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 less than $2n(\lceil \log_2 m \rceil + 2n)$ membership queries by extending the algorithm for ordered binary decision diagrams (OBDDs). The tightness of our upper bounds is checked in our experiments using synthetically generated target OMTBDDs. The possibility of applying our algorithm to classification problems is also indicated in our other experiments using datasets from the 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 manipulate [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 the sinks by selecting the 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 the 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.

Email address: atsu@ist.hokudai.ac.jp (Atsuyoshi Nakamura)

A *multi-terminal binary decision diagram (MTBDD)* [2] is an extension of BDD 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 a 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 the 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 $x_0 = a_0, x_1 = a_1, \ldots, x_{i-1} = a_{i-1}$ into an internal node labeled x_i in the hypothesis OMTBDD or μ in order to judge which node the partial assignment reaches 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. A straightforward modification is to let an internal node have more than two outgoing edges, but it makes the learning algorithm complex. By keeping the binary tree structure, no significant modification is needed for the algorithm extension.

We prove that an arbitrary target reduced OMTBDD can be learned using less than $2n(\lceil \log_2 m \rceil + 2n)$ membership queries and at most n equiv-

alence queries by Algorithm QLearn-OMTBDD, which are the same upper bounds[1] 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 the applicability of our algorithm to classification problems. 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 the 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 is 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 for three datasets but a significant reduction in the number of nodes is achieved for 5 of 10 datasets[2]. As for three of the 5 successfully reduced datasets, the accuracy of the OMTBDD learned from a random forest is better than that of the decision tree. Our results on the classification problem for these benchmark datasets indicate the 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

[1]Strictly speaking, the coefficient of n^2-term is improved from 6 to 4 by more detailed analysis.

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

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 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) [2] 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 the sinks must be consistent with a certain preset order. The variable order is fixed to $x_0, x_1, ..., x_{m-1}$ in this paper.

A K-valued function can be represented by different OMTBDDs, but those OMTBDDs can be reduced to a unique minimal-sized OMTBDD by repeatedly applying the following two reduction rules: (1) eliminate nodes whose 0-labeled and 1-labeled outgoing edges come in the same node, and (2) merge two nodes whose a-labeled outgoing edges come in the same node for each $a = 0, 1$. In this paper, we deal with a problem of function identification as the reduced OMTBDD.

Given an assignment $x_0 = a_0, \ldots, x_{m-1} = a_{m-1}$, 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 $x_0 = a_0, \ldots, x_{m-1} = a_{m-1}$ is also represented by the binary string $a_0 a_1 \ldots a_{m-1}$.

We use bold letters like $\boldsymbol{a}$ and $\boldsymbol{b}$ to represent strings, and the length of any string $\boldsymbol{a}$ is denoted as $|\boldsymbol{a}|$. For any letter a, the fine letter a with a subscript number $i \in \mathbb{N} \cup \{0\}$, which means a_i, denotes the $(i+1)$th letter of the string $\boldsymbol{a}$. 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 $\mathrm{suf}(\boldsymbol{a}, i)$ for $i \in \{0, \ldots, |\boldsymbol{a}|\}$ are the prefix and suffix strings of $\boldsymbol{a}$ with length i, respectively. Note that $\mathrm{pre}(\boldsymbol{a}, 0) = \mathrm{suf}(\boldsymbol{a}, 0) = \lambda$ and $\mathrm{pre}(\boldsymbol{a}, |\boldsymbol{a}|) = \mathrm{suf}(\boldsymbol{a}, |\boldsymbol{a}|) = \boldsymbol{a}$, where λ denotes the null string.

We abuse the notation, and the function represented by an OMTBDD D is also denoted as D, so $D(a_0, a_1, \ldots, a_{m-1})$ is the value of D for the assignment $x_0 = a_0, \ldots, x_{m-1} = a_{m-1}$ and it is also written as $D(\boldsymbol{a})$ for $\boldsymbol{a} = a_0 a_1 \cdots a_{m-1}$.

For node N in OMTBDD D, an *access string of node* N is a string $a_0 a_1 \cdots a_{i-1}$ such that the path on D for assignment $x_0 = a_0, x_1 = a_1, \ldots, x_{i-1} = a_{i-1}$ just reaches node N. The length of the access string is i for nodes labeled x_i for $i = 0, \ldots, m-1$, and m for sinks. Throughout this paper, a string $a_0 a_1 \cdots a_{i-1}$ is said to *reach* node N if and only if the label of N is x_i or N is a sink and $i = m$ for the string length i in addition to N's reachability from the root by selecting the a_j-labeled edge at node labeled x_j. We let $\mathrm{nodes}_i(D)$ denote the set of length-i access strings for nodes in D and let $\mathrm{nodes}(D) = \bigcup_{i=0}^{m} \mathrm{nodes}_i(D)$. Then, for $\boldsymbol{a}, \boldsymbol{b} \in \mathrm{nodes}(D)$, an equivalence relation '$\overset{D}{=}$' is defined as follows:

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

Let $[\boldsymbol{a}]$ denote the equivalence class of $\boldsymbol{a}$. We call a subset V of $\mathrm{nodes}(D)$ 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. Then $V = \bigcup_{i=0}^{m} V_i$, and $\mathrm{nodes}_i(D)$ can be partitioned into $\{[\boldsymbol{v}] \mid \boldsymbol{v} \in V_i\}$ and $\mathrm{nodes}(D)$ 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 $\boldsymbol{e}$ ($f(\boldsymbol{e}) \neq h(\boldsymbol{e})$) are returned. We treat an equivalence query as a function EQ from a hypothesis h to a pair of an answer and a counterexample $(\mathrm{Ans}, \boldsymbol{e})$ for $\mathrm{Ans} \in \{\text{'YES'}, \text{'NO'}\}$, and let $\mathrm{EQ}(h)$ represent $(\mathrm{Ans}, \boldsymbol{e})$. Note that, when $\mathrm{EQ}(h) = (\text{'YES'}, \boldsymbol{e})$, $\boldsymbol{e}$ is indeterminate and must not be used. A *membership query* asks for the value $f(\boldsymbol{a})$ of the target function f for an assignment $\boldsymbol{a}$.

3. Node Classification Trees

For each $i = 0, 1, 2, \ldots, m$, let $\mathcal{S}_i$ be the set of $\{0,1\}$-strings with length i. 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 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 the question of whether a given $\boldsymbol{a} = a_0 a_1 \cdots a_{i-1} \in \mathcal{S}_i$ reaches node $\boldsymbol{v} \in V_i$ or does not reach any node $\boldsymbol{v} \in V_i$, by checking the path in D for assignment $x_0 = a_0, x_1 = a_1, \ldots, x_{i-1} = a_{i-1}$. 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 $(x_i, x_{i+1}, \ldots, x_{m-1}) \in \{0,1\}^{m-i}$, that is, $D_{\boldsymbol{v}}(x_i, x_{i+1}, \ldots, x_{m-1}) = D(v_0, v_1, \ldots, v_{i-1}, x_i, x_{i+1}, \ldots, x_{m-1})$ for $\boldsymbol{v} = v_0 v_1 \cdots v_{i-1}$. Let node $\boldsymbol{v}_b \in V$ be the node in which the b-labeled edge outgoing from node $\boldsymbol{v}$ comes for $b \in \{0,1\}$. Then, $D_{\boldsymbol{v}}(0, x_{i+1}, \ldots, x_{m-1})$ and $D_{\boldsymbol{v}}(1, x_{i+1}, \ldots, x_{m-1})$ are functions represented by $D_{\boldsymbol{v}_0}$ and $D_{\boldsymbol{v}_1}$, respectively. If D is reduced, $D_{\boldsymbol{v}}(0, x_{i+1}, \ldots, x_{m-1})$ and $D_{\boldsymbol{v}}(1, x_{i+1}, \ldots, x_{m-1})$ 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 $a_{i+1} a_{i+2} \cdots a_{m-1}$ for which $D_{\boldsymbol{v}}(0, a_{i+1}, \ldots, a_{m-1}) \neq D_{\boldsymbol{v}}(1, a_{i+1}, \ldots, a_{m-1})$. Let $\boldsymbol{r}^{(\boldsymbol{v})}$ be one of such strings $0 a_{i+1} a_{i+2} \cdots a_{m-1}$ and $1 a_{i+1} a_{i+2} \cdots a_{m-1}$, 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,

$$\boldsymbol{a} \in \mathcal{S}_i \setminus \bigcup_{\boldsymbol{v} \in V_i} [\boldsymbol{v}] \Leftrightarrow D(\boldsymbol{a}\boldsymbol{r}^{(\boldsymbol{v})}) \neq D(\boldsymbol{v}\boldsymbol{r}^{(\boldsymbol{v})}) \text{ or } D(\boldsymbol{a}\dot{\boldsymbol{r}}^{(\boldsymbol{v})}) \neq D(\boldsymbol{v}\dot{\boldsymbol{r}}^{(\boldsymbol{v})}) \text{ for all } \boldsymbol{v} \in V_i \tag{1}$$

holds. This means that, if we know[3] $\boldsymbol{r}^{(\boldsymbol{v})}, D(\boldsymbol{v}\boldsymbol{r}^{(\boldsymbol{v})})$ and $D(\boldsymbol{v}\dot{\boldsymbol{r}}^{(\boldsymbol{v})})$ for all

[3] Using the fact that

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

$\boldsymbol{v} \in V_i$, 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 $\boldsymbol{a}\boldsymbol{r}^{(\boldsymbol{v})}$ and $\boldsymbol{a}\dot{\boldsymbol{r}}^{(\boldsymbol{v})}$ for all $\boldsymbol{v} \in V_i$. Note that $D(\boldsymbol{a}\boldsymbol{r}^{(v)}) = D(\boldsymbol{v}\boldsymbol{r}^{(\boldsymbol{v})})$ and $D(\boldsymbol{a}\dot{\boldsymbol{r}}^{(\boldsymbol{v})}) = D(\boldsymbol{v}\dot{\boldsymbol{r}}^{(\boldsymbol{v})})$ 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 $D(\boldsymbol{a}\boldsymbol{r}^{(\boldsymbol{v})}) = D(\boldsymbol{v}\boldsymbol{r}^{(\boldsymbol{v})})$ and $D(\boldsymbol{a}\dot{\boldsymbol{r}}^{(\boldsymbol{v})}) = D(\boldsymbol{v}\dot{\boldsymbol{r}}^{(\boldsymbol{v})})$ 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 $a_i a_{i+1} \cdots a_{m-1}$ such that $D_{\boldsymbol{v}}(a_i, a_{i+1}, \ldots, a_{m-1}) \neq D_{\boldsymbol{v}'}(a_i, a_{i+1}, \ldots, a_{m-1})$ 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 $\boldsymbol{r}^{(\boldsymbol{v},\boldsymbol{v}')}$ denote the string $a_i a_{i+1} \cdots a_{m-1}$ with $D_{\boldsymbol{v}}(a_i, a_{i+1}, \ldots, a_{m-1}) \neq D_{\boldsymbol{v}'}(a_i, a_{i+1}, \ldots, a_{m-1})$. 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 $D(\boldsymbol{v}'\boldsymbol{r}^{(\boldsymbol{v})}) = D(\boldsymbol{v}\boldsymbol{r}^{(\boldsymbol{v})})$ and $D(\boldsymbol{v}'\dot{\boldsymbol{r}}^{(\boldsymbol{v})}) = D(\boldsymbol{v}\dot{\boldsymbol{r}}^{(\boldsymbol{v})})$.

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 binary 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 $\boldsymbol{a}\dot{\boldsymbol{r}}^{(\boldsymbol{v})}$ are asked to classify string $\boldsymbol{a} \in \mathcal{S}_i$. Each twin-test node has two outgoing edges, the right 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 left one is unlabeled and coming in a twin-test node or a leaf labeled μ. A single-test node has label $\boldsymbol{r}^{(\boldsymbol{v},\boldsymbol{v}')}$ for some $\boldsymbol{v}, \boldsymbol{v}' \in V_i$ and one membership query for $\boldsymbol{a}\boldsymbol{r}^{(\boldsymbol{v},\boldsymbol{v}')}$ is asked to classify string $\boldsymbol{a} \in \mathcal{S}_i$. The right outgoing edge is labeled $D(\boldsymbol{v}\boldsymbol{r}^{(\boldsymbol{v},\boldsymbol{v}')})$ and comes in a single-test node or a leaf labeled $\boldsymbol{v}$. The left outgoing edge is unlabeled and comes in a single-test node or a leaf labeled $\boldsymbol{v}'$.

Tree T_i has at most $|V_i| + 1$ leaves and each of them is labeled $\boldsymbol{v} \in V_i$ or

---

we do not need values $D(\boldsymbol{v}\boldsymbol{r}^{(\boldsymbol{v})})$ and $D(\boldsymbol{v}\dot{\boldsymbol{r}}^{(\boldsymbol{v})})$ to check whether $\boldsymbol{a} \in \mathcal{S}_i \setminus \bigcup_{\boldsymbol{v} \in V_i}[\boldsymbol{v}]$ holds or not. One merit of checking whether $D(\boldsymbol{a}\boldsymbol{r}^{(\boldsymbol{v})}) \neq D(\boldsymbol{v}\boldsymbol{r}^{(\boldsymbol{v})})$ or $D(\boldsymbol{a}\dot{\boldsymbol{r}}^{(\boldsymbol{v})}) \neq D(\boldsymbol{v}\dot{\boldsymbol{r}}^{(\boldsymbol{v})})$ holds, is that the set of equivalence classes $\{[\boldsymbol{v}] \mid \boldsymbol{v} \in V_i\}$ that $\boldsymbol{a}$ may belong to is narrowed by the condition that $D(\boldsymbol{a}\boldsymbol{r}^{(\boldsymbol{v})}) = D(\boldsymbol{v}\boldsymbol{r}^{(\boldsymbol{v})})$ and $D(\boldsymbol{a}\dot{\boldsymbol{r}}^{(\boldsymbol{v})}) = D(\boldsymbol{v}\dot{\boldsymbol{r}}^{(\boldsymbol{v})})$ more than by the condition that $D(\boldsymbol{a}\boldsymbol{r}^{(\boldsymbol{v})}) \neq D(\boldsymbol{a}\dot{\boldsymbol{r}}^{(\boldsymbol{v})})$ when $\boldsymbol{a} \in \bigcup_{\boldsymbol{v} \in V_i}[\boldsymbol{v}]$.

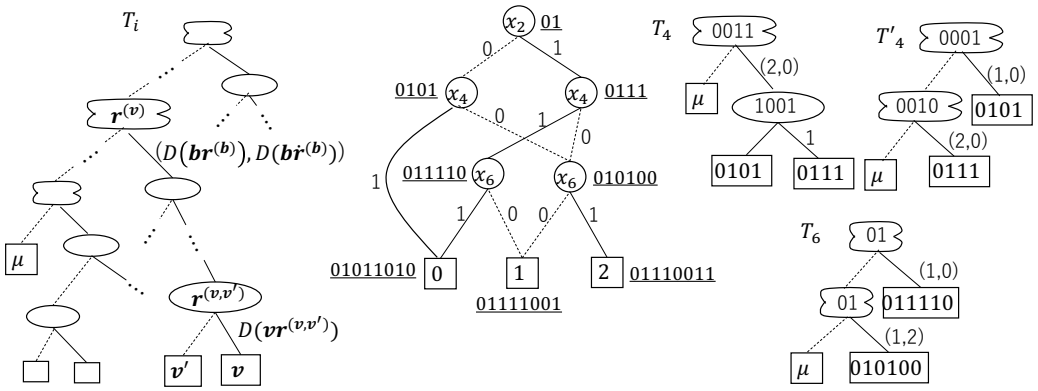


Figure 1: The form of a node classification tree (left) and its instances (right) for an example OMTBDD with the access strings of nodes (center).

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 right edge if $(D(\boldsymbol{ar}^{(v)}), D(\boldsymbol{a\dot{r}}^{(v)}))$ coincides with its label, otherwise select the left unlabeled edge. At a single-test node labeled $\boldsymbol{r}^{(v,v')}$, ask one membership query for $\boldsymbol{ar}^{(v,v')}$, and select the right outgoing edge if $D(\boldsymbol{ar}^{(v,v')})$ is equal to its label, otherwise select the left outgoing edge. The 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, T_6 and T'_4 (another choice of T_4) 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 the μ-labeled leaf. T'_4 and T_6 are composed of two twin-test nodes and three leaves. String $0011 \in \mathcal{S}_4$ reaches node 0111 in D. Since $D(00110011) = 2$, $D(00111011) = 0$ and $D(00111001) = 1$, it is classified into 0111 by T_4, which coincides with the reached node in D by the assignment $(x_0, x_1, x_2, x_3) = (0, 0, 1, 1)$.*

The node classification tree representation is not unique and T_4' is another realization of T_4. At the root twin test node of T_4', the left outgoing edge is selected because $D(00110001) = D(00111001) = 1$. String 0011 is also classified into 0111 by T_4' since the right outgoing edge is selected at the next twin test node from the membership query results $D(00110010) = 2$ and $D(00111010) = 0$. String $100111 \in \mathcal{S}_6$ does not reach any nodes in D. Since $D(10011101) = D(10011111) = 0$, it is classified into μ by T_6, which also coincides with the nonexistence of nodes reached by the assignment $(x_0, x_1, x_2, x_3, x_4, x_5) = (1, 0, 0, 1, 1, 1)$.

Remark 1. *Node classification trees for an OMTBDD are different from classification trees [4] for an OBDD in the following points. 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 the 1-labeled edge is selected for test string $\boldsymbol{a} \in \{0,1\}^i$ if and only if $D(\boldsymbol{ar}) = 1$ and $D(\boldsymbol{a\dot{r}}) = 0$. In the case with an OMTBDD, the number of combinations of different two function values can be more than one, thus one straightforward extension is to let a twin test node have an outgoing edge for each different (j, j') value for $j, j' = 0, \ldots, K-1$ and $j > j'$. Another natural extension is repeated bisection by checking whether $(D(\boldsymbol{ar}), D(\boldsymbol{a\dot{r}})) = (j, j')$ or not for one (j, j') at each twin test node. We adopt the latter extension and let the right outgoing edge have label $(j, j') \in \{0, \ldots, K-1\}^2$ and let the left outgoing edge have no label; (j, j')-edge is selected for $\boldsymbol{a}$ if and only if $(D(\boldsymbol{ar}), D(\boldsymbol{a\dot{r}})) = (j, j')$. We adopt the latter extension because its update cost is cheaper when one node is added, which is important in the learning process. As for a single-test node, repeated bisection (dividing the set of $\boldsymbol{a}$ with $D(\boldsymbol{ar}^{(\boldsymbol{v},\boldsymbol{v}')}) = j$ and the set of the others for some $j = 0, \ldots, K-1$) at each node is also adopted as an extension by the same reason, though a natural extension is to let a single-test node have an outgoing edge for each function value $0, \ldots, K-1$.*

Note that the node classification trees having child nodes of the number of different values are appropriate for the search of reached nodes in an OMTBDD because their depth can be smaller than the corresponding binary trees.

4. Algorithm

We extend the algorithm QLearn-π-OBDD [4] for OBDDs to an algorithm for OMTBDDs. Starting from a simple hypothesis OBDD, QLearn-π-OBDD

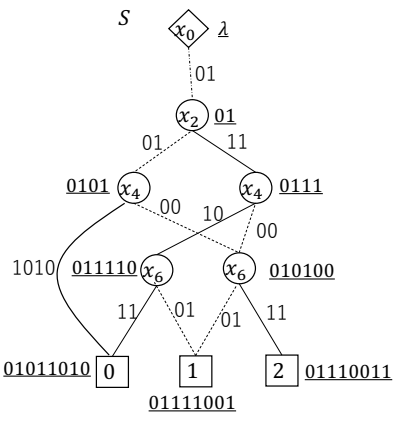
repeatedly asks an equivalence query for the current hypothesis OBDD and updates it using the obtained counterexample from the query until 'YES' is returned to the equivalence query. The basic structure of the extended algorithm is the same as QLearn-π-OBDD. In this section, we explain how to extend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

4.1. Hypothesis Data Structure

In QLearn-π-OBDD, the current hypothesis is stored with additional information to be updated easily. It keeps the current hypothesis as an *OBDD with access strings (OBDDAS)* and (node) classification trees. 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 following, we describe an extension of OBDDASs for OBDDs to those for OMTBDDs, which does not need modification except for 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 the following:

- Its root is always a node labeled x_0, 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 $m-i$. The first bits of the string labels of two edges going out from the same node must be different.



- Each node has an id string which is an access string of the corresponding node in the represented OMTBDD $\mathcal{D}(S)$ if the node is not a dummy, and λ (null string) if the node is a 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). Let $V(S)$ denote the set of the node ids in an OMTBDDAS S and let $V(\mathcal{D}(S))$ be that in $\mathcal{D}(S)$. We define $E(S)$ as the set of edges between nodes in $V(S)$, which is represented as a subset of $V(S) \times V(S) \times \{0, 1\}$; edge $(\boldsymbol{u}, \boldsymbol{v}, b)$ denotes the edge between nodes $\boldsymbol{u}$ and $\boldsymbol{v}$ whose first bit of label string is b. The label string of edge $(\boldsymbol{u}, \boldsymbol{v}, b) \in E(S)$ is denoted as $l_b(\boldsymbol{u}, \boldsymbol{v})$.

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 $V(S)$. The OMTBDD $\mathcal{D}(S)$ with node id set $V(\mathcal{D}(S))$ represented by the OBDDAS S is shown in the center of Figure 1.*

As QLearn-π-OBDD does[4], for an unknown target OMTBDD D, our extension of it 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 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. $\forall \boldsymbol{v}_1, \forall \boldsymbol{v}_2 \in \text{nodes}(D)$ 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)$V(\mathcal{D}(S)) \subseteq \text{nodes}(D)$,
(2)$\forall \boldsymbol{v} \in V(S)_m[\mathcal{D}(S)(\boldsymbol{v}) = D(\boldsymbol{v})]$, and
(3)$\forall \boldsymbol{v}_1, \forall \boldsymbol{v}_2 \in V(\mathcal{D}(S))[\boldsymbol{v}_1 \neq \boldsymbol{v}_2 \Rightarrow \boldsymbol{v}_1 \overset{D}{\neq} \boldsymbol{v}_2]$.

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

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

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

Algorithm 1 QLearn-OMTBDD()

Output: the reduced OMTBDD of a target function

```
(Ans, e) ← EQ(0)
if Ans = YES then return 0
ℓ ← D(e)
Create the initial OMTBDDAS S that is composed of a sink e labeled ℓ, a
    dummy root node λ labeled x_0, and an edge between them labeled e.
Set the initial node classification trees T_i to trees that are composed of one
    leaf labeled e for i = m and μ for i = 1, ..., m − 1.
loop
    if D(S)(e) = ℓ then
        (Ans, e) ← EQ(D(S))
        if Ans = YES then return D(S)
        ℓ ← D(e)
    (S, T_1, T_2, ..., T_m) ← Update-Hypothesis(S, T_1, T_2, ..., T_m, e, ℓ)
```

Condition CN is the set of conditions for the node id set $V(S)$: (1) any element of $V(\mathcal{D}(S))$ 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 $V(\mathcal{D}(S))$ must reach distinct nodes in D. Condition CT is the set of conditions for hypothesis node classification trees $T_1, T_2, \ldots, T_m$: (1) any node id except λ must be classified into itself, and (2) any non-access-string must be classified into μ. Condition CE is the set of conditions for edges in $E(S)$: 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.

4.2. Pseudocode

Our OMTBDD-version extension of algorithm QLearn-π-OBDD is called QLearn-OMTBDD, and its pseudocodes are shown in Algorithms 1–5. Thanks to adopting binary node classification trees, the algorithm needs no big modification to work for OMTBDDs.

First, QLearn-OMTBDD asks an equivalence query (EQ) for a trivial OMTBDD, denoted by $\mathbf{0}$, the OMTBDD being composed of only one sink labeled 0. If 'YES' is returned to the query, the algorithm outputs the hypothesis $\mathbf{0}$ and stops. If the answer is 'NO' and counterexample $\boldsymbol{e}$ is returned

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

Output: the updated OMTBDDAS S and node classification trees $T_1, \ldots, T_m$

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: $i \leftarrow i \in \{1, \ldots, k-1\}$ with $\ell = D(\boldsymbol{p}_i \mathrm{suf}(\boldsymbol{e}, m - |\boldsymbol{p}_i|)) \neq D(\boldsymbol{p}_{i+1} \mathrm{suf}(\boldsymbol{e}, m - |\boldsymbol{p}_{i+1}|))$
3: **if** $D(\boldsymbol{p}_i l_{e_{|\boldsymbol{p}_i|}}(\boldsymbol{p}_i, \boldsymbol{p}_{i+1}) \mathrm{suf}(\boldsymbol{e}, m - |\boldsymbol{p}_{i+1}|)) = \ell$ **then**
4: $\quad (S, T_1, T_2, ..., T_m) \leftarrow \mathrm{NodeSplit}(S, T_1, T_2, ..., T_m, \boldsymbol{e}, \boldsymbol{p}_i, \boldsymbol{p}_{i+1}, \ell)$
5: **else**
6: $\quad (S, T_1, T_2, ..., T_m) \leftarrow \mathrm{NewBranchingNode}(S, T_1, T_2, ..., T_m, \boldsymbol{e}, \boldsymbol{p}_i, \boldsymbol{p}_{i+1}, \ell)$
7: **return** $(S, T_1, T_2, ..., T_m)$

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.

Then, the algorithm makes an initial OMTBDDAS S that is composed of a sink $\boldsymbol{e}$ labeled ℓ, a dummy root node λ labeled x_0 and an edge between them labeled $\boldsymbol{e}$ at Line 4. The initial node classification tree T_m is set to the tree of one leaf only that is labeled $\boldsymbol{e}$, and other trees T_i $(i = 1, \ldots, m-1)$ are set to the trees of μ-labeled leaf only at Line 5.

Note that the initial OMTBDDAS S and node classification tree set $\{T_1, ..., T_m\}$ satisfy the conditions CN, CT and CE.

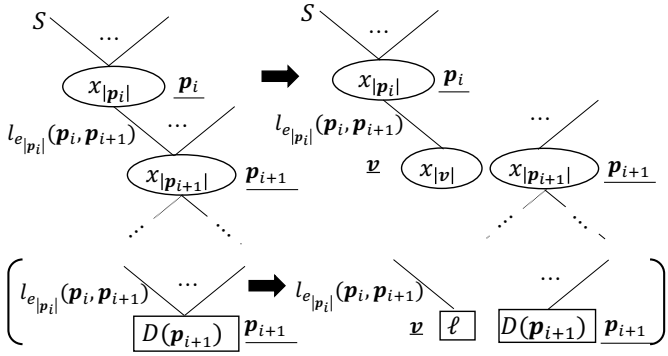
Assume that the algorithm has a current OMTBDDAS S and current node classification trees T_i for $i = 1, .., m$ which satisfy conditions CN, CT, and CE. Let $\boldsymbol{e}$ be the last counterexample and let $\ell = D(\boldsymbol{e})$. If $\mathcal{D}(S)(\boldsymbol{e}) \neq \ell$, then $\boldsymbol{e}$ is still a counterexample for the 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 just 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, $\mathcal{D}(S)(\boldsymbol{p}_k) = D(\boldsymbol{p}_k) \neq D(\boldsymbol{e})$. Let $\boldsymbol{e}_i = \mathrm{suf}(\boldsymbol{e}, m -$

Algorithm 3 NodeSplit$(S, T_1, \ldots, T_m, \boldsymbol{e}, \boldsymbol{p}_i, \boldsymbol{p}_{i+1}, \ell(= D(\boldsymbol{v}\boldsymbol{e}_{i+1})))$

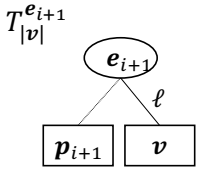
Output: the updated OMTBDDAS S and node classification trees $T_1, \ldots, T_m$

1: $\boldsymbol{e}_{i+1} \leftarrow \mathrm{suf}(\boldsymbol{e}, m - |\boldsymbol{p}_{i+1}|)$,
$\boldsymbol{v} \leftarrow \boldsymbol{p}_i l_{e_{|\boldsymbol{p}_i|}}(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$
$L \leftarrow \begin{cases} \ell & (\boldsymbol{e}_{i+1} = \lambda) \\ x_{|\boldsymbol{v}|} & (\text{otherwise}) \end{cases}$

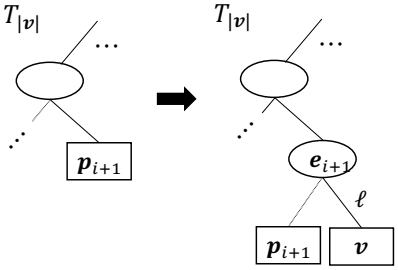


2: Add a node $\boldsymbol{v}$ labeled L and an edge $(\boldsymbol{p}_i, \boldsymbol{v}, e_{|\boldsymbol{p}_i|})$ labeled $l_{e_{|\boldsymbol{p}_i|}}(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$ to S, and remove the edge $(\boldsymbol{p}_i, \boldsymbol{p}_{i+1}, e_{|\boldsymbol{p}_i|})$ from S.

3: Create a tree $T_{|\boldsymbol{v}|}^{\boldsymbol{e}_{i+1}}$ that is composed of a single-test root node labeled $\boldsymbol{e}_{i+1}$, its right child node labeled $\boldsymbol{v}$ with an incoming edge labeled ℓ and the left child node with an unlabeled incoming edge.



4: **for** all $(\boldsymbol{v}_1, \boldsymbol{p}_{i+1}, b) \in E(S)$ **do**
5: 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 $\boldsymbol{q}' = l_b(\boldsymbol{v}_1, \boldsymbol{p}_{i+1})$.
6: **if** $D(\boldsymbol{v}_1\boldsymbol{q}'\boldsymbol{e}_{i+1}) = \ell$ **then**
7: Remove edge $(\boldsymbol{v}_1, \boldsymbol{p}_{i+1}, b)$ and add edge $(\boldsymbol{v}_1, \boldsymbol{v}, b)$ labeled $l_b(\boldsymbol{v}_1, \boldsymbol{p}_{i+1})$)



8: Replace the leaf node labeled $\boldsymbol{p}_{i+1}$ of the tree $T_{|\boldsymbol{v}|}$ with the tree $T_{|\boldsymbol{v}|}^{\boldsymbol{e}_{i+1}}$.
9: **if** $\boldsymbol{e}_{i+1} \neq \lambda$ **then**
10: $\boldsymbol{t} \leftarrow$ 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.
11: Add two new edges outgoing from $\boldsymbol{v}$ by executing $S \leftarrow \mathrm{AddEdge}(S, T_1, \ldots, T_m, \boldsymbol{v}, \boldsymbol{t})$ and $S \leftarrow \mathrm{AddEdge}(S, T_1, \ldots, T_m, \boldsymbol{v}, \dot{\boldsymbol{t}})$.
12: **return** $(S, T_1, T_2, ..., T_m)$

$|\boldsymbol{p}_i|$). 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 $D(\boldsymbol{e}) = D(\boldsymbol{p}_i\boldsymbol{e}_i) \neq D(\boldsymbol{p}_{i+1}\boldsymbol{e}_{i+1})$. Such i is calculated at Line 2 using $\lceil \log_2(k-1) \rceil (\leq \lceil \log_2 m \rceil)$ membership queries by a binary search. For this i, let $\boldsymbol{q}$ denote the label of the edge $(\boldsymbol{p}_i, \boldsymbol{p}_{i+1}, e_{|\boldsymbol{p}_i|})$, that is, $\boldsymbol{q} = l_{e_{|\boldsymbol{p}_i|}}(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$.

Algorithm 4 NewBranchingNode($S, T_1, \ldots, T_m, \boldsymbol{e}, \boldsymbol{p}_i, \boldsymbol{p}_{i+1}, \ell(= D(\boldsymbol{p}_i\boldsymbol{e}_i))$)

Output: the updated OMTBDDAS S and node classification trees $T_1, \ldots, T_m$

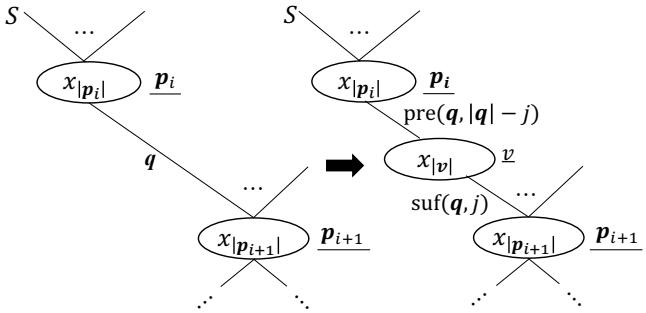
1: $\boldsymbol{e}_{i+1} \leftarrow \mathrm{suf}(\boldsymbol{e}, m - |\boldsymbol{p}_{i+1}|)$, $\boldsymbol{q} \leftarrow l_{e_{|\boldsymbol{p}_i|}}(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$

2: $j \leftarrow j \in \{0, \ldots, |\boldsymbol{q}| - 1\}$ with
$\ell = D(\boldsymbol{p}_i\mathrm{pre}(\boldsymbol{q}, j)\mathrm{suf}(\boldsymbol{e}, m-|\boldsymbol{p}_i|-j)) \neq D(\boldsymbol{p}_i\mathrm{pre}(\boldsymbol{q}, j+1)\mathrm{suf}(\boldsymbol{e}, m-|\boldsymbol{p}_i|-j-1))$

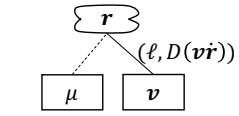
3: $\boldsymbol{v} \leftarrow \boldsymbol{p}_i \cdot \mathrm{pre}(\boldsymbol{q}, j)$,
$\boldsymbol{r} \leftarrow \mathrm{suf}(\boldsymbol{e}, m - |\boldsymbol{v}|)$

4: **if** $j \neq 0$ **then**

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



6: Create a tree $T^{\boldsymbol{r}}_{|\boldsymbol{v}|}$ that is composed of a twin-test node labeled $\boldsymbol{r}$ and its right and left child nodes labeled $\boldsymbol{v}$ and μ, respectively. Attach label $(\ell, D(\boldsymbol{v}\dot{\boldsymbol{r}}))$ to the right edge. Note that
$\boldsymbol{v}\dot{\boldsymbol{r}} = D(\boldsymbol{p}_i\mathrm{pre}(\boldsymbol{q}, j+1)\mathrm{suf}(\boldsymbol{e}, m - |\boldsymbol{v}| - 1)$.

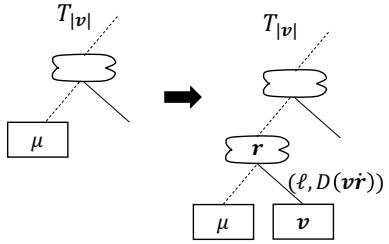


7: **for** all $(\boldsymbol{v}_1, \boldsymbol{v}_2, b) \in E(S)$ with $|\boldsymbol{v}_1| < |\boldsymbol{v}| < |\boldsymbol{v}_2|$ **do**

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

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

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



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

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
$S \leftarrow \mathrm{AddEdge}(S, T_1, \ldots, T_m, \boldsymbol{v}, \boldsymbol{r})$.

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

There are two cases depending on the value of $D(\boldsymbol{p}_i\boldsymbol{q}\boldsymbol{e}_{i+1})$. When $D(\boldsymbol{p}_i\boldsymbol{e}_i) = D(\boldsymbol{p}_i\boldsymbol{q}\boldsymbol{e}_{i+1}) \neq D(\boldsymbol{p}_{i+1}\boldsymbol{e}_{i+1})$, $\boldsymbol{p}_i\boldsymbol{q}$ and $\boldsymbol{p}_{i+1}$ must reach different nodes in D, and this case is dealt with in the algorithm NodeSplit (Algorithm 3), where a new node $\boldsymbol{v} = \boldsymbol{p}_i\boldsymbol{q}$ is split from the node $\boldsymbol{p}_{i+1}$ in the OMTBDDAS S, and

Algorithm 5 AddEdge$(S, T_1, \ldots, T_m, \boldsymbol{v}, \boldsymbol{t})$

Output: the updated OMTBDDAS S

1: **for** $j = 1$ to $|\boldsymbol{t}|$ **do**
2: $\quad \boldsymbol{u} \leftarrow T_{|\boldsymbol{v}|+j}(\boldsymbol{v}\mathrm{pre}(\boldsymbol{t}, j))$
3: $\quad$ **if** $\boldsymbol{u} \neq \mu$ **then**
4: $\quad\quad$ Add an edge $(\boldsymbol{v}, \boldsymbol{u}, t_0)$ labeled $\mathrm{pre}(\boldsymbol{t}, j)$ to S.
5: $\quad\quad$ **return** S

a leaf labeled $\boldsymbol{p}_{i+1}$ is replaced with a single-test node having the $\boldsymbol{p}_{i+1}$-labeled and $\boldsymbol{p}_i\boldsymbol{q}$-labeled child leaves in the node classification tree $T_{|\boldsymbol{p}_{i+1}|}$. When $D(\boldsymbol{p}_i\boldsymbol{e}_i) \neq D(\boldsymbol{p}_i\boldsymbol{q}\boldsymbol{e}_{i+1})$, there must exist a node $\boldsymbol{v}$ 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 a new node $\boldsymbol{v}$ is added to S for making the two paths branch, and a leaf labeled μ is replaced with a twin-test node having the $\boldsymbol{v}$-labeled and μ-labeled child leaves in the node classification tree $T_{|\boldsymbol{v}|}$. Both algorithms add a new node $\boldsymbol{v}$ to the current OMTBDDAS S, update $T_{|\boldsymbol{v}|}$, change all edges that must enter $\boldsymbol{v}$ and add edges going out from $\boldsymbol{v}$.

Let $(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_k)$ denote the id sequence of nodes in the path of the current counterexample $\boldsymbol{e}$ in the current OMTBDDAS S. In NodeSplit, a new node $\boldsymbol{v} = \boldsymbol{p}_i\boldsymbol{q}$ is split from node $\boldsymbol{p}_{i+1}$, where $\boldsymbol{q}$ is the label string of the edge $(\boldsymbol{p}_i, \boldsymbol{p}_{i+1}, e_{|\boldsymbol{p}_i|})$ passed by $\boldsymbol{e}$, whose incoming node is changed to $\boldsymbol{v}$. In the case with $|\boldsymbol{v}|(= |\boldsymbol{p}_{i+1}|) = m$, that is, when $\boldsymbol{e}_{i+1} = \lambda$, node $\boldsymbol{v}$ is a sink and its label is set to $\ell(= D(\boldsymbol{v}\boldsymbol{e}_{i+1}) = D(\boldsymbol{v}))$. Otherwise, node $\boldsymbol{v}$ is labeled $x_{|\boldsymbol{v}|}$. In the node classification tree $T_{\boldsymbol{v}}$, the leaf node labeled $\boldsymbol{p}_{i+1}$ is replaced with $T_{|\boldsymbol{v}|}^{\boldsymbol{e}_{i+1}}$, whose root node is the single-test node labeled $\boldsymbol{e}_{i+1}$ having the right child leaf labeled $\boldsymbol{v}$ and the left child leaf labeled $\boldsymbol{p}_{i+1}$, at Lines 3 and 8. The label of the right edge is set to ℓ and the left edge is unlabeled. Some edges incoming to $\boldsymbol{p}_{i+1}$ should come in $\boldsymbol{v}$, which is checked for all edges $(\boldsymbol{v}_1, \boldsymbol{p}_{i+1}, b) \in E(S)$ using $T_{|\boldsymbol{v}|}^{\boldsymbol{e}_{i+1}}$, and the incoming node of the edges $(\boldsymbol{v}_1, \boldsymbol{p}_{i+1}, b)$ with $T_{|\boldsymbol{v}|}^{\boldsymbol{e}_{i+1}}(\boldsymbol{v}_1 l_b(\boldsymbol{v}_1, \boldsymbol{p}_{i+1})) = \boldsymbol{v}$ is changed to $\boldsymbol{v}$ at Line 7. If $\boldsymbol{v}$ is not a sink, two edges must go out from $\boldsymbol{v}$. Since string $\boldsymbol{v} = \boldsymbol{p}_i\boldsymbol{q}$ is classified into $\boldsymbol{p}_{i+1}$ by $T_{|\boldsymbol{v}|}$ before replacing the $\boldsymbol{p}_{i+1}$-labeled leaf with $T_{|\boldsymbol{v}|}^{\boldsymbol{e}_{i+1}}$, the right edge is selected at the last twin test node. Let $\boldsymbol{t}$ be the label of the last twin test node. AddEdge is executed for $\boldsymbol{v}$ and $\boldsymbol{t}$, and also for $\boldsymbol{v}$ and $\dot{\boldsymbol{t}}$. In AddEdge, string $\boldsymbol{v}\mathrm{pre}(\boldsymbol{t}, j)$ is classified by $T_{|\boldsymbol{v}|+j}$ from $j = 1$ to $|\boldsymbol{t}|$ until $T_{|\boldsymbol{v}|+j}(\boldsymbol{v}\mathrm{pre}(\boldsymbol{t}, j)) = \boldsymbol{u} \neq \mu$. Then, edge $(\boldsymbol{v}, \boldsymbol{u}, t_0)$ is added to S and its label

is set to $\mathrm{pre}(\boldsymbol{t}, j)$.

In NewBranchingNode, a new node $\boldsymbol{v}$ is inserted between nodes $\boldsymbol{p}_i$ and $\boldsymbol{p}_{i+1}$. Since NewBranchingNode is executed in the case with $\ell = D(\boldsymbol{e}) = D(\boldsymbol{p}_i\boldsymbol{e}_i) \neq D(\boldsymbol{p}_i\boldsymbol{q}\boldsymbol{e}_{i+1})$, there is a $j \in \{0, \cdots, |\boldsymbol{q}-1|\}$ such that $\ell = D(\boldsymbol{p}_i\mathrm{pre}(\boldsymbol{q}, j)\mathrm{suf}(\boldsymbol{e}, m-|\boldsymbol{p}_i|-j)) \neq D(\boldsymbol{p}_i\mathrm{pre}(\boldsymbol{q}, j+1)\mathrm{suf}(\boldsymbol{e}, m-|\boldsymbol{p}_i|-j-1))$, where $\boldsymbol{q} = l_{e_{|\boldsymbol{p}_i|}}(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$. Such j is found by binary search at Line 2. If $\boldsymbol{p}_i$ is a non-dummy node, $q_0 = e_{|\boldsymbol{p}_i|}$ holds, which means $D(\boldsymbol{p}_i\mathrm{pre}(\boldsymbol{q}, 0)\mathrm{suf}(\boldsymbol{e}, m-|\boldsymbol{p}_i|)) = D(\boldsymbol{p}_i\mathrm{pre}(\boldsymbol{q}, 1)\mathrm{suf}(\boldsymbol{e}, m-|\boldsymbol{p}_i|-1))$, that is, $j \neq 0$. Thus, $j = 0$ means that node $\boldsymbol{p}_i (= \boldsymbol{p}_1)$ is a dummy node. In that case, which is dealt with at Line 13, a dummy node becomes a non-dummy node $\boldsymbol{v}$, and another outgoing edge is added by executing AddEdge with $\boldsymbol{v} = \lambda$ and $\boldsymbol{r} = \boldsymbol{e}$ at Line 14. When $j \neq 0$, the edge labeled $\boldsymbol{q}$ is divided into the edge labeled $\mathrm{pre}(\boldsymbol{q}, j)$ and the edge labeled $\mathrm{suf}(\boldsymbol{q}, |\boldsymbol{q}| - j)$, and a new node $\boldsymbol{v} = \boldsymbol{p}_i\mathrm{pre}(\boldsymbol{q}, j)$ labeled $x_{|\boldsymbol{v}|}$ is inserted between them at Line 5. In $T_{|\boldsymbol{v}|}$, the leaf labeled μ is replaced with a tree $T^{\boldsymbol{r}}_{|\boldsymbol{v}|}$ at Line 11, which is a twin-test node labeled $\boldsymbol{r} = \mathrm{suf}(\boldsymbol{e}, m - |\boldsymbol{v}|)$ having the right child leaf labeled $\boldsymbol{v}$ connected by an edge labeled $(\ell(= D(\boldsymbol{v}\boldsymbol{r})), D(\boldsymbol{v}\dot{\boldsymbol{r}}))$ and the left child leaf labeled μ connected by an unlabeled edge (Line 6). All the edges $(\boldsymbol{v}_1, \boldsymbol{v}_2, b) \in E(S)$ that should come in $\boldsymbol{v}$ are checked using $T^{\boldsymbol{r}}_{|\boldsymbol{v}|}$ at Line 7–10; the incoming node of the edges $(\boldsymbol{v}_1, \boldsymbol{v}_2, b)$ are changed to $\boldsymbol{v}$, and the labels of the strings are cut as $\mathrm{pre}(l_b(\boldsymbol{v}_1, \boldsymbol{v}_2), |\boldsymbol{v}| - |\boldsymbol{v}_1|)$. As in the case with $j = 0$, another edge outgoing from $\boldsymbol{v}$ is created by executing AddEdge with $\boldsymbol{v}$ and $\boldsymbol{r}$ at Line 14.

4.3. Example of Algorithm Execution

An example of QLearn-OMTBDD execution is shown in Figure 2–4. The target OMTBDD D is shown in the top left figure of Figure 2. Obtaining a counterexample 10100100 with $D(10100100) = 1$ for the first equivalence query, the initial OMTBDDAS and node classification trees $(S, T_1, \ldots, T_8)$ constructed at Lines 4 and 5 in the algorithm QLearn-OMTBDD are shown in $\boxed{1}$.

Since $\mathcal{D}(S)(10100100) = 1$, an equivalence query is asked for $\mathcal{D}(S)$ at Line 8 in QLearn-OMTBDD and a counterexample 01111100 with $\ell = D(01111100) = 0$ is assumed to be obtained. As a sequence of access strings passed by 01111100, $(\boldsymbol{p}_1, \boldsymbol{p}_2) = (\lambda, 10100100)$ is obtained at Line 1 in Update-Hypothesis. At Line 2 in Update-Hypothesis, i is set to 1 because $0 = D(01111100) = D(\boldsymbol{p}_1\mathrm{suf}(01111100, 8)) \neq D(\boldsymbol{p}_2\mathrm{suf}(01111100, 0)) = D(10100100) = 1$ holds. Since $D(\boldsymbol{p}_1 l_0(\boldsymbol{p}_1, \boldsymbol{p}_2)\mathrm{suf}(01111100, 0)) = D(10100100) \neq D(01111100) = 0$ holds, algorithm NewBranchingNode is executed at Line 6. At Line 2

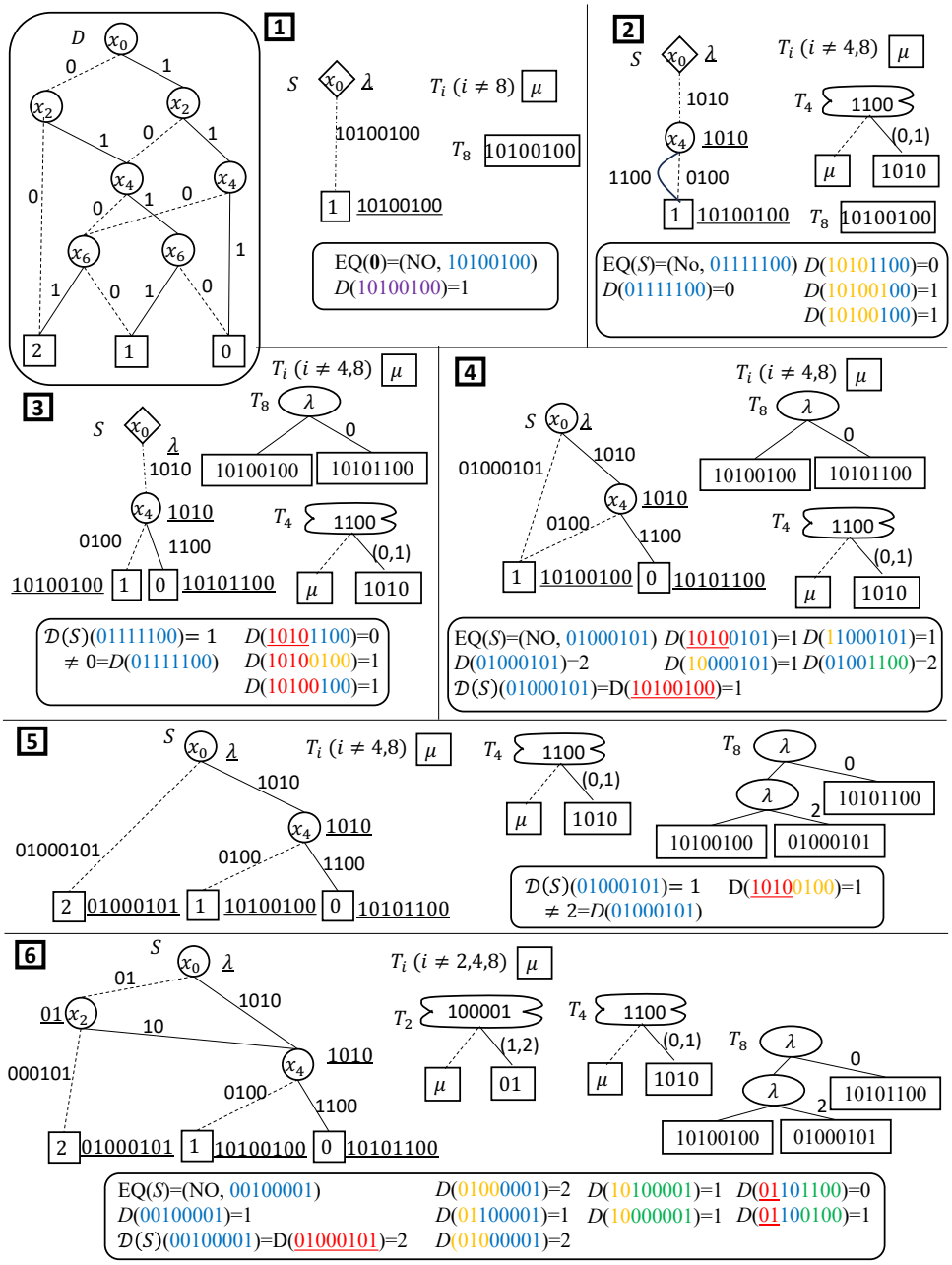


Figure 2: An OMTBDDAS constriction example by algorithm QLearn-OMTBDD (1)

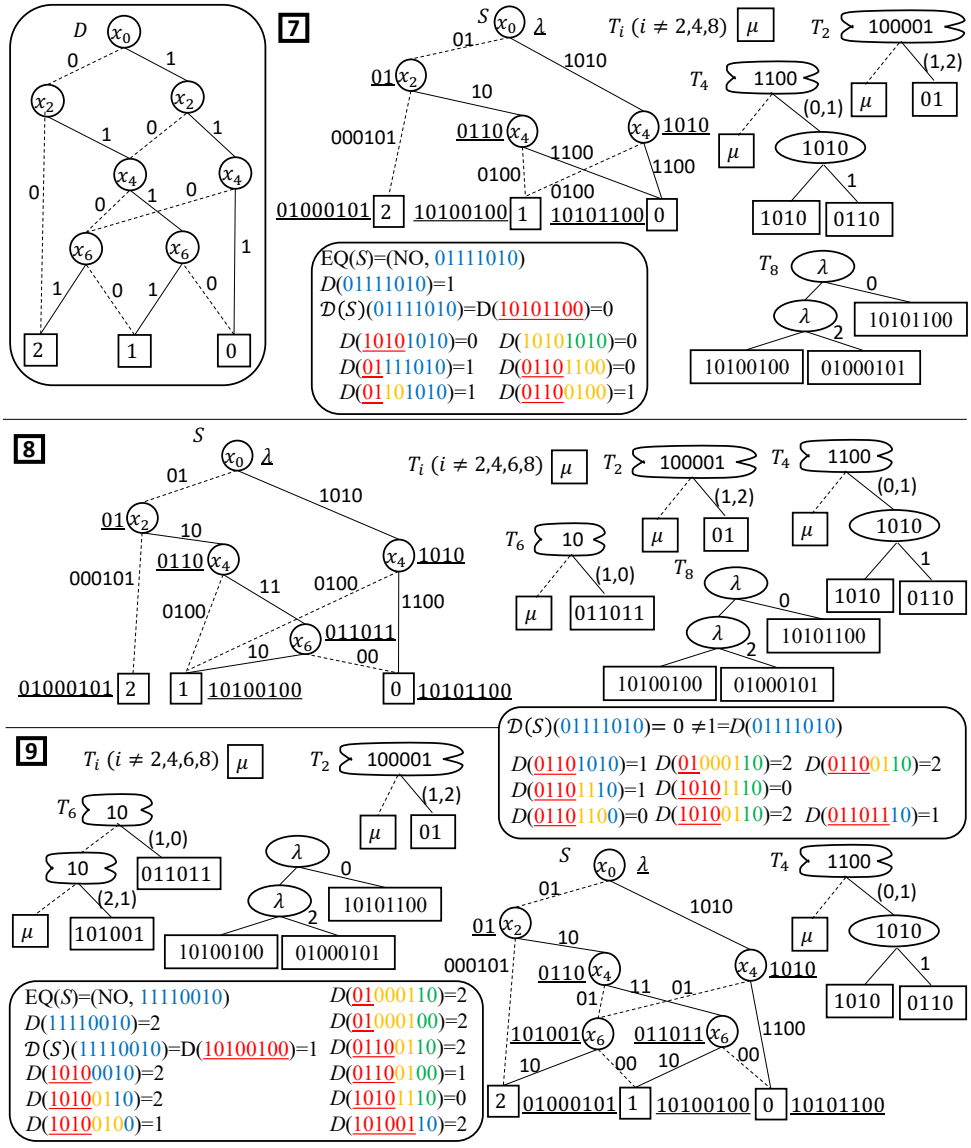


Figure 3: An OMTBDDAS constriction example by algorithm QLearn-OMTBDD (2)

of NewBranchingNode, j is set to $4 \neq 0$ because $0 = D(10101100) = D(\lambda\text{pre}(10100100, 4)\text{suf}(01111100, 4)) \neq D(\lambda\text{pre}(10100100, 5)\text{suf}(01111100, 3)) = D(10100100) = 1$. Thus, Line 5 is executed, the edge labeled 10100100 is divided into two edges, edges labeled 1010 and 0100, and a new node $\boldsymbol{v} = 1010$

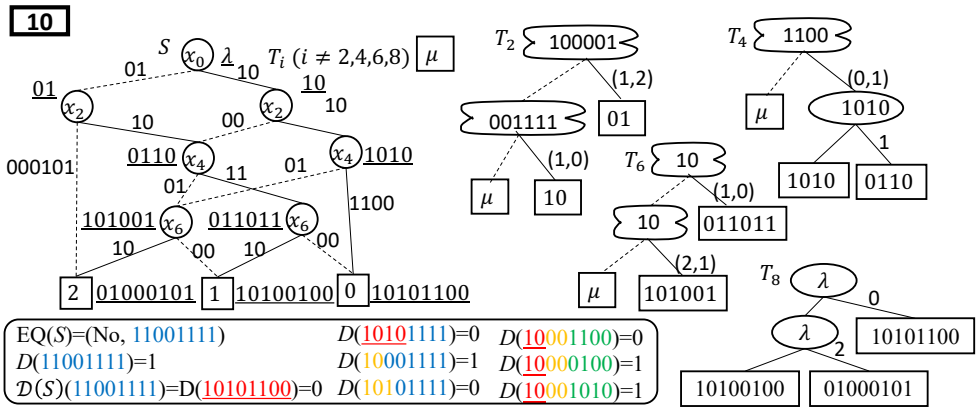


Figure 4: An OMTBDDAS constriction example by algorithm QLearn-OMTBDD (3)

labeled x_4 is added to S for connecting them. There is no edge $(\boldsymbol{v}_1, \boldsymbol{v}_2, b) \in E(S)$ with $|\boldsymbol{v}_1| < |\boldsymbol{v}| < |\boldsymbol{v}_2|$, thus nothing is done in the for-loop at Line 7. Since $\boldsymbol{r}$ is set to 1100 at Line 3, an edge outgoing from $\boldsymbol{v} = 1010$ is added by executing the algorithm AddEdge with $\boldsymbol{v} = 1010, \boldsymbol{t} = 1100$ (Algorithm 5). (See 2.) In the algorithm AddEdge, since $T_{|\boldsymbol{v}|+j}(1010\mathrm{pre}(1100, j)) = \mu$ for all $j = 1, \ldots, 3$ and $T_8(10101100) = 10100100$, an edge $(\boldsymbol{v}, 10100100)$ labeled 1100 is added to S (Line 4 in AddEdge). As for node classification trees, T_4 is updated from one node tree with a μ-labeled leaf to a twin-test root node labeled x_4 having the right child leaf labeled $\boldsymbol{v} = 1010$ connected by an edge labeled $(\ell, D(\boldsymbol{v}\dot{\boldsymbol{r}})) = (0, 1)$ and the left child leaf labeled μ connected by an unlabeled edge at Line 6 and 11.

Since $\boldsymbol{e} = 01111100$ is still a counterexample for the updated S shown in 2, that is, $\mathcal{D}(S)(01111100) = 1 \neq 0 = \ell$ (Line 7 in QLearn-OMTBDD), Update-Hypothesis is executed for $\boldsymbol{e} = 01111100$ again. In Update-Hypothesis, $(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3)$ is set to $(\lambda, 1010, 10100100)$ at Line 1. At Line 2, i is set to 2 because $0 = D(10101100) = D(\boldsymbol{p}_2\mathrm{suf}(01111100, 4)) \neq D(\boldsymbol{p}_3\mathrm{suf}(01111100, 0)) = D(10100100) = 1$ holds. Since $0 = D(10101100) = D(\boldsymbol{p}_2 l_1(\boldsymbol{p}_2, \boldsymbol{p}_3)\mathrm{suf}(01111100, 0))$ holds, algorithm NodeSplit is executed at Line 4. In algorithm NodeSplit, a new sink 10101100 is split from the sink $\boldsymbol{p}_3 = 10100100$ (Line 2), the leaf node labeled 10100100 in T_8 is replaced (Line 8) with a single-test node labeled λ that has the right child node labeled 10101100 connected by an edge labeled 0 and the left child node labeled 10100100 connected by an unlabeled edge (Line 3). (See 3.)

The next counterexample is assumed to be 01000101 with $D(01000101) = 2$ in $\boxed{4}$. The sequence of access strings passed by 01000101 is $(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3) = (\lambda, 1010, 10100100)$ in Update-Hypothesis, and i is set to 1 because $2 = D(01000101) = D(\boldsymbol{p}_1\mathrm{suf}(01000101, 8)) \neq D(\boldsymbol{p}_2\mathrm{suf}(01000101, 4)) = D(10100101) = 1$ holds. Since $2 = D(01000101) \neq D(\boldsymbol{p}_1 l_0(\boldsymbol{p}_1, \boldsymbol{p}_2)\mathrm{suf}(01000101, 4)) = D(10100101) = 1$ holds, algorithm NewBranchingNode is executed at Line 6. At Line 2 of NewBranchingNode, j is set to 0 because $2 = D(01000101) = D(\lambda\mathrm{pre}(1010, 0)\mathrm{suf}(01000101, 8)) \neq D(\lambda\mathrm{pre}(1010, 1)\mathrm{suf}(01000101, 7)) = D(11000101) = 1$. 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 the algorithm AddEdge with $\boldsymbol{v} = \lambda, \boldsymbol{t} = 01000101$. In the algorithm AddEdge, $T_{|\lambda|+j}(\lambda\mathrm{pre}(01000101, j)) = T_j(\mathrm{pre}(01000101, j)) = \mu$ for all $j = 1, \ldots, 7$ and $T_8(01000101) = 10100100$. Thus, an edge $(\lambda, 10100100)$ labeled 01000101 is added to S (Line 4 in AddEdge).

Algorithm NodeSplit is also executed in $\boxed{7}$. For S shown in $\boxed{6}$, 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 $1 = D(01111010) = D(\boldsymbol{p}_2\mathrm{suf}(01111010, 6)) \neq D(\boldsymbol{p}_3\mathrm{suf}(01111010, 4) = D(10101010) = 0$ at Line 2 in Update-Hypothesis. In this case, $D(\boldsymbol{p}_2 l_1(\boldsymbol{p}_2, \boldsymbol{p}_3)\mathrm{suf}(01111010, 4)) = D(01101010) = 1$ holds, thus 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 8) with a single-test node labeled 1010 that has the right child leaf labeled 0110 connected by an edge labeled 1 and the left child leaf labeled 1010 connected by an unlabeled edge (Line 3). After this node split, the last counterexample 01111010 is still a counterexample for the updated S shown in $\boxed{7}$, that is, $0 = \mathcal{D}(S)(01111010) \neq \ell = D(01111010) = 1$, therefore Lines 8-10 in QLearn-OMTBDD are not executed and Update-Hypothesis is executed for the same $(\boldsymbol{e}, \ell)$. In the update from $\boxed{8}$ to $\boxed{9}$, node 101001 labeled x_6 is added to S by the algorithm NewBranchingNode, in which the edge $(1010, 10100100)$ is replaced with the edge $(1010, 101001)$. This is done during the execution of the for-loop of Lines 7-10.

Since the OMTBDD corresponding to the OMTBDDAS S shown in $\boxed{10}$ is equivalent to D, the answer of the equivalence query for $\mathcal{D}(S)$ 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 with $n \geq 3$, algorithm QLearn-OMTBDD exactly learns D using at most n equivalence queries and less than $2n(\lceil \log_2 m \rceil + 2n)$ membership queries, where $m(\geq 1)$ is the number of variables and n is the number of nodes in D.*

Proof. See Appendix A. □

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 membership queries.

5. Experiments

We conducted experiments to show the effectiveness of our algorithm using a synthetic dataset and several benchmark datasets in the 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 $n = 3200$ 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 asked to learn OMTBDDs of each parameter triplet (n, m, K) is averaged over the ten OMTBDDs.

The results are shown in Figure 5. 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. Both the axes are log-scaled. Numerical values in the tables are rounded to three significant digits. Theoretical upper bounds ($2n(\lceil \log_2 m \rceil + 2n)$ for membership queries, n for equivalence queries, where n and m are the numbers of nodes and variables, respectively) are also shown for comparison. In terms of 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

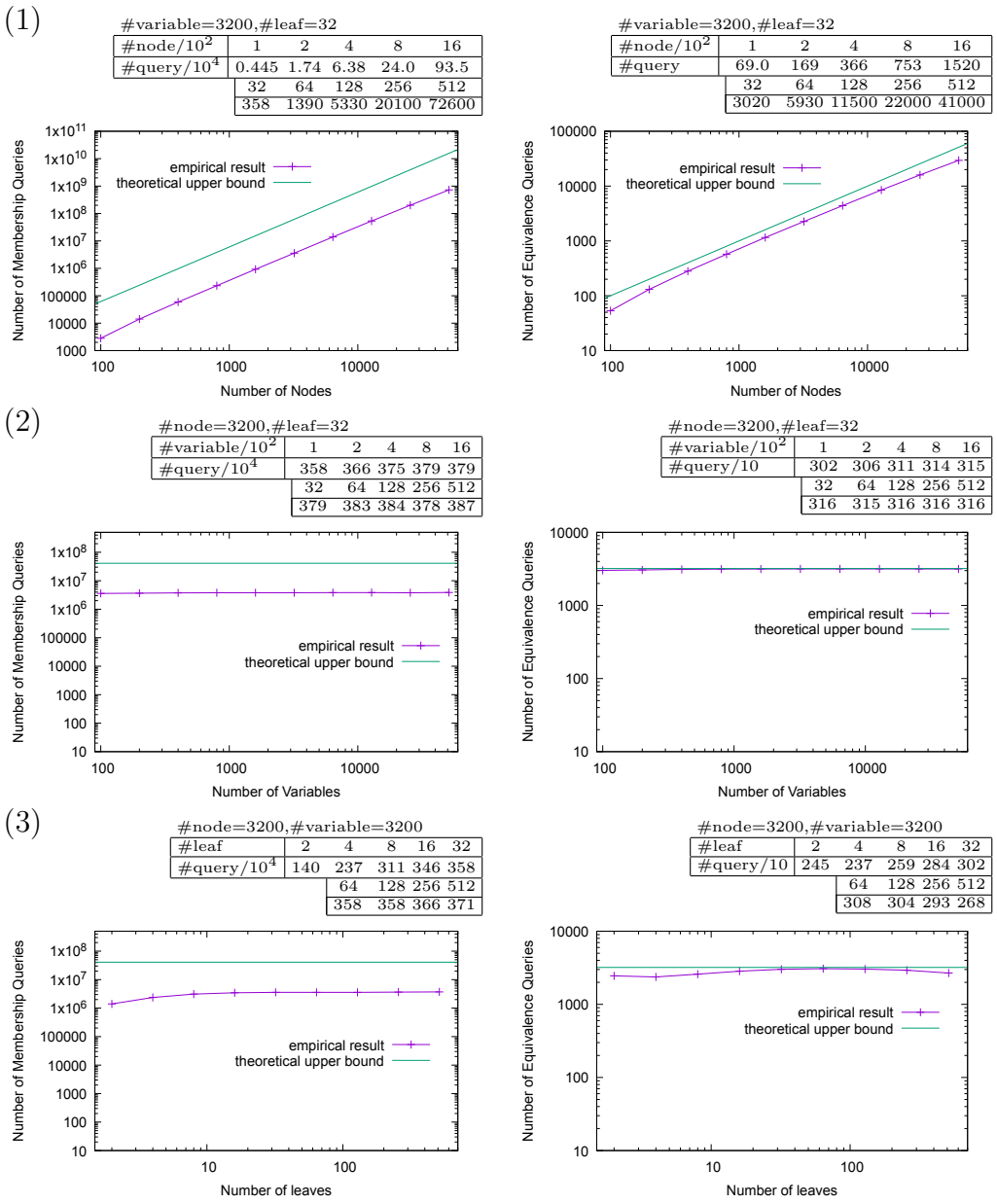


Figure 5: The number of membership and equivalence queries (1) for various numbers of nodes, (2) for various numbers of variables, and (3) for various numbers of leaves.

figure. With respect to the number of variables, $O(\log m)$ and $O(1)$ are the

Table 1: Datasets of the UCI Machine Learning Repository [8] that are 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

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 smaller as the number of leaves decreases, which means the complexity of the function class decreases. That might be the reason for this phenomenon.

5.2. Benchmark Datasets

One application of our algorithm is the 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 the UCI Machine Learning Repository [8] that are composed of real-valued features only. (See Table 1) The process of our experiment for each dataset $\mathcal{D} \subset \mathbb{R}^k \times \{0, \ldots, K-1\}$ is as follows.

1. A tree-based classifier $C[\mathrm{CT}_1, \ldots, \mathrm{CT}_d] : \mathbb{R}^k \to \{0, \ldots, K-1\}$ is learned using a given dataset $\mathcal{D}$, where $\mathrm{CT}_1, \ldots, \mathrm{CT}_d$ are component

decision trees with variables $y_1, \ldots, y_k$ over $\mathbb{R}$. Let $\mathcal{D}^{\text{correct}}$ denote the set of data $(\boldsymbol{y}, \ell) \in \mathcal{D}$ that satisfies $C[\mathrm{CT}_1, \ldots, \mathrm{CT}_d](\boldsymbol{y}) = \ell$.

2. The set of branching conditions $\mathrm{BC} = \{y_i \geq \theta$ is a branching condition of some node in some component decision tree $\mathrm{CT}_j\}$ is reduced to $\mathrm{BC}'(\subseteq \mathrm{BC})$ by the branching condition sharing algorithm Min_DBN [9]. The tree-based classifier $C[\mathrm{CT}_1, \ldots, \mathrm{CT}_d]$ is converted to a simpler classifier $C[\mathrm{CT}'_1, \ldots, \mathrm{CT}'_d]$ whose set of branching conditions is BC' by sharing some branching conditions.
3. Let x_j denote a $\{0,1\}$-variable whose value is 1 if and only if $y_{i_j} \geq \theta_j$, where $y_{i_j} \geq \theta_j$ is the jth branching condition in BC'. Let m denote the number of distinct branching conditions in BC'. Each training data $(\boldsymbol{y}, \ell) \in \mathcal{D}^{\text{correct}}$ is converted to a binary feature data $(\boldsymbol{x}, \ell) \in \{0,1\}^m \times \{0, \ldots, K-1\}$ using the definition of x_j through the condition $y_{i_j} \geq \theta_j$. Let $\mathcal{D}^{\mathrm{b}}$ denote the set of the converted training data $(\boldsymbol{x}, \ell)$. The classifier $C[\mathrm{CT}'_1, \ldots, \mathrm{CT}'_d] : \mathbb{R}^k \to \{1, \ldots, K-1\}$ is also converted to $C[\mathrm{CT}^{\mathrm{b}}_1, \ldots, \mathrm{CT}^{\mathrm{b}}_d] : \{0,1\}^m \to \{1, \ldots, K-1\}$ by replacing the branching conditions $y_{i_j} \geq \theta_j$ assigned to the nodes in each decision tree CT'_l $(l = 1, \ldots, d)$ with the condition $x_j = 1$ to construct CT^b_l.
4. The variable ordering of the binary features $x_1, \ldots, x_m$ is decided as follows. Consider a directed graph $G(V, E)$ that is composed of m vertices corresponding to the binary features. Here, we abuse notation and let each binary feature x_j also represent vertex x_j in V. For each pair of two binary features (x_i, x_j), count the number $\#(i \to j)$ of occurrences that a node with a branching condition $x_i = 1$ is an ancestor of a node with a branching condition $x_j = 1$ in $\mathrm{CT}^{\mathrm{b}}_1, \ldots, \mathrm{CT}^{\mathrm{b}}_d$, and the number $\#(j \to i)$ of occurrences that the opposite relation holds. The edge set E is composed of directed edges (x_i, x_j) for pairs of binary features (x_i, x_j) that satisfy $\#(i \to j) > \#(j \to i)$. Set the weight of edge $(x_i, x_j) \in E$ to $\#(i \to j) - \#(j \to i)$. Remove edges in increasing order of their weights until the topological sorting of the whole vertices $x_1, \ldots, x_m$ is succeeded. Define the binary feature order as the corresponding topological sorted vertex order $x_{i_1}, \ldots, x_{i_m}$.
5. An OMTBDD is learned by QLearn-OMTBDD using $C[\mathrm{CT}^{\mathrm{b}}_1, \ldots, \mathrm{CT}^{\mathrm{b}}_d]$ as the membership oracle and using consistency with all the data in $\mathcal{D}^{\mathrm{b}}$ as equivalence to the target function for an equivalence query and returning an inconsistent data in $\mathcal{D}^{\mathrm{b}}$ as a counterexample.

As tree-based classifiers, a decision tree and a random forest are used

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	20.0	0.924
blood	320	(162)	0.713	106	72.2	335	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	3460	0.657
wine quality white	2088	(1050)	0.604	769	370	23100	0.606
waveform	797	(401)	0.735	397	304	3810	0.741
robot	57.4	(32.7)	0.995	28.0	22.4	45.6	0.996
musk	252	(128)	0.965	126	121	123	0.964
epileptic seizure	3690	(1850)	0.472	1830	1290	410000	0.456
magic	3200	(1600)	0.818	1600	771	49000	0.818

in our experiment. We adopt DecisionTreeClassifier and RandomForestClassifier 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 the 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, respectively. 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 classifiers whose same-labeled leaves share a single leaf. An OMTBDD has only one leaf with the same label, thus comparison in the 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 a

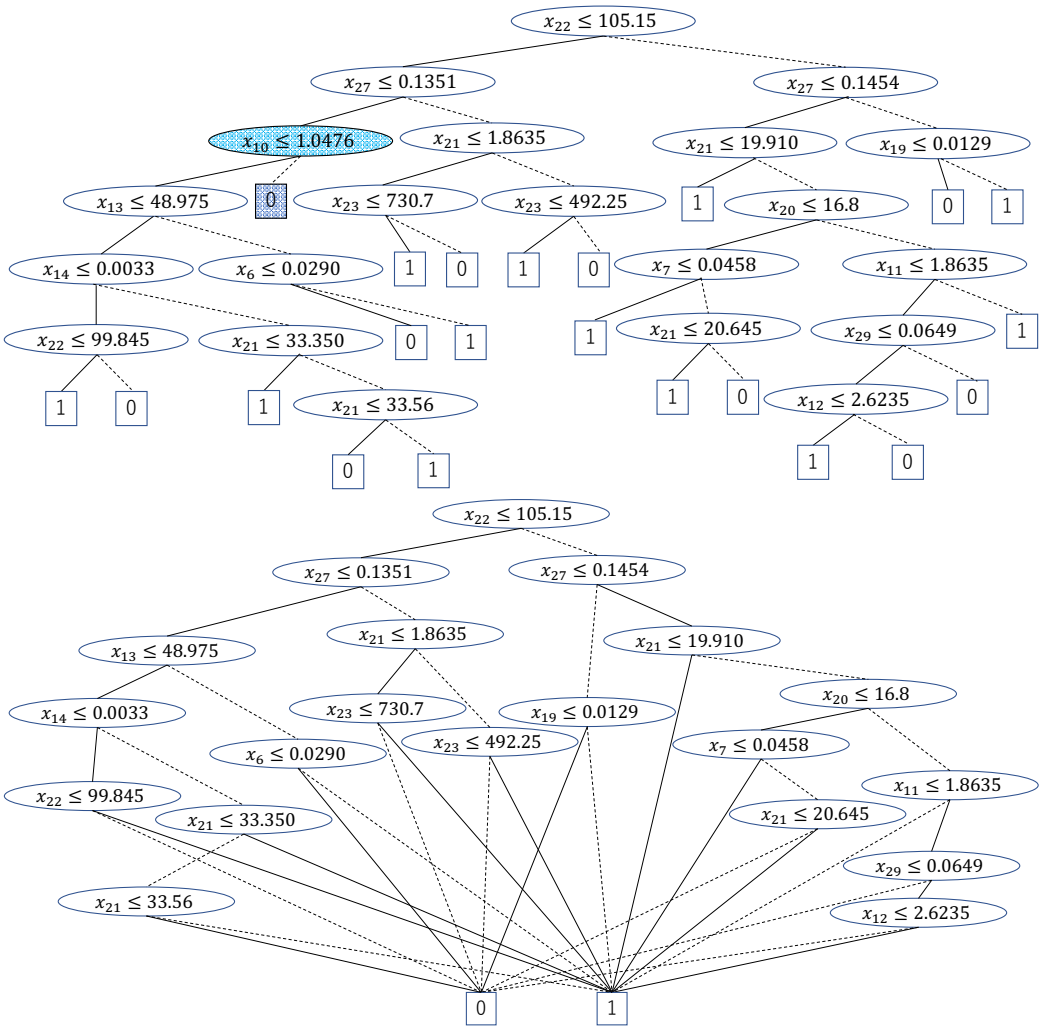


Figure 6: Decision tree for breast cancer dataset and the OMTBDD learned from it.

simple classifier problem, in which the original decision tree has less than 30 nodes, their leaf-shared decision trees are already OMTBDDs, and the query learning algorithm learns them exactly. (See the results for the iris, parkinson, RNA-Seq PANCAN datasets.) Interesting results are those for the 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 the breast cancer dataset is shown

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	19.4	0.947
parkinson	2660	(1480)	0.892	1010	404	175	0.892
breast cancer	3570	(1940)	0.961	1420	594	242	0.944
blood	23200	(11800)	0.749	322	145	1540	0.738
RNA-Seq PANCAN	3960	(2430)	0.996	1920	1840	2480	0.903
wine quality red	55900	(29000)	0.692	4120	900	168000	0.610
wine quality white	181000	(91600)	0.679	7290	1400	1150000	0.608
waveform	83800	(42200)	0.854	32500	5870	1760000	0.703
robot	25000	(12900)	0.995	9900	3140	746	0.996
musk	34600	(17500)	0.975	12200	6240	563000	0.906

in Figure 6. 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 it. The difference between them is the shaded part. The original decision tree is constructed in a top-down manner, and in that manner, $x_{10} \leq 1.0476$ is a good condition[5] 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 the epileptic seizure dataset, OMTBDD size is less than 31 times larger than the leaf-shared decision tree size of the original decision tree, and no accuracy deterioration is observed. For the epileptic seizure dataset, the OMTBDD size is 222 times larger and accuracy is deteriorated. The decision boundary of the decision tree for the 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, the 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.

[5]In the default setting, which we use in the experiments, DecisionTreeClassifier of scikit-learn chooses the split condition using the pair of a feature and a value whose Gini impurity is the best. Thus, 'good' meaning here is good in Gini impurity.

Especially for parkinson, breast cancer and blood 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 sizes are smaller than the random forests. For the other 5 datasets, accuracies are deteriorated significantly and sizes become $1.02 - 41.7$ 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 the epileptic seizure dataset.

6. Conclusions and Future Work

We developed a query learning algorithm QLearn-OMTBDD for OMTBDDs by extending QLearn-π-OBDD [4] for OBDDs. In our algorithm, node classification trees, which classify assignment prefixes into OMTBDD nodes or μ (no node corresponds to the assignment prefix) using membership queries, play an important role, and the key to the algorithm extension is how to extend the node classification trees to deal with more than two possible answers of membership queries. Our extension keeps their binary tree structure, which makes the algorithm extension simple. In the experiments using benchmark datasets, we showed the 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.

It is interesting to pursue a better way of finding compact and accurate OMTBDDs for real-world datasets including feature binarization and variable order decision.

Acknowledgments

This work was supported by JST CREST Grant Number JPMJCR18K3, Japan.

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 Lemmas 1, 2 and Theorem 1 for OMTBDDs are extensions of the proofs of Lemmas 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 $V(\mathcal{D}(S))$ 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, thus internal node labels are the same. The sink labels are guaranteed to be the same by CN (2). Thus $\langle 2 \rangle$ holds.

$\langle 3 \rangle$ 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 $(\boldsymbol{v}_1, \boldsymbol{v}_2, b) \in E(S)$.

Let $(\boldsymbol{v_1}, \boldsymbol{v}, b) \in E(S)$. Assume that $|\boldsymbol{v}| < |\boldsymbol{v}_2|$. Since $\boldsymbol{v}_1 \cdot l_b(\boldsymbol{v}_1, \boldsymbol{v}) \notin$ nodes(D), $T_{|v|}(\boldsymbol{v}_1 \cdot l_b(\boldsymbol{v}_1, \boldsymbol{v})) = \mu$ 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 $\boldsymbol{v}_1 \cdot \mathrm{pre}(l_b(\boldsymbol{v}_1, \boldsymbol{v}), |\boldsymbol{v}_2| - |\boldsymbol{v}_1|) \overset{D}{=} \boldsymbol{v}_2$, $T_{|\boldsymbol{v}_2|}(\boldsymbol{v}_1 \cdot \mathrm{pre}(l_b(\boldsymbol{v}_1, \boldsymbol{v}), |\boldsymbol{v}_2| - |\boldsymbol{v}_1|)) = \boldsymbol{v}_2$ by CT (1) and P1, which contradicts CE (2). Therefore, $|\boldsymbol{v}| = |\boldsymbol{v}_2|$. Since $\boldsymbol{v}_1 \cdot l_b(\boldsymbol{v}_1, \boldsymbol{v}) \overset{D}{=} \boldsymbol{v}_2$, $T_{|\boldsymbol{v}_2|}(\boldsymbol{v}_1 \cdot l_b(\boldsymbol{v}_1, \boldsymbol{v})) = \boldsymbol{v}_2$ by CT (1) and P1. On the other hand, $T_{|\boldsymbol{v}_2|}(\boldsymbol{v}_1 \cdot l_b(\boldsymbol{v}_1, \boldsymbol{v})) = \boldsymbol{v}$ by CE (1). Hence, $\boldsymbol{v} = \boldsymbol{v}_2$. This means $(\boldsymbol{v}_1, \boldsymbol{v}_2, b) \in E(S)$. Thus, $\langle 3 \rangle$ holds. □

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. 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 $(S, T_1, ..., T_{m+1}, \boldsymbol{e}, D(\boldsymbol{e}))$. Then, S' is an OMTBDDAS, $(S', T'_1, \ldots, T'_m)$ satisfies CN, CT and CE, and $|V(\mathcal{D}(S'))| = |V(\mathcal{D}(S))| + 1$.*

Proof. One execution of Algorithm Update-Hypothesis adds just one node to S because it executes algorithm NodeSplit or NewBranchingNode just once, either of which adds just one non-dummy node to S. Thus, $|V(\mathcal{D}(S'))| = |V(\mathcal{D}(S))| + 1$ holds trivially.

It is also trivial that the node-labeled edge-labeled graph structure of S' satisfies the OMTBDDAS definition from the way of updating S in algorithms

NodeSplit and NewBranchingNode. All the node ids in S' are still access strings of the corresponding nodes in $\mathcal{D}(S')$ because an id $\boldsymbol{v}$ of each node is constructed such that $\boldsymbol{v}$ is a concatenation of the edge label strings on one path from the node labeled x_0 to node $\boldsymbol{v}$ and the id of every node passed through by the path is a prefix of $\boldsymbol{v}$. Though algorithm NodeSplit may break some path including the edge labeled $l_{e_{|\boldsymbol{p}_i|}}(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$, the path made by any node id in S does not pass through the edge because, if the path made by some node id passes through the edge, $\boldsymbol{p}_i l_{e_{|\boldsymbol{p}_i|}}(\boldsymbol{p}_i, \boldsymbol{p}_{i+1}) = \boldsymbol{p}_{i+1}$ must hold, which contradict the fact that NodeSplit is executed.

In the following, we prove that, for $(S', T'_1, \ldots, T'_m)$, two conditions CN and CT hold first, then condition CE holds next. .

(1) Conditions CN&CT satisfaction for $(S', T'_1, \ldots, T'_m)$

In NodeSplit, the new node $\boldsymbol{v} = \boldsymbol{p}_i l_{e_{|\boldsymbol{p}_i|}}(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$ is added to S at Line 2. For this $\boldsymbol{v}$, $\boldsymbol{v} \in \mathrm{nodes}(D)$ is implied by CT (2) because $T_{|\boldsymbol{v}|}(\boldsymbol{v}) = T_{|\boldsymbol{v}|}(\boldsymbol{p}_i l_{e_{|\boldsymbol{p}_i|}}(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})) = \boldsymbol{p}_{i+1} \neq \mu$ holds by CE (1), thus CN (1) holds for S'. If $\boldsymbol{v}$ is not a sink, $V_m(S') = V_m(S)$ and thus CN (2) holds for S' by the assumption for S. If $\boldsymbol{v}$ is a sink, $\mathcal{D}(S')(\boldsymbol{v}) = \ell = D(\boldsymbol{v})$ holds, thus CN(2) holds for the new sink $\boldsymbol{v}$. We can show $\boldsymbol{v} \overset{D}{\neq} \boldsymbol{v}'$ for any $\boldsymbol{v}' \in V(S)$ as follows. It is trivial if $|\boldsymbol{v}| \neq |\boldsymbol{v}'|$, thus 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 S'. Therefore, CN is satisfied by $(S', T'_1, \ldots, T'_m)$. As for node classification trees, only the $\boldsymbol{p}_{i+1}$-labeled leaf is replaced with an $\boldsymbol{e}_{i+1}$-labeled single-test node with its child leaves labeled $\boldsymbol{v}$ and $\boldsymbol{p}_{i+1}$, Thus $T'_{|\boldsymbol{v}|}(\boldsymbol{u}) = \boldsymbol{u}$ holds for all $\boldsymbol{u} \in V_{|\boldsymbol{v}|}(S')$. Since $\{\boldsymbol{a} \mid T_i(\boldsymbol{a}) = \mu\} = \{\boldsymbol{a} \mid T'_i(\boldsymbol{a}) = \mu\}$, CT (2) still holds for T'_i $(i = 1, 2, \ldots, m)$. Thus, CT is also satisfied by $(S', T'_1, \ldots, T'_m)$.

In the algorithm NewBranchingNode, the new node $\boldsymbol{v}$ is added to S at Line 5 or 13. In this case, $\boldsymbol{v} \in \mathrm{nodes}(D)$ 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 S'. Since $\boldsymbol{v}$ cannot be a sink because $j < |\boldsymbol{q}|$, CN (2) still holds for 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 S'. Trivially, $T'_{|\boldsymbol{v}|}(\boldsymbol{v}) = \boldsymbol{v}$ 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 $(D(\boldsymbol{vr}), D(\boldsymbol{v\dot{r}}))$ to the leaf labeled $\boldsymbol{v}$. Since no other update is done to $T_{|\boldsymbol{v}|}$, so $T'_{|\boldsymbol{v}|}(\boldsymbol{v}') = T_{|\boldsymbol{v}|}(\boldsymbol{v}') = \boldsymbol{v}'$ holds for $\boldsymbol{v}' \in V_{|\boldsymbol{v}|}(S)$. Therefore, CT (1) holds for $T'_1, T'_2, \ldots, T'_m$. $T'_{|\boldsymbol{a}|}(\boldsymbol{a}) = T(\boldsymbol{a})$ holds for all $\boldsymbol{a} \in \{0,1\}^i, i = 1, \ldots, m$ except for some $\boldsymbol{a} \in \{0,1\}^{|\boldsymbol{v}|}$ satisfying $\mu = T_{|\boldsymbol{v}|}(\boldsymbol{a}) \neq T'_{|\boldsymbol{v}|}(\boldsymbol{a}) = \boldsymbol{v}$, which belongs to nodes$(D)$ because $D(\boldsymbol{ar}) \neq D(\boldsymbol{a\dot{r}})$ holds for the label $\boldsymbol{r}$ of the last twin-test node on the path. Thus CT (2) still holds for $T'_{|\boldsymbol{v}|}$. Therefore, CT holds for $(S', T'_1, \ldots, T'_m)$.

(2) Condition CE satisfaction 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 μ-labeled 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 $(\boldsymbol{u}, \boldsymbol{v}, b) \in E(S') \cap E(S)$. Let $\boldsymbol{v}$ be the new node added to S at Line 2. Then, $T_{|\boldsymbol{v}|}$ is the only node classification tree that is changed in NodeSplit. In $T_{|\boldsymbol{v}|}$, the leaf labeled $\boldsymbol{p}_{i+1}$ is replaced with $T^{\boldsymbol{e}_{i+1}}_{|\boldsymbol{v}|}$ at Line 8. Thus, edges $(\boldsymbol{v}', \boldsymbol{p}_{i+1}, b)$ with $T_{|\boldsymbol{v}|}(\boldsymbol{v}' l_b(\boldsymbol{v}', \boldsymbol{p}_{i+1})) = \boldsymbol{p}_{i+1}$ and $T'_{|\boldsymbol{v}|}(\boldsymbol{v}' l_b(\boldsymbol{v}', \boldsymbol{p}_{i+1})) = \boldsymbol{v} \neq \boldsymbol{p}_{i+1}$ must be removed and new edges $(\boldsymbol{v}', \boldsymbol{v}, , b)$ must be added. All such edge removals and additions are done at Lines 2 and 7. Thus, all the edges $(\boldsymbol{v}', \boldsymbol{p}_{i+1}, b)$ that do not satisfy CE (1) are removed and all the added edges $(\boldsymbol{v}', \boldsymbol{v}, b)$ satisfy CE (1). Since the label strings $l_b(\boldsymbol{v}', \boldsymbol{v})$ for the added edges $(\boldsymbol{v}', \boldsymbol{v}, b)$ are the same as the label strings $l_b(\boldsymbol{v}', \boldsymbol{p}_{i+1})$ for the removed edges $(\boldsymbol{v}', \boldsymbol{p}_{i+1}, b)$ and $T'_i = T_i$ for $i \neq |\boldsymbol{v}|$, CE (2) still holds for new edges $(\boldsymbol{v}', \boldsymbol{v}, b)$. Other two new edges $(\boldsymbol{v}, \boldsymbol{u}, b)$ are added through AddEdge at Line 11 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, $T_{|\boldsymbol{v}|}$ is the only node classification tree that is changed in NewBranchingNode. In $T_{|\boldsymbol{v}|}$, the leaf labeled μ is replaced with $T^{\boldsymbol{r}}_{|\boldsymbol{v}|}$ at Line 11. Thus, edges $(\boldsymbol{u}, \boldsymbol{v}', b)$ with $|\boldsymbol{u}| < |\boldsymbol{v}| < |\boldsymbol{v}'|$, $T_{|\boldsymbol{v}|}(\boldsymbol{u}\mathrm{pre}(l_b(\boldsymbol{u}, \boldsymbol{v}'), |\boldsymbol{v}| - |\boldsymbol{u}|)) = \mu$ and $T'_{|\boldsymbol{v}|}(\boldsymbol{u}\mathrm{pre}(l_b(\boldsymbol{u}, \boldsymbol{v}'), |\boldsymbol{v}| - |\boldsymbol{u}|)) = \boldsymbol{v}$ must be removed and new edges $(\boldsymbol{u}, \boldsymbol{v}, b)$ must be added. All such edge removals and additions are done at Lines 5 and 10. Thus, all the edges $(\boldsymbol{u}, \boldsymbol{v}', b)$ that do not satisfy CE (2) are removed and all the added edges $(\boldsymbol{u}, \boldsymbol{v}, b)$ satisfy CE (1). Since the label strings $l_b(\boldsymbol{u}, \boldsymbol{v})$ for the added edges $(\boldsymbol{u}, \boldsymbol{v}, b)$ are the same

as the label strings $\mathrm{pre}(l_b(\boldsymbol{u}, \boldsymbol{v}'), |\boldsymbol{v}| - |\boldsymbol{u}|)$ for the removed edges $(\boldsymbol{u}, \boldsymbol{v}', b)$ and $T'_j = T_j$ for $j \neq |\boldsymbol{v}|$, CE (2) still holds for the added edges $(\boldsymbol{u}, \boldsymbol{v}, b)$. Edge $(\boldsymbol{v}, \boldsymbol{p}_{i+1}, \dot{e}_{|\boldsymbol{v}|})$ is also added at Line 5. Since $\boldsymbol{v}l_{\dot{e}_{|\boldsymbol{v}|}}(\boldsymbol{v}, \boldsymbol{p}_{i+1})$ in S' is equal to $\boldsymbol{p}_i l_{e_{|\boldsymbol{p}_i|}}(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$ in S and $T_{|\boldsymbol{p}_{i+1}|}$ does not change in NewBranchingNode, $T'_{|\boldsymbol{p}_{i+1}|}(\boldsymbol{v}l_{\dot{e}_{|\boldsymbol{v}|}}(\boldsymbol{v}, \boldsymbol{p}_{i+1}))$ for S' is equal to $T_{|\boldsymbol{p}_{i+1}|}(\boldsymbol{p}_i l_{e_{|\boldsymbol{p}_i|}}(\boldsymbol{p}_i, \boldsymbol{p}_{i+1}))$ for S, thus $T'_{|\boldsymbol{p}_{i+1}|}(\boldsymbol{v}l_{\dot{e}_{|\boldsymbol{v}|}}(\boldsymbol{v}, \boldsymbol{p}_{i+1})) = \boldsymbol{p}_{i+1}$ by CE (1) for $(S, T_1, \ldots, T_m)$. Since the label string $l_{\dot{e}_{|\boldsymbol{v}|}}(\boldsymbol{v}, \boldsymbol{p}_{i+1})$ for the added edges $(\boldsymbol{v}, \boldsymbol{p}_{i+1}, \dot{e}_{|\boldsymbol{v}|})$ is the same as some suffix of the label string $\mathrm{suf}(l_{e_{|\boldsymbol{p}_i|}}(\boldsymbol{p}_i, \boldsymbol{p}_{i+1}), |\boldsymbol{p}_{i+1}| - |\boldsymbol{v}|)$ for the removed edge $(\boldsymbol{p}_i, \boldsymbol{p}_{i+1}, e_{|\boldsymbol{p}_i|})$ and no T_j for $j \neq |v|$ changes, CE (2) still holds for the added edge $(\boldsymbol{v}, \boldsymbol{p}_{i+1}, \dot{e}_{|\boldsymbol{v}|})$. Other new edges $(\boldsymbol{v}, \boldsymbol{u}, e_{|\boldsymbol{v}|})$ are added at Line 14 through AddEdge and that edge is made so as to satisfy CE (1) and CE (2). Thus, CE is satisfied by $(S', T'_1, \ldots, T'_m)$. □

Proof of Theorem 1. First, we prove that QLearn-OMTBDD outputs D with at most n equivalence queries. Let $\boldsymbol{e}'$ denote the first counterexample. Then, the initial OMTBDD S is composed of a dummy node λ labeled x_0, a leaf node $\boldsymbol{e}'$ labeled $D(\boldsymbol{e}')$, and an edge between them labeled $\boldsymbol{e}'$. The initial node classification trees T_i are the one-node trees with L-labeled leaves, where $L = \boldsymbol{e}'$ for $i = m$ and $L = \mu$ otherwise. The initial OMTBDD S and node classification trees $T_1, \ldots, T_m$ satisfy conditions CN, CT and CE trivially. For the counterexample $\boldsymbol{e}$ for $\mathcal{D}(S)$ of the initial OMTBDDAS S, strings $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ are set to λ and $\boldsymbol{e}'$, respectively at Line 1 of Update-Hypothesis. NewBranchingNode is executed because $D(\boldsymbol{p}_1\boldsymbol{e}) = D(\boldsymbol{e}) = \ell \neq \mathcal{D}(S)(\boldsymbol{e}) = D(\boldsymbol{e}') = D(\boldsymbol{p}_1 l_{e_0}(\boldsymbol{p}_1, \boldsymbol{p}_2))$. In NewBranchingNode, one non-sink node $\boldsymbol{v}$ is inserted between the dummy and leaf nodes by partitioning the edge between them into two edges $(\lambda, \boldsymbol{v}, e'_0)$ and $(\boldsymbol{v}, \boldsymbol{e}', e'_{|\boldsymbol{v}|})$ labeled $\mathrm{pre}(\boldsymbol{e}', |\boldsymbol{v}|)$ and $\mathrm{suf}(\boldsymbol{e}', m - |\boldsymbol{v}|)$. Furthermore, one more edge to the sink labeled $D(\boldsymbol{e}')$, edge $(\boldsymbol{v}, \boldsymbol{e}', e_{|\boldsymbol{v}|})$ labeled $\mathrm{suf}(\boldsymbol{e}, m - |\boldsymbol{v}|)$, is added to S through the algorithm AddEdge. The OMTBDDAS S and node classification trees $T_1, \ldots, T_m$ returned by Update-Hypothesis of the first call satisfy conditions CN, CT and CE and the number of non-dummy nodes in S is two by Lemma 2. For this OMTBDDAS S, $\boldsymbol{e}$ is still a counterexample, thus Update-Hypothesis is called again without asking an equivalence query.

By Lemma 2, the number of non-dummy nodes of the OMTBDDAS S maintained by the algorithm increases by just 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, $|V(\mathcal{D}(S))|$ reaches just the number of nodes in D after executing the algorithm Update-Hypothesis at

most $n-1$ times. Then, Lemma 1 guarantees that $\mathcal{D}(S) = D$. Therefore, for $n \geq 3$, the n'th equivalence query in QLearn-OMTBDD for some $n' \leq n$ is answered with 'YES' because two equivalence queries are asked before the first execution of Update-Hypothesis, no equivalence query is asked right after its first execution, and at most one equivalence query is asked right after each execution of Update-Hypothesis.

Next, we consider the number of membership queries. To construct an initial OMTBDDAS and node classification trees, the algorithm uses one membership query. Let us consider how many membership queries are asked in one execution of Update-Hypothesis. In Update-Hypothesis, at most $\lceil \log_2 m \rceil$ membership queries are asked at Line 2 and one more membership query is asked at Line 3. Let $\boldsymbol{v}$ be the new node added in NodeSplit or NewBranchingNode and let n_1 and n_2 denote the number of nodes $\boldsymbol{u}$ in S with $|\boldsymbol{u}| < |\boldsymbol{v}|$ and $|\boldsymbol{u}| > |\boldsymbol{v}|$, respectively. Note that $n_1 + n_2 \leq n - 1$.

Case with NodeSplit execution in Update-Hypothesis

At Line 5 of NodeSplit, at most n_1 membership queries are asked because at most one membership query is asked for each node $\boldsymbol{u}$ with $|\boldsymbol{u}| < |\boldsymbol{v}|$ in S. In each execution of Line 11 of NodeSplit, at most $4n_2$ membership queries are asked because at most two membership queries are asked for each internal node of $T_{|\boldsymbol{v}|+1}, ..., T_m$ in AddEdge and the number of internal nodes is at most n_2. In this case of Update-Hypothesis execution, the algorithm asks at most $\lceil \log_2 m \rceil + 1 + n_1 + 4n_2 < \lceil \log_2 m \rceil + 4n$ membership queries in total.

Case with NewBranchingNode execution in Update-Hypothesis

At Line 2 of NewBranchingNode, at most $\lceil \log_2 m \rceil$ membership queries are asked by using a binary search. At Line 9 of NewBranchingNode, at most $4n_1$ membership queries are asked because at most two membership queries are asked for each edge in S that goes out from nodes $\boldsymbol{u}$ with $|\boldsymbol{u}| < |\boldsymbol{v}|$. At Line 14 of NewBranchingNode, at most $2n_2$ membership queries are asked in AddEdge for the reason described above. In this case of Update-Hypothesis execution, the algorithm asks at most $2\lceil \log_2 m \rceil + 1 + 4n_1 + 2n_2 < 2\lceil \log_2 m \rceil + 4n$ membership queries in total .

Thus, QLearn-OMTBDD asks less than $2n(\lceil \log_2 m \rceil + 2n)$ membership queries. □

Appendix B. Random OMTBDD Generation Procedure

In our experiment for a 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 $x_0 < x_1 < \cdots < x_{m-1}$.

1: $n', n'' \leftarrow n$
2: **repeat**
3: $n' \leftarrow n' + (n - n'')$
4: Select $n' - K$ variables from $\{x_0, \ldots, x_{m-1}\}$ 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** $\exists h$ s.t. $\mathrm{n\_var}(j) \leq h < \mathrm{n\_var}(\mathrm{n\_var}(j))$ and v_h has no incoming edge $\{\mathrm{n\_var}(j) = \min(\{h \mid i_h > i_j\} \cup \{n' - K + 1\})\}$ **then**
11: Add edge (v_j, v_h) for h that is randomly selected from $\{\mathrm{n\_var}(j), \ldots, \mathrm{n\_var}(\mathrm{n\_var}(j))\}$ 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 $\{\mathrm{n\_var}(j), \ldots, n\}$ 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.