymbol{v}$ labeled $x_{|\boldsymbol{v}|+1}$ and an edge $(\boldsymbol{p}_i, \boldsymbol{v})$ labeled $l(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$ to S, and remove the edge $(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$ from S.
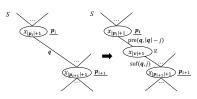
3: Create a tree $T_{|\boldsymbol{v}|}^{\boldsymbol{e}_{i+1}}$ that is composed of a single-test root node labeled $\boldsymbol{e}_{i+1}$ and its two child nodes labeled $\boldsymbol{p}_{i+1}$ and $\boldsymbol{v}$ which are connected by edges labeled $D(\boldsymbol{p}_k)$ and $D(\boldsymbol{v}\boldsymbol{e}_{i+1})$, respectively.

4: $\boldsymbol{t} \leftarrow$ the label of the last twin-test node in $T_{|\boldsymbol{v}|}$ on the path from the root to the $\boldsymbol{p}_{i+1}$-labeled leaf.

5: Add two new edges outgoing from $\boldsymbol{v}$ by executing AddEdge$(\boldsymbol{v}, \boldsymbol{t})$ and AddEdge$(\boldsymbol{v}, \dot{\boldsymbol{t}})$, where $\dot{\boldsymbol{t}}$ denotes the string obtained by flipping the first bit of $\boldsymbol{t}$.

6: **for** all $\boldsymbol{v}_1 \in V(S)$ s.t. $(\boldsymbol{v}_1, \boldsymbol{p}_{i+1}) \in E(S)$ **do**

7:     Ask a membership query for $\boldsymbol{v}_1 \boldsymbol{q}' \boldsymbol{e}_{i+1}$ and obtain $D(\boldsymbol{v}_1 \boldsymbol{q}' \boldsymbol{e}_{i+1})$, where $\boldsymbol{q}' = l(\boldsymbol{v}_1, \boldsymbol{p}_{i+1})$.

8:     **if** $T_{|\boldsymbol{v}|}^{\boldsymbol{e}_{i+1}}$ has an edge labeled $D(\boldsymbol{v}_1 l(\boldsymbol{v}_1, \boldsymbol{p}_{i+1}) \boldsymbol{e}_{i+1})$ **then**

9:       **if** $D(\boldsymbol{v}_1 \boldsymbol{q}' \boldsymbol{e}_{i+1}) \neq D(\boldsymbol{p}_k)$ **then**

10:         Remove edge $(\boldsymbol{v}_1, \boldsymbol{p}_{i+1})$ and add edge $(\boldsymbol{v}_1, \boldsymbol{u})$, where $\boldsymbol{u}$ is the label of the leaf with incoming edge labeled $D(\boldsymbol{v}_1 \boldsymbol{q}' \boldsymbol{e}_{i+1})$ in $T_{|\boldsymbol{v}|}^{\boldsymbol{e}_{i+1}}$.

11:     **else**

12:       Add a node $\boldsymbol{v}' = \boldsymbol{v}_1 \boldsymbol{q}'$ labeled $x_{|\boldsymbol{v}'|+1}$ and an edge $(\boldsymbol{v}_1, \boldsymbol{v}')$ labeled $\boldsymbol{q}'$ to S, and remove the edge $(\boldsymbol{v}_1, \boldsymbol{p}_{i+1})$ from S.

13:       Add a leaf node labeled $\boldsymbol{v}'$ to $T_{|\boldsymbol{v}|}^{\boldsymbol{e}_{i+1}}$ as a child of the root with an edge labeled $D(\boldsymbol{v}_1 \boldsymbol{q}' \boldsymbol{e}_{i+1})$.

14:       Add two new edges outgoing from $\boldsymbol{v}'$ by executing AddEdge$(\boldsymbol{v}', \boldsymbol{t})$ and AddEdge$(\boldsymbol{v}', \dot{\boldsymbol{t}})$.

15: Replace the leaf node labeled $\boldsymbol{p}_{i+1}$ of the tree $T_{|\boldsymbol{v}|}$ with the tree $T_{|\boldsymbol{v}|}^{\boldsymbol{e}_{i+1}}$.
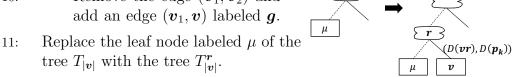

12

Algorithm 4 NewBranchingNode$(S, T_1, T_2, ..., T_m, \boldsymbol{e}, \boldsymbol{p}_i, \boldsymbol{p}_{i+1})$

Output: the updated OMTBDDAS S and node classification trees $T_1, T_2, ..., T_m$

1: $\boldsymbol{e}_i \leftarrow \text{suf}(\boldsymbol{e}, m - |\boldsymbol{p}_i|)$, $\boldsymbol{e}_{i+1} \leftarrow \text{suf}(\boldsymbol{e}, m - |\boldsymbol{p}_{i+1}|)$, $\boldsymbol{q} \leftarrow l(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$, $\boldsymbol{f} \leftarrow \text{pre}(\boldsymbol{e}_i, |\boldsymbol{q}|)$

2: $j \leftarrow j$ with $1 \leq j \leq |\boldsymbol{q}|$ s.t. $D(\boldsymbol{p}_i\text{cro}(\boldsymbol{q}, \boldsymbol{f}, j)\boldsymbol{e}_{i+1}) \neq D(\boldsymbol{p}_i\text{cro}(\boldsymbol{q}, \boldsymbol{f}, j - 1)\boldsymbol{e}_{i+1}) = D(\boldsymbol{p}_k)$

3: $\boldsymbol{v} \leftarrow \boldsymbol{p}_i \cdot \text{pre}(\boldsymbol{q}, |\boldsymbol{q}| - j)$, $\boldsymbol{r} \leftarrow \text{suf}(\boldsymbol{e}, m - |\boldsymbol{v}|)$

4: **if** $j \neq |\boldsymbol{q}|$ **then**

5:     Add a node $\boldsymbol{v}$ labeled $x_{|\boldsymbol{v}|+1}$ and edges $(\boldsymbol{p}_i, \boldsymbol{v})$ labeled $\text{pre}(\boldsymbol{q}, |\boldsymbol{q}| - j)$ and $(\boldsymbol{v}, \boldsymbol{p}_{i+1})$ labeled $\text{suf}(\boldsymbol{q}, j)$ to S, and remove edge $(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$ from S.

6:     Create a tree $T_{|\boldsymbol{v}|}^{\boldsymbol{r}}$ that is composed of a twin-test node labeled $\boldsymbol{r}$ and its two child nodes labeled μ and $\boldsymbol{v}$. Attach label $(D(\boldsymbol{v}\boldsymbol{r}), D(\boldsymbol{p}_k))$ to the edge between the twin-test node and the leaf node labeled $\boldsymbol{v}$.

7:     **for** all $(\boldsymbol{v}_1, \boldsymbol{v}_2) \in V(S)$ with $|\boldsymbol{v}_1| < |\boldsymbol{v}| < |\boldsymbol{v}_2|$ **do**

8:         $\boldsymbol{g} \leftarrow \text{pre}(l(\boldsymbol{v}_1, \boldsymbol{v}_2), |\boldsymbol{v}| - |\boldsymbol{v}_1|)$

9:         **if** $T_{|\boldsymbol{v}|}^{\boldsymbol{r}}(\boldsymbol{v}_1\boldsymbol{g}) = \boldsymbol{v}$ **then**

10:             Remove the edge $(\boldsymbol{v}_1, \boldsymbol{v}_2)$ and add an edge $(\boldsymbol{v}_1, \boldsymbol{v})$ labeled $\boldsymbol{g}$.

11:     Replace the leaf node labeled μ of the tree $T_{|\boldsymbol{v}|}$ with the tree $T_{|\boldsymbol{v}|}^{\boldsymbol{r}}$.

12: **else**

13:     Do nothing.($\boldsymbol{p}_i$ is an access string for a dummy node.)

14: Add a new edge outgoing from $\boldsymbol{v}$ by executing AddEdge$(\boldsymbol{v}, \boldsymbol{r})$.

15: **return** $(S, T_1, T_2, ..., T_m)$

(Algorithm 3), and add other child leaves when they are tried to be accessed (Line 13 of NodeSplit and Line 8 of AddEdge).

- Each child leaf addition to some single-test node in a classification tree is accompanied by discovery and addition of a node in D (Line 12 of NodeSplit and Line 12 of AddEdge). As a result, more than one node in D can be added to the hypothesis OMTBDDAS S during one execution

13

Algorithm 5 AddEdge($\boldsymbol{v}$,$\boldsymbol{t}$)

1: **for** $j = 1$ to $|t|$ **do**
2:     $\boldsymbol{u} \leftarrow T_{|\boldsymbol{v}|+j}(\boldsymbol{v}\mathrm{pre}(\boldsymbol{t}, j))$
3:     **if** $\boldsymbol{u} \neq \mu$ **then**
4:         **if** The node labeled $\boldsymbol{u}$ is a leaf node of $T_{|\boldsymbol{v}|+j}$ **then**
5:             Add an edge $(\boldsymbol{v}, \boldsymbol{u})$ labeled $\mathrm{pre}(\boldsymbol{t}, j)$.
6:         **else**
7:             $\boldsymbol{u}' \leftarrow \boldsymbol{v}\mathrm{pre}(\boldsymbol{t}, j)$
8:             Add a leaf node labeled $\boldsymbol{u}'$ as a child of the node labeled $\boldsymbol{u}$ in $T_{|\boldsymbol{v}|+j}$ with an edge labeled $D(\boldsymbol{u}'\boldsymbol{u})$.
9:             **if** $j = m - |\boldsymbol{v}|$ **then**
10:                 Add a new sink $\boldsymbol{u}'$ labeled $D(\boldsymbol{u}')$ and an edge $(\boldsymbol{v}, \boldsymbol{u}')$ labeled $\mathrm{pre}(\boldsymbol{t}, j)$ to S.
11:             **else**
12:                 Add a new node $\boldsymbol{u}'$ labeled $x_{|\boldsymbol{v}|+j+1}$ and an edge $(\boldsymbol{v}, \boldsymbol{u}')$ labeled $\mathrm{pre}(\boldsymbol{t}, j)$ to S.
13:                 Execute AddEdge($\boldsymbol{u}', \boldsymbol{t}$) and AddEdge($\boldsymbol{u}', \dot{\boldsymbol{t}}$), where $\dot{t}$ denotes the string obtained by flipping the first bit of t.
14:         **return**

---

of Update-Hypothesis, which never happens in QLearn-π-OBDD.

First, QLearn-OMTBDD asks an equivalence query (EQ) for a trivial OMTBDD, denoted by $\boldsymbol{0}$, the OMTBDD being composed of only one sink labeled 0. If 'YES' is returned to the query, the algorithm outputs the hypothesis $\boldsymbol{0}$ and stops. If the answer is 'NO' and counterexample $\boldsymbol{e}'$ is returned to the equivalence query, then the algorithm asks a membership query for assignment $\boldsymbol{e}'$ and obtains $\ell' = D(\boldsymbol{e}')$, where D is the target OMTBDD. Next, the algorithm asks an equivalence query for another trivial OMTBDD $\boldsymbol{\ell}'$ which is composed of one sink labeled ℓ' alone. If 'YES' is returned to the query, the algorithm outputs the hypothesis $\boldsymbol{\ell}'$ and stops. If answer 'NO' and counterexample $\boldsymbol{e}$ is returned, then the algorithm obtains $\ell = D(\boldsymbol{e})$ by asking a membership query. Then, the algorithm makes an initial OMTBD-DAS and node classification trees by algorithm Initial-Hypothesis from the two counterexamples $\boldsymbol{e}'$ and $\boldsymbol{e}$ with $D(\boldsymbol{e}') = \ell'$ and $D(\boldsymbol{e}) = \ell$.

The algorithm Initial-Hypothesis works as follows. Since $\boldsymbol{e}' = \mathrm{cro}(\boldsymbol{e}', \boldsymbol{e}, 0)$, $\boldsymbol{e} = \mathrm{cro}(\boldsymbol{e}', \boldsymbol{e}, m)$ and $D(\boldsymbol{e}') = \ell' \neq D(\boldsymbol{e})$, there exists i with $0 < i \leq m$ such that $D(\mathrm{cro}(\boldsymbol{e}', \boldsymbol{e}, i-1)) = \ell' \neq D(\mathrm{cro}(\boldsymbol{e}', \boldsymbol{e}, i))$, and such an i can be
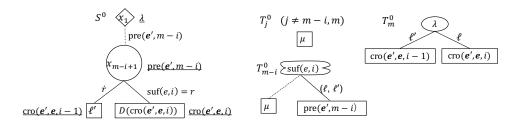
Figure 2: Initial OMTBDDAS S^0 and initial classification trees T_j^0 for $j = 1, ..., m$: T_{m-i}^0 has one twin-test node, T_m^0 has one single-test node, and the other T_j^0s are composed of only one leaf node labeled μ.

found by a binary search using $\lceil \log_2 m \rceil$ membership queries. See Figure 2 for an initial OMTBDDAS S^0 and initial node classification trees T_j^0 for $j = 1, ..., m$ constructed in the procedure. Note that S^0 and $\{T_1^0, ..., T_m^0\}$ satisfy the conditions CN, CT and CE.

Assume that the algorithm has a current OMTBDDAS S and current node classification trees T_i for $i = 1, .., m$. Let $\boldsymbol{e}$ be the last counterexample and let $\ell = D(\boldsymbol{e})$. If $S(\boldsymbol{e}) \neq \ell$, then $\boldsymbol{e}$ is still a counterexample for current hypothesis OMTBDDAS S. Otherwise, the algorithm asks an equivalence query for $\mathcal{D}(S)$ and outputs it if 'YES' is returned. When 'NO' is returned, a new counterexample $\boldsymbol{e}$ can be obtained, and then a membership query for $\boldsymbol{e}$ is asked to obtain $\ell = D(\boldsymbol{e})$. Using the counterexample $\boldsymbol{e}$, it executes the algorithm Update-Hypothesis shown in Algorithm 2. This process is repeated until 'YES' is returned to the equivalence query.

Each execution of the procedure Update-Hypothesis finds at least one node of the target-reduced OMTBDD and updates the current hypothesis. Consider the path in S made by a given counterexample $\boldsymbol{e}$. Assume that there are k nodes on the path and let $\boldsymbol{p}_i$ be the id string of the ith node on the path from the root. Since $\boldsymbol{e}$ is a counterexample for S, the leaf node $\boldsymbol{p}_k$ reached by the path is not correct, that is, $D(\boldsymbol{p}_k) \neq D(\boldsymbol{e})$. Let $\boldsymbol{e}_i = \mathrm{suf}(\boldsymbol{e}, m - |\boldsymbol{p}_i|)$. Since $\boldsymbol{p}_k = \boldsymbol{p}_k \boldsymbol{e}_k$ and $\boldsymbol{e} = \boldsymbol{p}_1 \boldsymbol{e}_1$, there must exist i such that $1 \leq i < k$ and $D(\boldsymbol{p}_i \boldsymbol{e}_i) \neq D(\boldsymbol{p}_{i+1} \boldsymbol{e}_{i+1}) = D(\boldsymbol{p}_k)$. Such i is calculated at Line 2 by using membership queries. For this i, let $\boldsymbol{q}$ denote the label of the edge $(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$, that is, $\boldsymbol{q} = l(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$. There are two cases depending on the value of $D(\boldsymbol{p}_i \boldsymbol{q} \boldsymbol{e}_{i+1})$. When $D(\boldsymbol{p}_i \boldsymbol{q} \boldsymbol{e}_{i+1}) \neq D(\boldsymbol{p}_{i+1} \boldsymbol{e}_{i+1})$, $\boldsymbol{p}_i \boldsymbol{q}$ and $\boldsymbol{p}_{i+1}$ must reach different nodes in D, and this case is dealt with in the algorithm NodeSplit (Algorithm 3), where at least one single-test node is added to one

15

of the classification trees. When $D(\boldsymbol{p}_i\boldsymbol{q}\boldsymbol{e}_{i+1}) = D(\boldsymbol{p}_{i+1}\boldsymbol{e}_{i+1}) = D(\boldsymbol{p}_k)$, there must exist a node between $\boldsymbol{p}_i$ and $\boldsymbol{p}_{i+1}$ from which the path for $\boldsymbol{p}_i\boldsymbol{e}_i$ and the path for $\boldsymbol{p}_i\boldsymbol{q}\boldsymbol{e}_{i+1}$ branches, and this case is dealt with in the algorithm NewBranchingNode (Algorithm 4), where at least one twin-test node is added to one of the classification trees. Both algorithms add a new node $\boldsymbol{v}$ to the current OMTBDDAS, update $T_{|\boldsymbol{v}|}$, change all edges that must enter $\boldsymbol{v}$ and add edges going out from $\boldsymbol{v}$.

4.3. Example of Algorithm Execution

An example of QLearn-OMTBDD execution is shown in Figure 3. The OMTBDDAS and node classification trees $(S, T_1, \dots, T_8)$ constructed by Initial-Hypothesis$((10100100, 1), 01111100))$ is shown in $\boxed{1}$. Since $S(01111100) = 0$, an equivalence query is asked for S at Line 8 in QLearn-OMTBDD and a counterexample 01000101 with $D(01000101) = 2$ is assumed to be obtained. As a sequence of access strings passed by 01000101, $(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3) = (\lambda, 1010, 10100100)$ is obtained at Line 1 in Update-Hypothesis. At Line 2 in Update-Hypothesis, i is set to 1 because $2 = D(01000101) = D(\boldsymbol{p}_1\text{suf}(01000101, 8)) \neq D(\boldsymbol{p}_2\text{suf}(01000101, 4)) = D(10100101) = 1$ holds. Since $1 = D(10100101) = D(\boldsymbol{p}_1 l(\boldsymbol{p}_1, \boldsymbol{p}_2)\text{suf}(01000101, 4)) = D(\boldsymbol{p}_3)$ holds, algorithm NewBranchingNode is executed at Line 6. At Line 2 of NewBranchingNode, j is set to $4 = |\boldsymbol{q}|$ because $2 = D(01000101) = D(\lambda\text{cro}(1010, 0100, 4)0101) \neq D(\lambda\text{cro}(1010, 0100, 3)0101) = D(11000101) = 1$. Thus, Line 13 is executed, and the dummy root becomes a non-dummy root. Since $\boldsymbol{r}$ is set to 01000101 at Line 3, an edge outgoing from the root λ is added by executing algorithm AddEdge$(\lambda, 01000101)$ (Algorithm 5). (See $\boxed{2}$.) In algorithm AddEdge, $T_{|\lambda|+j}(\lambda\text{pre}(01000101, j)) = T_j(\text{pre}(01000101, j)) = \mu$ for all $j = 1, \dots, 7$ and $T_8(01000101) = \lambda$ because no edge outgoing from the root of T_8 is labeled 2. Then, a leaf node labeled 01000101 is added as a child of node labeled λ in T_8 (Line 8 in AddEdge), and a sink labeled 2 and an edge $(\lambda, 01000101)$ labeled 01000101 is added to S (Line 10 in AddEdge).

Algorithm NodeSplit is executed in $\boxed{4}$. For S shown in $\boxed{3}$, a counterexample 01111010 is assumed to be obtained. The sequence of access strings passed by 01111010 is $(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4) = (\lambda, 01, 1010, 10101100)$ and i is set to 2 because $1 = D(01111010) = D(\boldsymbol{p}_2\text{suf}(01111010, 6)) \neq D(\boldsymbol{p}_3\text{suf}(01111010, 4)) = D(10101010) = 0$ at Line 2 in Update-Hypothesis. In this case, $1 = D(01101010) = D(\boldsymbol{p}_2 l(\boldsymbol{p}_2, \boldsymbol{p}_3)\text{suf}(01111010, 4)) \neq D(\boldsymbol{p}_4) = 0$ holds, so algorithm NodeSplit is executed at Line 4. In algorithm NodeSplit, node 0110 is split from node 1010 (Line 2), the leaf node labeled 1010 in T_4 is replaced (Line 15) with
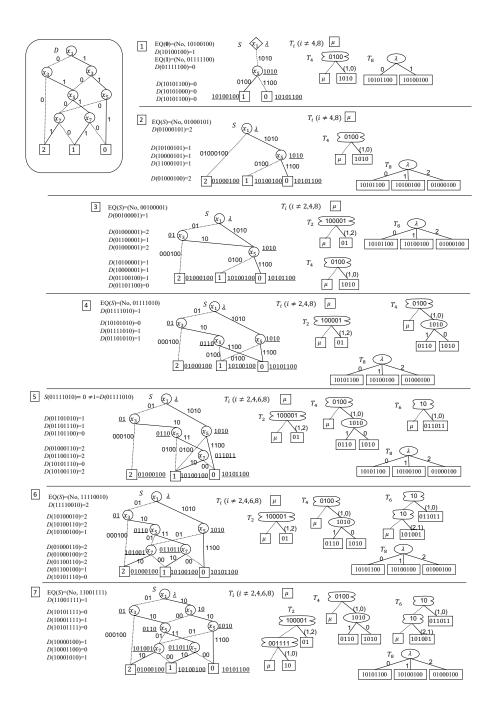
Figure 3: An example of an OMTBDDAS constriction by algorithm QLearn-OMTBDD

a single-test node labeled 1010 that has a child node labeled 0110 connected by an edge labeled 1 and a child node labeled 1010 connected by an edge labeled 0 (Line 3). After this node split, the last counterexample 01111010 is still counterexample for the updated S shown in $\boxed{4}$, that is, $0 = S(01111010) \neq \ell = D(01111010) = 1$, so Lines 8-10 in QLearn-OMTBDD are not executed and Update-Hypothesis is executed for the same (e, ℓ). In the update from $\boxed{5}$ to $\boxed{6}$, node 101001 labeled x_7 is added to S by algorithm NewBranchingNode, in which the edge $(1010, 10100100)$ is replaced with the edge $(1010, 101001)$. This is done during the execution of for-loop of Lines 7-10.

Since the OMTBDD corresponding to the OMTBDDAS S shown in $\boxed{7}$ is equivalent to D, the answer of the equivalence query for S is 'Yes' at Line 9 in QLearn-OMTBDD and $\mathcal{D}(S)(= D)$ is outputted.

4.4. Correctness and Efficiency

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

Proof. See Appendix A. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\square$

Assuming that all 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.

5. Experiments

We conducted experiments to show effectiveness of our algorithm using a synthetic dataset and several benchmark datasets in UCI Machine Learning Repository.

5.1. Synthetic Dataset

We investigated the empirical sample complexity of our algorithm using a synthetic dataset. We randomly generated OMTBDDs with the number of nodes n, the number of variables m, and the number of leaves K for various (n, m, K). Ten OMTBDDs were generated for each (n, m, K) (1) with $n = 1, 2, 4, 8, 16, 32, 64, 128, 256, 512(\times 10^2)$, and fixed $m = 3200$ and $K = 32$, (2) with $m = 1, 2, 4, 8, 16, 32, 64, 128, 256, 512(\times 10^2)$, and fixed

18

$n = 3200$ and $K = 32$, and (3) with $K = 2, 4, 8, 16, 32, 64, 128, 256, 512$, and fixed $n, m = 3200$, using a procedure similar to the OBDD generation procedure [4] (See Appendix B). The number of queries for OMTBDDs of parameter list (n, m, K) are averaged over the ten OMTBDDs.

The results are shown in Figure 4. The tables and line graphs in the left column are the results for membership queries and those in the right column are for equivalence queries. The both axes are log-scaled. Numerical values in the tables are rounded to three significant digits. Theoretical upper bounds $(2n(\lceil \log_2 m \rceil + 3n)$ for membership queries, n for equivalence queries, where n, m are the numbers of nodes and variables, respectively) are also shown for comparison. As for the number of nodes, the theoretical upper bound query numbers for membership and equivalence queries are $O(n^2)$ and $O(n)$, respectively, and those orders of increasing are observable in row (1) of the figure. With respect to the number of variables, $O(\log m)$ and $O(1)$ are the orders of increasing for membership and equivalence queries, respectively, but not only the number of equivalence queries but also that of membership queries looks almost constant in row (2) of the figure. The upper bound on the number of membership queries is $6400 \lceil \log_2 m \rceil + 6 \times 6400^2$, thus additional 6400 queries are required for two times larger number of variables, and 6400 is small compared to 6×6400^2. The numbers of both the queries are not affected by the number of leaves as shown in raw (3) of the figure, though the number of membership queries looks reduced in the case that the number of leaves is very small ($K = 2, 4$). The number of distinct functions becomes larger as the number of leaves increase, which means the complexity of the function class increases. That might be the reason of this phenomenon.

5.2. Benchmark Datasets

One of applications is transformation of learned classifiers to OMTBDDs. If compact OMTBDD representations of learned classifiers are obtained, those are more appropriate for hardware implementation with resource-limited devices and more tractable because various operations between functions are available for OMTBDDs.

We used 12 datasets registered in UCI Machine Learning Repository [8] that are composed of real-valued features only. (See Table 1) The process of our experiment for each dataset is as follows.

1. A tree-based classifier is learned using a given dataset.
2. Branching conditions of component decision trees are reduced by the branching condition sharing algorithm Min_DBN [9]. The tree-based

(1)

#node/10^2	1	2	4	8	16
#query/10^4	0.281	1.41	5.90	23.2	92.6
	32	64	128	256	512
	357	1390	5330	20000	72600

#node/10^2	1	2	4	8	16
#query	53.2	131	281	573	1150
	32	64	128	256	512
	2260	4400	8480	16000	29400

(2)

#variable/10^2	1	2	4	8	16
#query/10^4	357	365	373	377	378
	32	64	128	256	512
	378	382	383	377	386

#variable/10^2	1	2	4	8	16
#query/10	226	234	238	241	245
	32	64	128	256	512
	245	246	247	243	246

(3)

#leaf	2	4	8	16	32
#query/10^4	140	237	311	345	357
		64	128	256	512
		354	347	337	299

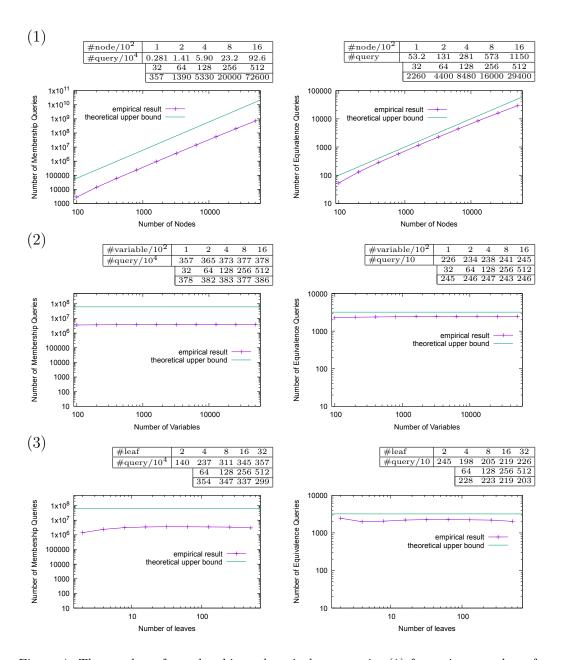| #leaf	2	4	8	16	32
#query/10	245	198	205	219	226
		64	128	256	512
		228	223	219	203

Figure 4: The number of membership and equivalence queries (1) for various number of nodes, (2) for various number of variables, and (3) for various number of leaves.

classifier is converted to a simpler classifier by sharing the branching conditions.

Table 1: Datasets of UCI Machine Learning Repository [8] that is used in our experiment.

dataset	#data	#feature	#class	details
iris	150	4	3	Iris
parkinsons	195	22	2	Parkinsons
breast cancer	569	30	2	Breast Cancer Wisconsin (Diagnostic)
blood	748	4	2	Blood Transfusion Service Center
RNA-Seq PANCAN	801	20531	5	gene expression cancer RNA-Seq
wine quality red	1599	11	11	Wine Quality
wine quality white	4898	11	11	Wine Quality
waveform	5000	40	3	Waveform Database Generator (Version 2)
robot	5456	24	4	Wall-Following Robot Navigation
musk	6598	166	2	Musk (Version 2)
epileptic seizure	11500	178	5	Epileptic Seizure Recognition
magic	19020	10	2	MAGIC Gamma Telescope

3. Each training data is converted to a binary feature data using each distinct branching condition of the converted classifier as a binary feature.

4. The variable ordering of the binary features is decided as follows. Consider a directed graph that is composed of vertices corresponding to the binary features. For each pair of two binary features (x_i, x_j), count the number of occurrences that node labeled x_i is an ancestor of node labeled x_j and the number of occurrences that the opposite relation holds. Define the direction of edges between vertices corresponding to x_i and x_j as that from the vertex corresponding to the feature whose number of ancestor occurrences is more than the other feature's number of ancestor occurrences. Also define the weight of the directed edge as the difference between their number of ancestor occurrences. Remove edges from those with the smallest weights until the topological sorting of the whole vertices is succeeded. Define the binary feature order as the order of corresponding vertices.

5. An OMTBDD is learned by QLearn-OMTBDD from the set of the converted labeled binary feature data using the converted tree-based classifier as the membership oracle and using consistency with all the given data as equivalence to the target function for an equivalence query and returning an inconsistent given data as a counter example. Note that we only use the data whose labels are predicted correctly by the converted tree-based classifier.

As tree-based classifiers, a decision tree and a random forest are used in our experiment. We adopt DecisionTreeClassifier and RandomForest-

Table 2: Number of nodes and accuracy of OMTBDDs learned from decision trees.

dataset	decision tree					OMTBDD	
	#node	(l. share)	accuracy	#DC	#RDC	#node	accuracy
iris	14.6	(9.8)	0.947	6.8	6.8	9.8	0.947
parkinson	25.4	(14.2)	0.856	12.2	11.8	14.2	0.856
breast cancer	37.4	(20.2)	0.924	18.2	18.2	19.8	0.924
blood	320	(162)	0.713	106	72.2	342	0.722
RNA-Seq PANCAN	14.6	(11.8)	0.974	6.8	6.8	11.8	0.974
wine quality red	668	(345)	0.650	300	179	3811	0.652
wine quality white	2088	(1050)	0.604	769	370	29700	0.598
waveform	797	(401)	0.735	397	304	3883	0.743
robot	57.4	(32.7)	0.995	28.0	22.4	49.0	0.996
musk	252	(128)	0.965	126	121	123	0.964
epileptic seizure	3690	(1850)	0.472	1830	1290	722000	0.452
magic	3200	(1600)	0.818	1600	771	49800	0.820

Classifier of scikit-learn version 1.1.dev0. The number of component trees (n_estimators) in RandomForestClassifier is set to 100. Other parameters but n_jobs and random_state are set to defaults in RandomForestClassifier: n_jobs= −1 (the number of jobs to run in parallel is set to the number of processors), random_state= 0 (the seed of randomized selections is set to 0). Parameter random_state (the seed of random permutation of features at each split) is set to 0 in DecisionTreeClassifier and other parameters are set to defaults. We evaluated compactness and accuracy of the learned OMTBDDs by the number of nodes and accuracy for test datasets separated from training datasets by 5-fold crossvalidation. The largest two datasets, epileptic seizure and magic, are too large for our query learning algorithm to learn an OMTBDD from the random forest classifier learned using them, thus we exclude them from our experiments for random forest classifiers.

Results are shown in Table 2 for decision trees and in Table 3 for random forests. In the tables, '#node' is the number of nodes and '(l.share)' means that of its leaf-shared tree-based classifier whose same-labeled leaves share a single leaf. An OMTBDD has only one leaf with the same label, thus comparison in number of nodes to the original tree-based classifier should be done in that of its leaf-shared form. '#DC' is the number of distinct branching conditions in an original tree-based classifier and '#RDC' means that in the classifier reduced by the branching condition sharing algorithm Min_DBN. All the numbers are rounded to three significant digits. For simple classifier problem, in which the original decision tree has less than 30 nodes, their leaf-shared decision trees are already OMTBDDs, and query learning
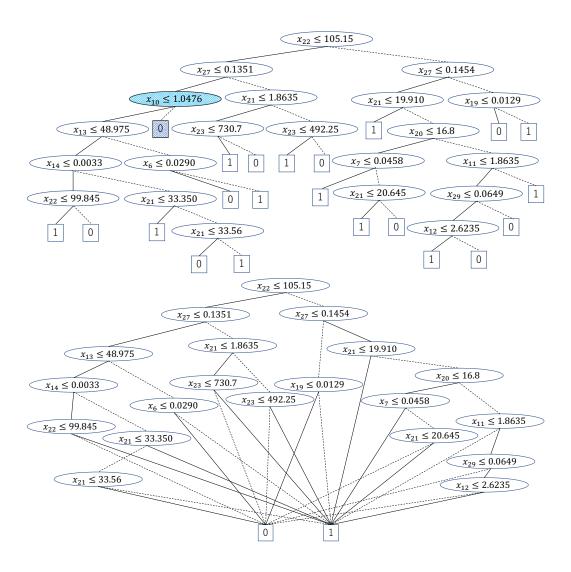
Figure 5: Decision tree for breast cancer dataset and the OMTBDD learned from it.

algorithm learns them exactly. (See the results for iris, parkinson, RNA-Seq PANCAN datasets.) Interesting results are those for breast cancer and musk datasets; the OMTBDD size is smaller than the original leaf-shared decision tree size for those. What happened in one execution of our 5-fold crossvalidation for breast cancer dataset is shown in Figure 5. The corresponding leaf-shared tree of the decision tree in the upper figure is already an OMTBDD, and the OMTBDD in the lower figure is the one learned from

Table 3: Number of nodes and accuracy of OMTBDDs learned from random forests.

dataset	random forest					OMTBDD	
	#node(l. share)		accuracy	#DC	#RDC	#node	accuracy
iris	1430	(965)	0.947	104	43.6	18.2	0.947
parkinson	2660	(1480)	0.892	1010	404	147	0.872
breast cancer	3570	(1940)	0.961	1420	594	203	0.944
blood	23200	(11800)	0.749	322	145	1520	0.723
RNA-Seq PANCAN	3960	(2430)	0.996	1920	1840	3540	0.885
wine quality red	55900	(29000)	0.692	4120	900	271000	0.598
wine quality white	181000	(91600)	0.679	7290	1400	2080000	0.594
waveform	83800	(42200)	0.854	32500	5870	2010000	0.689
robot	25000	(12900)	0.995	9900	3140	1010	0.996
musk	34600	(17500)	0.975	12200	6240	574000	0.908

it. The difference between them is the shaded part. The original decision tree is constructed in top-down manner, and in that manner, $x_{10} \leq 1.0476$ is a good condition to classify. After constructing all the descendants, all the training data classified into 0 by the condition are found to be classified also into 0 in the descendant part. Thus, the condition is not needed. For other larger classifiers except that for epileptic seizure dataset, OMTBDD size is less than 30 times larger than the leaf-shared decision tree size of the original decision tree, and no significant accuracy deterioration is observed. For epileptic seizure dataset, OMTBDD size is 165 times larger and accuracy is also deteriorated. Decision boundary of the decision tree for epileptic seizure dataset is guessed to be complex compared to those for the other datasets in the given variable ordering, and the OMTBDD that is consistent with the given data might not be a good approximation for the original decision tree. As for random forest classifier results, original classifiers' accuracies for all the datasets are improved or the same. Accuracies of OMTBDDs learned from them, however, are the same or deteriorated. Accuracy deterioration is small for the datasets for which size reduction by the learned OMTBDD is succeeded: iris, parkinson, breast cancer, blood, and robot. Especially for parkinson and breast cancer datasets, accuracies of the OMTBDDs learned from the random forest classifiers are better than those learned from the decision tree classifiers. Those OMTBDDs are meaningful in the point that their accuracies are better than the decision trees and their size are smaller than the random forests. For other 5 datasets, accuracies are deteriorated significantly and sizes become 1.5-48 times larger. The reason why such deterioration occurred for the 5 datasets seems the same as the above-mentioned

reason for decision tree accuracy deterioration using epileptic seizure dataset.

6. Conclusions and Future Work

We developed a query learning algorithm QLearn-OMTBDD for OMTB-DDs by extending QLearn-π-OBDD [4] for OBDDs. In our algorithm, the length-$(i-1)$ prefix of an assignment is classified into a node with label x_i or μ (no node corresponds to the assignment prefix) by the node classification tree for length $i-1$ using membership queries, and the fact that the number of possible answers is more than two for each membership query, prevents straightforward extension of the classification trees and their operating procedure. In the experiments using benchmark datasets, we showed possibility of our algorithm's application to a classification problem by constructing compact and accurate OMTBDDs for some datasets. On the other hand, there are other datasets for which learned OMTBDDs have a lot of nodes and low accuracy. we would like to clarify whether there are compact and accurate OMTBDDs for such datasets or not.

Acknowledgments

References

[1] R. E. Bryant, Symbolic boolean manipulation with ordered binary-decision diagrams, ACM Comput. Surv. 24 (1992) 293–318.

[2] M. Fujita, P. C. McGeer, J. C. Y. Yang, Multi-terminal binary decision diagrams: An efficient data structure for matrix representation, Formal Methods in System Design 10 (1997) 149–169.

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

[4] A. Nakamura, An efficient query learning algorithm for ordered binary decision diagrams, Inf. Comput. 201 (2005) 178–198.

[5] M. J. Kearns, U. V. Vazirani, An Introduction to Computational Learning Theory, MIT Press, Cambridge, MA, USA, 1994.

[6] R. Gavaldà, D. Guijarro, Learning ordered binary decision diagrams, in: Algorithmic Learning Theory, 1995, pp. 228–238.

[7] H. Mizumoto, S. Todoroki, Diptarama, R. Yoshinaka, A. Shinohara, An efficient query learning algorithm for zero-suppressed binary decision diagrams, in: Proceedings of the 28th International Conference on Algorithmic Learning Theory, volume 76, 2017, pp. 360–371.

[8] D. Dua, E. Karra Taniskidou, UCI machine learning repository, 2017. URL: `http://archive.ics.uci.edu/ml`.

[9] A. Nakamura, K. Sakurada, An algorithm for reducing the number of distinct branching conditions in a decision forest, in: Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2019, Würzburg, Germany, September 16-20, 2019, Proceedings, Part I, 2019, pp. 578–589.

Appendix A. Proof of Theorem 1

The following proofs of lemma 1, 2 and Theorem 1 for OMTBDDs are extensions of the proofs of Lemma 3, 4 and Theorem 5 for OBDDs in [4]. We write the whole detailed proofs for selfcontainedness..

Lemma 1. *For a reduced OMTBDD D, assume that an OMTBDDAS S and node classification trees T_i $(i = 1, ..., m)$ satisfy conditions CN, CT, and CE. Then, $\mathcal{D}(S) = D$ if the cardinality of $V(S)$ is exactly the number of nodes in D.*

Proof. Let $N(\mathcal{D}(S))$ and $N(D)$ be the set of nodes in $\mathcal{D}(S)$ and D, respectively. Define mapping $M : N(\mathcal{D}(S)) \to N(D)$ that maps node $\boldsymbol{v} \in V(S)$ in $\mathcal{D}(S)$ to the node with access string $\boldsymbol{v}$ in D. In order to prove $\mathcal{D}(S) = D$, we prove $\langle 1 \rangle$ M is well-defined, one-to-one and onto, $\langle 2 \rangle$ M preserves node labels, and $\langle 3 \rangle$ M preserves edge relations and labels. $\langle 1 \rangle$ is proved easily; mapping M is well-defined by CN (1), one-to-one by CN (3), and onto by the assumption that $|V(S)|$ is equal to the number of nodes in D. Both $\mathcal{D}(D)$ and D are OMTBDDs for the same variable ordering, and nodes are mapped to the nodes with the same access strings, so internal node labels are the same. The sink labels are guaranteed to be the same by CN (2). Thus $\langle 2 \rangle$ holds.

26

$\langle 3 \rangle$ is proved as follows. Since $\langle 1 \rangle$ and $\langle 2 \rangle$ hold, the number of non-sink nodes of $\mathcal{D}(S)$ and D are the same, so the number of edges is the same. Thus, it is enough to prove that, for any $\boldsymbol{v}_1, \boldsymbol{v}_2 \in V(S)$, if there exists a b-labeled edge between the nodes with access strings $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ in D, then $(\boldsymbol{v}_1, \boldsymbol{v}_2) \in E(S)$ and the first bit of $l(\boldsymbol{v}_1, \boldsymbol{v}_2)$ is b in S.

Assume that there exists a b-labeled edge between the nodes with access strings $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ in D for $\boldsymbol{v}_1, \boldsymbol{v}_2 \in V(S)$. Let $(\boldsymbol{v}_1, \boldsymbol{v})$ be the edge in S which goes out from node $\boldsymbol{v}_1$ and is labeled by a string with the first bit b. Assume that $|\boldsymbol{v}| < |\boldsymbol{v}_2|$. Since $\boldsymbol{v}_1 \cdot l(\boldsymbol{v}_1, \boldsymbol{v}) \notin \mathrm{nodes}(D)$, $T_{|v|}(\boldsymbol{v}_1 \cdot l(\boldsymbol{v}_1, \boldsymbol{v})) = \mu$ by CT (2), thus $(\boldsymbol{v}_1, \boldsymbol{v}) \notin E(S)$ by CE (1), which is a contradiction. Hence, $|\boldsymbol{v}| \geq |\boldsymbol{v}_2|$. Assume that $|\boldsymbol{v}| > |\boldsymbol{v}_2|$. Since $\boldsymbol{v}_1 \cdot \mathrm{pre}(l(\boldsymbol{v}_1, \boldsymbol{v}), |\boldsymbol{v}_2| - |\boldsymbol{v}_1|) \stackrel{D}{=} \boldsymbol{v}_2$, $T_{|\boldsymbol{v}_2|}(\boldsymbol{v}_1 \cdot \mathrm{pre}(l(\boldsymbol{v}_1, \boldsymbol{v}), |\boldsymbol{v}_2| - |\boldsymbol{v}_1|)) = \boldsymbol{v}_2$ by CT (1) and P1, which contradicts CE (2). Therefore, $|\boldsymbol{v}| = |\boldsymbol{v}_2|$. Since $\boldsymbol{v}_1 \cdot l(\boldsymbol{v}_1, \boldsymbol{v}) \stackrel{D}{=} \boldsymbol{v}_2$, $T_{|\boldsymbol{v}_2|}(\boldsymbol{v}_1 \cdot l(\boldsymbol{v}_1, \boldsymbol{v})) = \boldsymbol{v}_2$ by CT (1) and P1. On the other hand, $T_{|\boldsymbol{v}_2|}(\boldsymbol{v}_1 \cdot l(\boldsymbol{v}_1, \boldsymbol{v})) = \boldsymbol{v}$ by CE (1). Hence, $\boldsymbol{v} = \boldsymbol{v}_2$. This means $(\boldsymbol{v}_1, \boldsymbol{v}_2) \in E(S)$ and the first bit of $l(\boldsymbol{v}_1, \boldsymbol{v}_2)$ is b in S. Thus, $\langle 3 \rangle$ holds. $\square$

Lemma 2. *For a target OMTBDD D, assume that an OMTBDDAS S and classification trees T_i for $i = 1, ..., m$ satisfy CN, CT and CE and that S has at least two sinks. Let $\boldsymbol{e}$ be a counterexample of D for $\mathcal{D}(S)$. Let $(S', T'_1, \ldots, T'_m)$ denote the output of algorithm Update-Hypothesis for $(S, T_1, ..., T_m, \boldsymbol{e})$. Then, $(S', T'_1, \ldots, T'_m)$ satisfies CN, CT and CE, and $|V(S')| \geq |V(S)| + 1$.*

Proof. First of all, note that, if $T_{|\boldsymbol{v}|}(\boldsymbol{v}) \neq \mu$, then $T_{|\boldsymbol{v}|}(\boldsymbol{v}) \neq \mu$ holds after any update of $T_{|\boldsymbol{v}|}$ done in algorithms NodeSplit and NewBranchingNode. This is because the path from the root to the leaf labeled μ in $T_{|\boldsymbol{v}|}$ contains twin-test nodes only except the μ-labeled leaf node itself, and twin-test nodes are never updated, μ-labeled leaves are never added to single-test nodes and any non-μ-labeled leaf is not replaced with a subtree with μ-labeled leaf in those algorithms.

One execution of Algorithm Update-Hypothesis might add more than one node. First, we prove that satisfaction of two conditions CN and CT is preserved after the first node addition and its accompanying node classification tree updates through Claim 1. Then, any following one node addition and its accompanying node classification tree updates are also proved to preserve the property of satisfying CN and CT. Finally, we prove that satisfaction of CE is guaranteed after all the additions and their accompanying node classification tree updates.

Claim 1. *Let $(\tilde{S}, \tilde{T}_1, \ldots, \tilde{T}_m)$ be the OMTBDDAS and the node classification trees right after the first addition of node $\boldsymbol{v}$ and its accompanying update of $T_{|\boldsymbol{v}|}$ in algorithm Update-Hypothesis for $(S, T_1, ..., T_m, \boldsymbol{e})$. Then, CN and CT are satisfied by $(\tilde{S}, \tilde{T}_1, \ldots, \tilde{T}_m)$.*

Proof of Claim 1. In NodeSplit, the first new node $\boldsymbol{v} = \boldsymbol{p}_i l(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$ is added to S at Line 2. For this $\boldsymbol{v}$, $\boldsymbol{v} \in \mathrm{nodes}(D)$ is implied by CT (2) because $T_{|\boldsymbol{v}|}(\boldsymbol{v}) = T_{|\boldsymbol{v}|}(\boldsymbol{p}_i l(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})) = \boldsymbol{p}_{i+1} \neq \mu$ holds by CE (1), thus CN (1) holds for $\tilde{S}$. Node $\boldsymbol{v}$ cannot be a sink because, if so, $D(\boldsymbol{v}) = D(\boldsymbol{p}_{k-1} l(\boldsymbol{p}_{k-1}, \boldsymbol{p}_k)) = D(\boldsymbol{p}_k) \neq D(\boldsymbol{e})$ by CE (1), which contradicts the fact that NodeSplit is executed. Therefore, $V_m(\tilde{S}) = V_m(S)$ and thus CN (2) holds for $\tilde{S}$ by the assumption for S. We can show $\boldsymbol{v} \overset{D}{\neq} \boldsymbol{v}'$ for any $\boldsymbol{v}' \in V(S)$ as follows. It is trivial if $|\boldsymbol{v}| \neq |\boldsymbol{v}'|$, so assume that $\boldsymbol{v}' \in V_{|\boldsymbol{v}|}(S)$. If $\boldsymbol{v}' \neq \boldsymbol{p}_{i+1}$, $\boldsymbol{v} \overset{D}{\neq} \boldsymbol{v}'$ because $T_{|\boldsymbol{v}|}(\boldsymbol{v}) = \boldsymbol{p}_{i+1} \neq \boldsymbol{v}'$, therefore $D(\boldsymbol{v}\boldsymbol{t}) \neq D(\boldsymbol{v}'\boldsymbol{t})$ holds for $\boldsymbol{t} = \boldsymbol{r}$ or $\boldsymbol{t} = \dot{\boldsymbol{r}}$, where $\boldsymbol{r}$ is the label of the least common ancestor of $\boldsymbol{v}'$-labeled and $\boldsymbol{p}_{i+1}$-labeled leaves in $T_{|\boldsymbol{v}|}$. For $\boldsymbol{v}' = \boldsymbol{p}_{i+1}$, $D(\boldsymbol{v}\boldsymbol{e}_{i+1}) \neq D(\boldsymbol{p}_{i+1}\boldsymbol{e}_{i+1})$ holds, therefore $\boldsymbol{v} \overset{D}{\neq} \boldsymbol{p}_{i+1}$ holds. Thus, CN (3) also holds for $\tilde{S}$. Therefore, CN is satisfied by $(\tilde{S}, \tilde{T}_1, \ldots, \tilde{T}_m)$. As for node classification trees, only $\boldsymbol{p}_{i+1}$-labeled leaf is replaced with $\boldsymbol{e}_{i+1}$-labeled single-test node with children labeled $\boldsymbol{v}$ and $\boldsymbol{p}_{i+1}$, Thus $\tilde{T}_{|\boldsymbol{v}|}(\boldsymbol{u}) = \boldsymbol{u}$ holds for all $\boldsymbol{u} \in V_{|\boldsymbol{v}|}(\tilde{S})$. Since $\{\boldsymbol{a} \mid T_i(\boldsymbol{a}) = \mu\} = \{\boldsymbol{a} \mid \tilde{T}_i(\boldsymbol{a}) = \mu\}$, CT (2) still holds for $\tilde{T}_i$ $(i = 1, 2, \ldots, m)$. Thus CT is also satisfied by $(\tilde{S}, \tilde{T}_1, \ldots, \tilde{T}_m)$.

In algorithm NewBranchingNode, the first new node $\boldsymbol{v}$ is added to S at Line 5 or 13. In both the cases, $\boldsymbol{v} \in \mathrm{nodes}(D)$ because $D(\boldsymbol{v}\boldsymbol{r}) \neq D(\boldsymbol{v}\dot{\boldsymbol{r}})$ holds for $\boldsymbol{r} = \mathrm{suf}(\boldsymbol{e}, m - |\boldsymbol{v}|)$, which is guaranteed from the selection of j at Line 2, thus CN (1) holds for $\tilde{S}$. Since $\boldsymbol{v}$ cannot be a sink because $j \geq 1$, CN (2) still holds for $\tilde{S}$. By CE (2), $T_{|\boldsymbol{v}|}(\boldsymbol{v}) = \mu$ holds, therefore $\boldsymbol{v} \overset{D}{\neq} \boldsymbol{v}'$ for any $\boldsymbol{v}' \in V(S)$ because $D(\boldsymbol{v}\boldsymbol{r}) \neq D(\boldsymbol{v}'\boldsymbol{r})$ or $D(\boldsymbol{v}\dot{\boldsymbol{r}}) \neq D(\boldsymbol{v}'\dot{\boldsymbol{r}})$ holds for label $\boldsymbol{r}$ of the least common ancestor of nodes labeled μ and $\boldsymbol{v}'$. Thus CN (3) holds for $\tilde{S}$. Trivially, $\tilde{T}_{|\boldsymbol{v}|}(\boldsymbol{v}) = \boldsymbol{v}$ holds because $T_{|\boldsymbol{v}|}(\boldsymbol{v}) = \mu$ is guaranteed by CE (2) and the μ-labeled leaf in $T_{|\boldsymbol{v}|}$ is replaced with a twin-test node labeled $\boldsymbol{r}$ having outgoing edge labeled $(D(\boldsymbol{v}\boldsymbol{r}), D(\boldsymbol{v}\dot{\boldsymbol{r}}))$ to the leaf labeled $\boldsymbol{v}$. Since no other update is done to $T_{|\boldsymbol{v}|}$, so $\tilde{T}_{|\boldsymbol{v}|}(\boldsymbol{v}') = T_{|\boldsymbol{v}|}(\boldsymbol{v}') = \boldsymbol{v}'$ holds for $\boldsymbol{v}' \in V_{|\boldsymbol{v}|}(S)$. Therefore CT (1) holds for $\tilde{T}_1, \tilde{T}_2, \ldots, \tilde{T}_m$. $\tilde{T}_{|\boldsymbol{a}|}(\boldsymbol{a}) = T(\boldsymbol{a})$ holds for all $\boldsymbol{a} \in \{0, 1\}^i, i = 1, \ldots, m$ except for some $\boldsymbol{a} \in \{0, 1\}^{|\boldsymbol{v}|}$ satisfying $\mu = T_{|\boldsymbol{v}|}(\boldsymbol{a}) \neq \tilde{T}_{|\boldsymbol{v}|}(\boldsymbol{a}) = \boldsymbol{v}$, which belongs to $\mathrm{nodes}(D)$ because $D(\boldsymbol{a}\boldsymbol{r}) \neq D(\boldsymbol{a}\dot{\boldsymbol{r}})$

holds for the label $\boldsymbol{r}$ of the twin-test node on the path from the root to the leaf labeled $\boldsymbol{v}$ in $\tilde{T}_{|\boldsymbol{v}|}$. Thus CT (2) still holds for $\tilde{T}_{|\boldsymbol{v}|}$. Therefore CT holds for $(\tilde{S}, \tilde{T}_1, \ldots, \tilde{T}_m)$. $\hfill\square$

Claim 2. *Assume that at least one node addition and its accompanying node classification tree update has been already done for the current OMTBD-DAS and classification trees $(\tilde{S}, \tilde{T}_1, \ldots, \tilde{T}_m)$ in algorithm Update-Hypothesis for $(S, T_1, ..., T_m, \boldsymbol{e})$. Also assume that $(\tilde{S}, \tilde{T}_1, \ldots, \tilde{T}_m)$ satisfies CN and CT. Let $(\tilde{S}', \tilde{T}_1', \ldots, \tilde{T}_m')$ be the OMTBDDAS and node classification trees after one node addition and its accompanying update of a node classification tree. Then, CN and CT are satisfied by $(\tilde{S}', \tilde{T}_1', \ldots, \tilde{T}_m')$.*

Proof of Claim 2. Consider the node addition and the accompanying node classification tree update done at Lines 12 and 13 in NodeSplit. The added node $\boldsymbol{v}'$ is $\boldsymbol{v}_1 l(\boldsymbol{v}_1, \boldsymbol{p}_{i+1})$ for some $\boldsymbol{v}_1 \in V(S)$. Then, $\boldsymbol{v}' \in \text{nodes}(D)$ by CT (2) because $T(\boldsymbol{v}') = T(\boldsymbol{v}_1 l(\boldsymbol{v}_1, \boldsymbol{p}_{i+1})) = \boldsymbol{p}_{i+1} \neq \mu$ by CE (1). Thus, CN (1) holds for $\tilde{S}'$. CN (2) holds for $\tilde{S}'$ since node $\boldsymbol{v}'$ cannot be a sink because $|\boldsymbol{v}'| = |\boldsymbol{v}|$ and $\boldsymbol{v}$ is not a sink. Furthermore, $\boldsymbol{v}' \overset{D}{\neq} \boldsymbol{u}$ holds for any $\boldsymbol{u} \in V(\tilde{S})$ because $\tilde{T}_{|\boldsymbol{v}'|}(\boldsymbol{v}') = \boldsymbol{e}_{i+1}$ (no $D(\boldsymbol{v}'\boldsymbol{e}_{i+1})$-labeled edge outgoing from the single-test node labeled $\boldsymbol{e}_{i+1}$ exists), which means that, for any label $\boldsymbol{u}(\neq \mu)$ of a leaf, label $\boldsymbol{r}$ (or $\dot{\boldsymbol{r}}$) of the least common ancestor of $\boldsymbol{u}$-labeled leaf and $\boldsymbol{e}_{i+1}$-labeled single test node satisfies $D(\boldsymbol{v}'\boldsymbol{r}) \neq D(\boldsymbol{u}\boldsymbol{r})$ (or $D(\boldsymbol{v}'\dot{\boldsymbol{r}}) \neq D(\boldsymbol{u}\dot{\boldsymbol{r}})$). Thus CN (3) is satisfied by $\tilde{S}'$. CT (1) holds for $\tilde{T}_1', \ldots, \tilde{T}_m'$ because $\tilde{T}_{|\boldsymbol{v}'|}$ is updated so as to satisfy $\tilde{T}_{|\boldsymbol{v}'|}'(\boldsymbol{v}') = \boldsymbol{v}'$ by adding a leaf node labeled $\boldsymbol{v}'$ and an edge labeled $D(\boldsymbol{v}')$ from the single test node labeled $\boldsymbol{e}_{i+1}$ to the leaf. Since $\{\boldsymbol{a} \mid \tilde{T}_i(\boldsymbol{a}) = \mu\} = \{\boldsymbol{a} \mid \tilde{T}_i'(\boldsymbol{a}) = \mu\}$, CT (2) still holds for $\tilde{T}_i'$ ($i = 1, \ldots, m$).

Let us consider other node additions and classification tree updates done at Lines 5 and 14 in NodeSplit and at Line 14 in NewBranchingNode, where the operations may be conducted repeatedly in algorithm AddEdge. In AddEdge, a new node $\boldsymbol{u}'$ is added to $\tilde{S}$ at Line 10 or 12. Node $\boldsymbol{u}'$ belongs to $\text{nodes}(D)$ because $\tilde{T}_{|\boldsymbol{u}'|}(\boldsymbol{u}') = \boldsymbol{u} \neq \mu$ (no $D(\boldsymbol{u}'\boldsymbol{u})$-labeled edge outgoing from the single-test node labeled $\boldsymbol{u}$ exists), so $|\boldsymbol{u}'| = m$ or $D(\boldsymbol{u}'\boldsymbol{r}) \neq D(\boldsymbol{u}'\dot{\boldsymbol{r}})$ for label $\boldsymbol{r}$ of a twin-test node on the path from the root to the single-test node labeled $\boldsymbol{u}$. Therefore CN (1) holds for $\tilde{S}'$. In the case that $\boldsymbol{u}'$ is a sink, $\tilde{S}$ is updated so as to satisfy $\mathcal{D}(\tilde{S}')(\boldsymbol{u}') = D(\boldsymbol{u}')$ (Line 10) because node $\boldsymbol{u}'$ is labeled by $D(\boldsymbol{u}')$ and $S(\boldsymbol{v}) = \boldsymbol{v}$ is guaranteed for $\boldsymbol{v}$ satisfying $\boldsymbol{v}\text{pre}(\boldsymbol{t}, |\boldsymbol{u}'| - |\boldsymbol{v}|) = \boldsymbol{u}'$. Thus, CN (2) is satisfied by $\tilde{S}'$. Furthermore,

$\boldsymbol{u}' \overset{D}{\neq} \boldsymbol{v}'$ holds for any $\boldsymbol{v}' \in V(\tilde{S})$ because $\tilde{T}_{|\boldsymbol{u}'|}(\boldsymbol{u}') = \boldsymbol{u}$ and $\boldsymbol{u}$ is the label of a single-test internal node, which means that, for any label $\boldsymbol{v}'(\neq \mu)$ of a leaf, the label $\boldsymbol{r}$ (or $\dot{\boldsymbol{r}}$) of the least common ancestor of $\boldsymbol{v}'$-labeled node and the node labeled $\boldsymbol{u}$ satisfies $D(\boldsymbol{u}'\boldsymbol{r}) \neq D(\boldsymbol{v}'\boldsymbol{r})$ (or $D(\boldsymbol{u}'\dot{\boldsymbol{r}}) \neq D(\boldsymbol{v}'\dot{\boldsymbol{r}})$). Therefore CN (3) is satisfied by $\tilde{S}'$. $\tilde{T}_{|\boldsymbol{u}'|}$ is updated so as to satisfy $\tilde{T}'_{|\boldsymbol{u}'|}(\boldsymbol{u}') = \boldsymbol{u}'$ by adding a leaf node labeled $\boldsymbol{u}'$ and an edge labeled $D(\boldsymbol{u}'\boldsymbol{u})$ from the single-test node labeled $\boldsymbol{u}$ to the leaf. Therefore CT (1) holds for $\tilde{T}'_1, \ldots, \tilde{T}'_m$. Since $\{\boldsymbol{a} \mid \tilde{T}_i(\boldsymbol{a}) = \mu\} = \{\boldsymbol{a} \mid \tilde{T}'_i(\boldsymbol{a}) = \mu\}$, CT (2) still holds for $\tilde{T}'_1, \ldots, \tilde{T}'_m$. $\qquad\square$

We prove that CE holds for $(S', T'_1, \ldots, T'_m)$.

First, consider the NodeSplit case. In NodeSplit, the μ-labeled leaf and the twin-test nodes on the path from the root to the leaf for any T_i ($i = 1, \ldots, m$) does not change, which means that $T'_i(\boldsymbol{u}) = \mu$ if and only if $T_i(\boldsymbol{u}) = \mu$ for all $\boldsymbol{u} \in \{0, 1\}^i, i = 1, \ldots, m$. Therefore, CE (2) holds for $(\boldsymbol{u}, \boldsymbol{v}) \in E(S') \cap E(S)$. Let $\boldsymbol{v}$ be the new node added to S at Line 2. Then, T_i that are changed in NodeSplit are $T_{|\boldsymbol{v}|}$ and T_j for some $j > |\boldsymbol{v}|$ through AddEdge. In $T_{|\boldsymbol{v}|}$, the leaf labeled $\boldsymbol{p}_{i+1}$ is replaced with $T^{\boldsymbol{e}_{i+1}}_{|\boldsymbol{v}|}$ at Line 15. Thus, edges $(\boldsymbol{v}', \boldsymbol{p}_{i+1})$ with $T_{|\boldsymbol{v}|}(\boldsymbol{v}'l(\boldsymbol{v}', \boldsymbol{p}_{i+1})) = \boldsymbol{p}_{i+1}$ and $T'_{|\boldsymbol{v}|}(\boldsymbol{v}'l(\boldsymbol{v}', \boldsymbol{p}_{i+1})) = \boldsymbol{v} \neq \boldsymbol{p}_{i+1}$ must be removed and new edges $(\boldsymbol{v}', \boldsymbol{v})$ must be added. All such edge removals and additions are done at Lines 2, 10 and 12. Thus, all the edges $(\boldsymbol{v}', \boldsymbol{p}_{i+1})$ that do not satisfy CE (1) are removed and all the added edges $(\boldsymbol{v}', \boldsymbol{v})$ satisfy CE (1). Since strings $l(\boldsymbol{v}', \boldsymbol{v})$ for the added edges $(\boldsymbol{v}', \boldsymbol{v})$ are the same as strings $l(\boldsymbol{v}', \boldsymbol{p}_{i+1})$ for the removed edges $(\boldsymbol{v}', \boldsymbol{p}_{i+1})$ and $T'_i = T_i$ for $i < |\boldsymbol{v}|$, CE (2) still holds for new edges $(\boldsymbol{v}', \boldsymbol{v})$. Other new edges $(\boldsymbol{v}', \boldsymbol{u}')$ are added through AddEdge$(\boldsymbol{v}', \boldsymbol{t})$ and those edges are made so as to satisfy CE (1) and CE (2). Thus, CE is satisfied by $(S', T'_1, \ldots, T'_m)$.

Next, consider the NewBranchingNode case. Let $\boldsymbol{v}$ denote the new node added to S at Line 5. Then, T_j that are changed in NewBranchingNode are $T_{|\boldsymbol{v}|}$ and T_j for some $j > |\boldsymbol{v}|$ through AddEdge. In $T_{|\boldsymbol{v}|}$, the leaf labeled μ is replaced with $T^{\boldsymbol{r}}_{|\boldsymbol{v}|}$ at Line 11. Thus, edges $(\boldsymbol{u}, \boldsymbol{v}')$ with $|\boldsymbol{u}| < |\boldsymbol{v}| < |\boldsymbol{v}'|$, $T_{|\boldsymbol{v}|}(\boldsymbol{u}\mathrm{pre}(l(\boldsymbol{u}, \boldsymbol{v}'), |\boldsymbol{v}| - |\boldsymbol{u}|)) = \mu$ and $T'_{|\boldsymbol{v}|}(\boldsymbol{u}\mathrm{pre}(l(\boldsymbol{u}, \boldsymbol{v}'), |\boldsymbol{v}| - |\boldsymbol{u}|)) = \boldsymbol{v}$ must be removed and new edges $(\boldsymbol{u}, \boldsymbol{v})$ must be added. All such edge removals and additions are done at Lines 5 and 10. Thus, all the edges $(\boldsymbol{u}, \boldsymbol{v}')$ that do not satisfy CE (2) are removed and all the added edges $(\boldsymbol{u}, \boldsymbol{v})$ satisfy CE (1). Since strings $l(\boldsymbol{u}, \boldsymbol{v})$ for the added edges $(\boldsymbol{u}, \boldsymbol{v})$ are the same as strings $\mathrm{pre}(l(\boldsymbol{u}, \boldsymbol{v}'), |\boldsymbol{v}| - |\boldsymbol{u}|)$ for the removed edges $(\boldsymbol{u}, \boldsymbol{v}')$ and $T'_j = T_j$ for $j < |\boldsymbol{v}|$, CE (2) still holds for the added edges $(\boldsymbol{u}, \boldsymbol{v})$. Edge $(\boldsymbol{v}, \boldsymbol{p}_{i+1})$

is also added at Line 5. Since $\boldsymbol{v}l(\boldsymbol{v}, \boldsymbol{p}_{i+1})$ in S' is equal to $\boldsymbol{p}_i l(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$ in S and the nodes on the path from the root to $\boldsymbol{p}_{i+1}$-labeled leaf in $T_{|\boldsymbol{p}_{i+1}|}$ does not change in NewBranchingNode, $T'_{|\boldsymbol{p}_{i+1}|}(\boldsymbol{v}l(\boldsymbol{v}, \boldsymbol{p}_{i+1}))$ for S' is equal to $T_{|\boldsymbol{p}_{i+1}|}(\boldsymbol{p}_i l(\boldsymbol{p}_i, \boldsymbol{p}_{i+1}))$ for S, thus $T'_{|\boldsymbol{p}_{i+1}|}(\boldsymbol{v}l(\boldsymbol{v}, \boldsymbol{p}_{i+1})) = \boldsymbol{p}_{i+1}$ by CE (1) for $(S, T_1, \ldots, T_m)$. Since strings $l(\boldsymbol{v}, \boldsymbol{p}_{i+1})$ for the added edges $(\boldsymbol{v}, \boldsymbol{p}_{i+1})$ are the same as strings $\mathrm{suf}(l(\boldsymbol{p}_i, \boldsymbol{p}_{i+1}), |\boldsymbol{p}_{i+1}| - |\boldsymbol{v}|)$ for the removed edges $(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$ and no node on the path from the root to the μ-labeled leaf in T_j for $j > |v|$ changes, CE (2) still holds for the added edges $(\boldsymbol{v}, \boldsymbol{p}_{i+1})$. Other new edges $(\boldsymbol{v}', \boldsymbol{u}')$ for some $|\boldsymbol{v}| \le |\boldsymbol{v}'| < |\boldsymbol{u}'|$ are added at Line 14 through AddEdge$(\boldsymbol{v}, \boldsymbol{r})$ and those edges are made so as to satisfy CE (1) and CE (2). Thus, CE is satisfied by $(S', T'_1, \ldots, T'_m)$. $\qquad\square$

Proof of Theorem 1. We assume that D has at least two sinks because the case with one sink is trivial. First, we prove that QLearn-OMTBDD outputs D with at most n equivalence queries. It is trivial that S^0 and $T_1^0, ..., T_m^0$ satisfy the conditions CN, CT and CE. By Lemma 2, the number of non-dummy nodes of the OMTBDDAS S maintained by the algorithm increases by at least one for every execution of algorithm Update-Hypothesis, which updates S and node classification trees $T_1, ..., T_m$ so as to satisfy CN, CT and CE. Thus, $|V(S)|$ reaches just the number of nodes in D after executing algorithm Update-Hypothesis at most $n-3$ times. Then, Lemma 1 guarantees that $\mathcal{D}(S) = D$. Therefore, the n'th equivalence query by QLearn-OMTBDD for $n' \le n$ is answered with 'YES' because three equivalence queries are asked before the first execution of Update-Hypothesis and at most one equivalence query is asked after each execution of Update-Hypothesis.

Next, we consider the number of membership queries. To construct S^0 and $T_1^0, ..., T_m^0$, 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 per one node addition to S.

Case with NodeSplit execution in Update-Hypothesis

At Line 2 of Update-Hypothesis, at most $\lceil \log_2 m \rceil$ membership queries are asked. At Line 7 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 Line 5 of NodeSplit, at most $4n$ membership queries are asked if no node is added in AddEdge because at most two membership queries are asked for each internal node of $T_1, ..., T_m$ 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 if there is no node addition at Line 12 and in AddEdge. At

most $4n$ additional membership queries are asked per one node addition at Line 12 and at most $2n$ additional membership queries are asked per one node addition in AddEdge. Thus, at most $\lceil \log_2 m \rceil + 5n$ membership queries are asked per one node addition to S.

Case with NewBranchingNode execution in Update-Hypothesis

At Line 2 of Update-Hypothesis and Line 2 of NewBranchingNode, at most $2\lceil \log_2 m \rceil$ membership queries are asked by using a binary search. At Line 9 of NewBranchingNode, at most $4n$ membership queries are asked because at most two membership queries are asked for each edge in S. At Line 14 of NewBranchingNode, at most $2n$ membership queries are asked if no node is added in AddEdge for the reason described above. In this case, the algorithm asks at most $2\lceil \log_2 m \rceil + 6n$ membership queries in total if there is no node addition in AddEdge. Even in the case that there are node additions, at most $2n$ additional membership queries are asked per one node addition to S. Thus, at most $2\lceil \log_2 m \rceil + 6n$ membership queries are totally asked per one node addition to S.

Thus, QLearn-OMTBDD asks at most $2n(\lceil \log_2 m \rceil + 3n)$ membership queries. $\qquad\square$

Appendix B. Random OMTBDD Generation Procedure

In our experiment for synthetic dataset, OMTBDDs are generated by Algorithm 6.

Algorithm 6 OMTBDD Generation Algorithm

Input: m: number of variables, n: number of nodes, K: number of sinks

Output: D: reduced OMTBDD with n nodes and (at most) K sinks for m variables of ordering $x_1 < x_2 < \cdots < x_m$.

1: $n', n'' \leftarrow n$
2: **repeat**
3:     $n' \leftarrow n' + (n - n'')$
4:     Select $n' - K$ variables from $\{x_1, \ldots, x_m\}$ randomly and sort them to $x_{i_1}, x_{i_2}, \ldots, x_{i_{n'-K}}$ so as to satisfy $i_1 \leq i_2 \leq \cdots \leq i_{n'-K}$. Create node v_j labeled x_{i_j} for $j = 1, \ldots, n' - K$ and sink v_j for $j = n' - K + 1, \ldots, n'$.

5:     **for** $j = 1$ **to** $n' - K$ **do**
6:       **if** $j > 1$ and v_j has no incoming edge **then**
7:         Delete node v_j. Proceed to the next j in the FOR-loop.
8:       Set ℓ to 0 or 1 randomly.
9:       **for** $k = \ell$ **to** $(\ell + 1)\%2$ **do**
10:         **if** $\exists v_j$ s.t. n_var$(j) \leq h <$ n_var(n_var$(j))$ and v_h has no incoming edge $\{$n_var$(j) = \min(\{h \mid i_h > i_j\} \cup \{n' - K + 1\})\}$ **then**
11:           Add edge (v_j, v_h) for h that is randomly selected from $\{$n_var$(j), \ldots,$ n_var(n_var$(j))\}$ as the k-labeled outgoing edge of v_j.
12:         **else**
13:           Add edge (v_j, v_h) for h that is randomly selected from $\{$n_var$(j), \ldots, n\}$ as the k-labeled outgoing edge of v_j. Reselect h if $k = (\ell + 1)\%2$ and the same edge as that for $k = \ell$ is selected.
14:     Set the label of sink node v_j $(j = n' - K + 1, \ldots, n')$ to $0, 1, \ldots K - 2$ or $K - 1$ at random such that each sink node label is distinct.
15:     Transform the current OMTBDD into the unique OMTBDD D in the reduced form. Set n'' to the number of nodes in D.
16: **until** $n'' = n$
17: Output D.