An efficient query learning algorithm for zero-suppressed binary decision diagrams

Hayato Mizumoto HAYATO_MIZUMOTO@SHINO.ECEI.TOHOKU.AC.JP
Shota Todoroki SHOTA_TODOROKI@SHINO.ECEI.TOHOKU.AC.JP
Diptarama DIPTARAMA@SHINO.ECEI.TOHOKU.AC.JP
Ryo Yoshinaka RY@ECEI.TOHOKU.AC.JP
Ayumi Shinohara AYUMI@ECEI.TOHOKU.AC.JP
Graduate School of Information Sciences, Tohoku University
6-6-05, Aramakiaza Aoba, Aoba-ku, Sendai City, Miyagi, Japan

Editors: Steve Hanneke and Lev Reyzin

Abstract

A ZDD is a directed acyclic graph that represents a family of sets over a fixed universe set. In this paper, we propose an algorithm that learns zero-suppressed binary decision diagrams (ZDDs) using membership and equivalence queries. If the target ZDD has n nodes and the cardinality of the universe is m, our algorithm uses n equivalence queries and at most $n(\lfloor \log m \rfloor + 4n)$ membership queries to learn the target ZDD.

Keywords: Zero-suppressed binary decision diagrams; Query learning; MAT learning

1. Introduction

A *zero-suppressed binary decision diagram (ZDD)* (Minato, 1993) is a data structure that can efficiently compress and represent large-scale combination data over a universe X. An important virtue of ZDDs is that various set operations can be performed efficiently on ZDDs. This paper proposes an efficient algorithm for learning ZDDs under Angluin's query learning model (Angluin, 1987). In her model a learner can ask two types of queries, called *equivalence queries (EQs)* and *membership queries (MQs)*. An EQ asks whether a learner's hypothesis, which is a ZDD in our case, represents the learning target. If it is the case, the learning is done. Otherwise, the learner gets a counterexample. An MQ asks whether a datum, which is a subset of X in our case, belongs to the learning target.

There are data structures somewhat similar to ZDDs, that can be used for representing families of sets. One possible approach to learn ZDDs via queries would be to use existing learning algorithms for those data structures. Since Angluin proposed the query learning model, *deterministic finite state automata (DFAs)* have been a major target. A family F of subsets of $X = \{x_1, \dots, x_m\}$ can be represented as a set L_F of binary strings such that $b_1 \dots b_m \in L_F$ if and only if $\{\, x_i \mid b_i = 1 \,\} \in F$. The algorithm by Rivest and Schapire (1993) learns DFAs corresponding to such families of sets with n_A EQs and $O(n_A(n_A + \log m))$ MQs, where n_A denotes the size of the learning target DFA. No algorithms with lower query complexity have been proposed so far. Another data structure can be used to represent a family of sets is a *binary decision diagram (BDD)* (Bryant, 1986). Actually, ZDDs are closer to BDDs than DFAs and ZDDs were originally proposed based on BDDs. BDDs are supposed to represent Boolean functions, but through a straightforward translation, one can assume that they represent families of sets. BDDs tend to be much more compact than DFAs if data sets to represent include “symmetric” substructures. Nakamura (2005) has proposed an efficient algorithm that learns BDDs with n_B EQs and $O(n_B(n_B + \log m))$ MQs, where n_B is the size of the learning target BDD, based on classical algorithms for DFAs by Rivest and Schapire (1993) and Kearns and Vazirani (1994). However, when representing the same data (modulo the above trivial translation), those numbers n_A, n_B and n_Z, the size of the ZDD, can often be significantly different. For any $n_1 \in \{n_A, n_B, n_Z\}$ and $n_2 \in \{n_B, n_Z\}$, it holds that $n_1 \leq n_A$ and $n_1 \leq (m+1)(n_2+1)/2$ (Knuth, 2009), and there are data for which we have $n_A \geq m(n_B - 1)/2$ (Nakamura, 2005). Using those existing learning algorithms for learning ZDDs requires $O(mn_Z)$ EQs and $O((mn_Z)(mn_Z + \log m))$ MQs in the worst case. ZDDs tend to be much more compact than DFAs and BDDs when the data sets to represent are large but sparse. Therefore, using existing algorithms does not seem to be the best solution for learning ZDDs.

In this paper, we propose an algorithm for learning ZDDs following Nakamura's for BDDs. It uses n_Z EQs and $O(n_Z(n_Z + \log m))$ MQs, which achieves the same query complexity as the existing best algorithms for DFAs and BDDs. Thus, our algorithm has an advantage when learning data for which a ZDD is more suitable than a DFA or a BDD.

2. Preliminaries

The set of strings of length k over an alphabet Σ is denoted by Σ^k. We let $\Sigma^* = \bigcup_{k \geq 0} \Sigma^k$, $\Sigma^+ = \bigcup_{k>0} \Sigma^k$, $\Sigma^{\leq m} = \bigcup_{k \leq m} \Sigma^k$, and $\Sigma^{<m} = \bigcup_{k<m} \Sigma^k$. We allow to write a^k to mean the unique element of $\{a\}^k$, while a^* means $\{a\}^*$ for $a \in \Sigma$. The length of a string p is denoted by $|p|$. The empty string is denoted by ε. For a string $p = a_1 \dots a_k \in \Sigma^k$, we write $\mathrm{pre}(p, i)$ for the prefix $a_1 \dots a_i$ of length i and $\mathrm{suf}(p, i)$ for the suffix $a_{k-i+1} \dots a_k$ of length i. In this paper, we treat binary strings over $\{0, 1\}$, where, for example, 10^i means the string where 0 follows i times after the first occurrence of 1, rather than the ith power of ten.

2.1. Zero-suppressed binary decision diagram

A zero-suppressed binary decision diagram (ZDD) (Minato, 1993) is a data structure for representing a family of sets over a fixed totally ordered universe $X = \{x_1, \dots, x_m\}$ in a compact manner. A ZDD is formally a directed acyclic graph such that

- it has exactly one node of in-degree 0, called the *root*,
- it has exactly two nodes $\bot$ and $\top$ of out-degree 0, called $\bot$-*terminal* and $\top$-*terminal*, respectively,
- every internal node has 2 outgoing edges called the *0-edge* and *1-edge*, where the node directed from a node by its b-edge is called the b-*child* for $b \in \{0, 1\}$,
- every internal node v is labeled by an element of the universe,
- if a node and a child of it are labeled with x_i and x_j, respectively, then $i < j$.

A path π from a node to a descendant in a ZDD represents a subset of X

$$[\![\pi]\!] = \{\, x_i \in X \mid \text{the 1-edge of a node labeled with } x_i \text{ is on } \pi \,\}$$

and a ZDD D represents a family $[\![D]\!]$ of subsets of X by

$$[\![D]\!] = \{\, [\![\pi]\!] \mid \pi \text{ is a path from the root to the } \top\text{-terminal of } D \,\} .$$

There can be different ZDDs that represent the same family of sets. We call a ZDD *reduced* if it satisfies the following properties:

- there are no distinct nodes whose 0-children are identical and 1-children are identical,
- there is no node whose 1-child is the $\bot$-terminal.

If a ZDD violates the first condition, such two nodes can be merged. If a ZDD violates the second condition, such a node v can be suppressed and the edges pointing to v can be redirected to the 0-child of v. Let us call the former *merging reduction* and the latter *zero-suppression reduction*. Those reduction operations do not change the semantics of the ZDD. It is known that for any family of sets over X with a fixed ordering, a reduced ZDD is minimum and unique up to isomorphism (Minato, 1993).

One can represent a subset Y of X by a binary string $b_1 \dots b_m \in \{0,1\}^m$ such that $b_i = 1$ iff $x_i \in Y$. Moreover by identifying a ZDD D and the indicator function $\{0,1\}^m \to \{\bot, \top\}$ of the represented set family, we write $D(b_1 \dots b_m) = \top$ if the corresponding set is in $[\![D]\!]$. Moreover, we say that a binary string $b_1 \dots b_k \in \{0,1\}^{\le m}$ *reaches* a node v in D if there is a path π from the root to v such that $[\![\pi]\!] = \{\, x_i \in X \mid b_i = 1 \,\}$ and v is labeled with x_{k+1}, assuming that the terminal nodes are labeled with x_{m+1}. The set of strings that reach some nodes in D is denoted as $\mathrm{Reach}(D)$. For two binary strings $p, q \in \{0,1\}^{\le m}$ of the same length, we write $p \approx_D q$ if $D(pr) = D(qr)$ for any $r \in \{0,1\}^{m-|p|}$.

Fig. 3 shows a ZDD D_* on the left bottom corner, for which $[\![D_*]\!] = \{\{x_3\}, \{x_1, x_2\}, \{x_1, x_3\}, \{x_1, x_2, x_3\}\}$.

Lemma 1 *Let D be a reduced ZDD and $p \in \{0,1\}^{<m}$ a binary string. We have $p \in \mathrm{Reach}(D)$ if and only if there is r such that $D(p1r) = \top$. For any $p_1, p_2 \in \mathrm{Reach}(D)$, p_1 and p_2 reach the same node if and only if $p_1 \approx_D p_2$. The number of nodes of D is exactly the number of equivalence classes in $\mathrm{Reach}(D)/\approx_D$, unless $[\![D]\!] = 2^X$ or $[\![D]\!] = \emptyset$.*

2.2. Query learning

We will work under the learning model proposed by Angluin (1987). Hereafter we fix our learning target to be a reduced ZDD D_*. A learner can ask two types of queries to the teacher, called an *equivalence query (EQ)* and a *membership query (MQ)*. An EQ instance is a ZDD D and the teacher answers Yes if $D(p) = D_*(p)$ for all $p \in \{0,1\}^m$. Otherwise, the teacher gives a counterexample $r \in \{0,1\}^m$ such that $D(r) \neq D_*(r)$. An MQ instance is an arbitrary string $p \in \{0,1\}^m$, to which the teacher answers whether $D_*(p) = \top$. Through communication with the teacher, the learner must identify D_*.

3. Data Structures for Learning ZDDs

Following Nakamura (2005), we use two types of auxiliary data structures when learning a ZDD. One is an augmentation of a ZDD, which we call a *ZDD with access strings (ZDDAS)*, and the other is a collection T of binary trees $T_0, \dots, T_m$, called *classification forest/trees*. Those data structures will be monotonically expanded during learning.

3.1. ZDD with Access Strings

A ZDDAS S is obtained from a ZDD D with the following augmentation.

- If the root of D is not labeled with x_1, we add a *dummy* root node with label x_1 in S. The dummy root of S has only one child, which is the root of D, connected by a 0-edge.
- Edges have an extra label. If a b-edge goes from a node labeled with x_i to another with x_j, the edge is labeled with $b0^{j-i-1}$. In this labeling, we assume that the terminal nodes are labeled with x_{m+1}.
- Each internal node v labeled with x_i has another label of a binary string $p \in \{0,1\}^{i-1}$ called an *access string* and no distinct nodes have the same access label.
- The access strings of the $\bot$-terminal and $\top$-terminal nodes are $\bot$ and $\top$, respectively.

By stripping the extra labels and removing the dummy node (if exists) from S, one obtains the original ZDD D, which we will denote by $\tilde{S}$. Our algorithm maintains a ZDDAS S instead of a ZDD and asks an EQ on $\tilde{S}$. Since no distinct nodes can have the same access string, we will use the access string as the name of a node in S. Note that the extra label of each edge is determined by the labels of the nodes, so this is redundant but convenient to decide the node where p reaches without paying attention to the labels x_i on nodes. On the other hand, the choice of an access string is arbitrary, so different ZDDAS may correspond to the same ZDD, but those strings give us a guide to grow the ZDDAS. We would like S to finally satisfy that for any string $p \in \{0,1\}^{<m}$ and any internal node q in S

- p reaches q in S if and only if $p \approx_{D_*} q$

and that for any string $p \in \{0,1\}^m$

- p reaches $\top$ in S if and only if $D_*(p) = \top$.

Note that we do not guarantee that $\tilde{S}$ is reduced in the middle of learning, but it will be reduced at the end of learning.

We denote an edge in S labeled with q that goes from a node p to r by a triple (p, q, r). The sets of nodes and edges of S are denoted by V and E, respectively. We define $V_i = V \cap \{0,1\}^i$ for $i < m$ and $V_m = \{\bot, \top\}$. At the beginning of learning, our algorithm has a ZDDAS with only a dummy and the terminal nodes. We add a non-dummy node with label p to V only when we find an evidence that shows

- $p \in \mathrm{Reach}(D_*)$, which is witnessed by a string r such that $D_*(p1r) = \top$ (thus zero-suppression reduction does not apply to p),
- $q \not\approx_{D_*} p$ for every $q \in V_{|p|}$, which is witnessed by a string r_q such that $D_*(pr_q) \neq D_*(qr_q)$ (thus merging reduction does not apply to p and q).

3.2. Classification Trees

For each $0 \leq i \leq m$, we have a binary tree called a *classification tree* T_i. We will use classification trees to determine where outgoing edges of a node in S should point. Each T_i works as a function from $\{0,1\}^i$ to $V_i \cup \{\mu\}$, where μ is a special symbol different from any of $0, 1, \bot, \top$. It will be guaranteed that $T_i(p) = q$ if $p \approx_{D_*} q$ and $q \in V$, and that $T_i(p) = \mu$ if $p \notin \mathrm{Reach}(D_*)$.

Formally, each T_i for $0 < i < m$ satisfies the following properties (see Fig. 3).

- Each internal node is labeled by a string of length $m - i$.
- Each internal node has two children, called $\bot$-*child* and $\top$-*child*.

- T_i has one special leaf node labeled by the special symbol μ, which we call the *μ-leaf.*
- Every internal node on the path from the root to the μ-leaf is labeled with a string starting with 1.
- Every node on the path from the root to the μ-leaf except the root is a $\bot$-child of its parent.
- Each leaf node but the μ-leaf is labeled by an element of V_i.
- No distinct leaves in T_i have the same label.

On the other hand, T_0 is just a node, which is labeled either μ (when the root of S is a dummy) or ε (otherwise). T_m has just three nodes and no μ-leaf. The root of T_m is labeled with ε and its $\bot$-child and $\top$-child are labeled with $\bot$ and $\top$, respectively.

Each classification tree T_i classifies strings p of length i in the following manner with the aid of MQs. We start from the root of T_i. If the current internal node is labeled with q, we ask the MQ on pq and go to the $D_*(pq)$-child. When we reach a leaf node, return the node label. Therefore, classifying a string $p \in \{0,1\}^m$ by T_m is nothing but an MQ on p. For $0 < i < m$, we get $T_i(p) \neq \mu$ if and only if we find a witness $r \in 1\{0,1\}^{m-i-1}$ such that $D_*(pr) = \top$ on the path from the root to μ. We remark that to know the value $T_i(p)$ for $p \in \{0,1\}^i$, one requires at most $|V_i|$ MQs.

Since distinct T_i and T_j have disjoint domains, we can omit the subscript and simply write $T(p)$ for $T_i(p)$ and call T the *classification forest.*

3.3. Invariant Properties

When learning D_*, our algorithm maintains the ZDDAS S and classification forest T so that they satisfy the following invariant properties.

C1. (1) $\hat{V} \subseteq \mathrm{Reach}(D_*)$, where $\hat{V}$ is obtained by removing a dummy and terminal nodes from V,
(2) $p \not\approx_{D_*} q$ for any distinct $p, q \in \hat{V}$,

C2. (1) $T(p) = p$ for all $p \in \hat{V}$,
(2) $T(p) = \mu$ for any $p \in \{0,1\}^{<m} \setminus \mathrm{Reach}(D_*)$,

C3. $(p, b0^k, T(pb0^k)) \in E$ for the smallest k satisfying $T(pb0^k) \neq \mu$, for each $p \in \hat{V}$ and $b \in \{0,1\}$. Moreover, if ε is a dummy node, $(\varepsilon, 0^k, T(0^k)) \in E$ for the smallest k satisfying $T(0^k) \neq \mu$.

Lemma 2 *If our ZDDAS S has n non-dummy nodes, where n is the number of nodes of D_*, then $\tilde{S}$ is equivalent to D_*.*

Proof For each node $p \in \hat{V}$, let $\psi(p)$ be the node of D_* where p reaches. By C1(1), this is well-defined and $\psi(p)$ is labeled with $x_{|p|+1}$ in D_*. Moreover, define $\psi(\bot) = \bot$ and $\psi(\top) = \top$.[1] Then ψ is a bijection from non-dummy nodes of S to the nodes of D_* by C1(2).

To prove $\tilde{S}$ and D_* are isomorphic, we show that for any $p \in \hat{V}$ and $q \in V$, if D_* has a b-edge from $\psi(p)$ to $\psi(q)$, then $(p, b0^j, q) \in E$ for $j = |q| - |p| - 1$, assuming $|\bot| = |\top| = m$. Suppose q is not a terminal node. Since both $pb0^j$ and q reach the same node $\psi(q)$ in D_*, we have $pb0^j \approx_{D_*} q$. By the definition of classification by T, we have $T(pb0^j) = T(q)$, and this is actually $T(pb0^j) = T(q) = q$ by C2(1). In addition, for any $j' < j$, $pb0^{j'} \notin \mathrm{Reach}(D_*)$, which implies $T(pb0^{j'}) = \mu$ by C2(2). Therefore, we conclude $(p, b0^j, q) \in E$ by C3.

1. We use the same letters to represent the terminal nodes of both S and D_*.

Suppose that D_* has a b-edge from $\psi(p)$ to a terminal $B \in \{\perp, \top\}$. Then $D_*(pb0^{m-|p|-1}) = B$. By definition $T_m(pb0^{m-|p|-1}) = B$. In addition, for any $j' < j$, $pb0^{j'} \notin \mathrm{Reach}(D_*)$, which implies $T(pb0^{j'}) = \mu$ by C2(2). Therefore, we conclude $(p, b0^j, B) \in E$ by C3. ■

4. Algorithm

We present our learning algorithm in this section, whose pseudo codes are shown as Algorithms 1 to 4, where S and T are treated as global variables. A running example is described in Section 5.

Algorithm 1: Learning ZDDs

```
Let $(S, T) :=$ Initial-Hypothesis($D_*(0^m)$);
Let $A := \mathrm{EQ}(\tilde{S})$;
while $A \neq$ Yes do
    call Update-Hypothesis($A$);
    Let $A := \mathrm{EQ}(\tilde{S})$;
end
output $\tilde{S}$;
```

4.1. Initial Hypothesis

Each classification tree T_i for $i < m$ is initialized to be the tree with a single node labeled μ, while T_m is a tree with three nodes respectively labeled with $\varepsilon, \top$ and $\perp$. The initial ZDDAS has only a dummy node and the terminal nodes, $V = \{\varepsilon, \perp, \top\}$. By making the first MQ on 0^m, we let $E = \{(\varepsilon, 0^m, D_*(0^m))\}$.

4.2. Updating Hypothesis

Suppose the learner has got a counterexample $r \in \{0,1\}^m$ to an EQ on $\tilde{S}$. If our ZDDAS S has a dummy node and r starts with 1, this means that $S(r) = \perp \neq D_*(r) = \top$ and $\varepsilon \in \mathrm{Reach}(D_*)$. In this case, we promote ε to be non-dummy. Then we need to give a 1-edge to this node ε. We search for the smallest j such that $T(10^j) \neq \mu$ and add $(\varepsilon, 10^j, T(10^j))$ to E. The unique node of the classification tree T_0 is relabeled with ε.

Suppose otherwise. The string r may but not necessarily reach a terminal node in S. For technical convenience, we assume $D_*(\perp) = \perp$, $D_*(\top) = \top$ and $|\perp| = |\top| = m$ hereafter. Let $p_0, \dots, p_k$ be (the access labels of) the nodes that prefixes of r reach, where $p_0 = \varepsilon$ is the root, $|p_{i-1}| < |p_i|$ for $i = 1, \dots, k$, and p_k is the last node where a prefix of r reaches in S. Let r_i for $i = 0, \dots, k$ be the suffix of r of length $m - |p_i|$. We know that $D_*(r) \neq S(r)$. Our learner asks an MQ on $p_k r_k$. Then either $D_*(p_k r_k) = D_*(r)$ or $D_*(p_k r_k) = S(r)$ holds. If $p_k \in \{\perp, \top\}$, then r_k is empty and we assume $p_k r_k = p_k$. In this case, we do not need to ask a teacher to get the value $D_*(p_k)$. By definition it is $S(r)$ and thus $D_*(p_k r_k) = S(r) \neq D_*(r)$.

(Case 1) Suppose $D_*(r) = D_*(p_k r_k) \neq S(r)$. In this case, p_k is not terminal. Thus r does not reach a terminal in S. This means that $S(r) = \perp$ and $D_*(p_k r_k) = \top$ and that no prefix of r_k labels an outgoing edge of p_k. There are $b \in \{0,1\}$, $i \geq 0$ and $r'_k \in \{0,1\}^*$ such that $r_k = b0^i1r'_k$ and the b-edge of p_k is labeled with $b0^j$ for some $j > i$. We call EdgeSplit$(p_k, b0^i, 1r'_k)$.

(Case 2) Suppose $D_*(r) \neq D_*(p_kr_k) = S(r)$. One can find j by binary search such that

$$D_*(r) = D_*(p_0r_0) = D_*(p_jr_j) \neq D_*(p_{j+1}r_{j+1}) = D_*(p_kr_k) = S(r)$$

with the aid of MQs on p_ir_i for at most $\lceil \log k \rceil$ different numbers i. Let q be such that $(p_j, q, p_{j+1}) \in E$, i.e., $r_j = qr_{j+1}$. Now r_{j+1} witnesses that $p_jq \not\approx_{D_*} p_{j+1}$, so the edge labeled by q from p_j should not point to p_{j+1}. We call NodeSplit(p_j, q, r_{j+1}). We note that p_j cannot be the root node. (Otherwise, it must hold $p_jq = q = p_{j+1}$ by C2(1) and C3 and thus $p_jq \approx_{D_*} p_{j+1}$.) Moreover, p_{j+1} is not terminal. (Otherwise, we would not have $(p_j, q, p_{j+1}) \in E$ by C3.)

Algorithm 2: Update-Hypothesis

```
Input: a counterexample $r$
if $T_0(\varepsilon) = \mu$ and $r \in 1\{0,1\}^{m-1}$ then
    Add $(\varepsilon, 10^j, T(10^j))$ to $E$ for the smallest $j$ such that $T(10^j) \neq \mu$;
    Relabel the unique node of $T_0$ with $\varepsilon$;
else
    Let $(p_0, p_1, \ldots, p_k)$ be the sequence of nodes where prefixes of $r$ reach in $S$;
    if $D_*(r) = D_*(p_k \cdot \mathrm{suf}(r, m - |p_k|))$ then
        Let $q \in \{0,1\}0^*$ and $r'$ be such that $\mathrm{suf}(r, m - |p_k|) = q1r'$;
        call EdgeSplit$(p_k, q, 1r')$;
    else
        Find $i$ such that $D_*(p_i \cdot \mathrm{suf}(r, m - |p_i|)) \neq D_*(p_{i+1} \cdot \mathrm{suf}(r, m - |p_{i+1}|))$;
        call NodeSplit$(p_i, q, \mathrm{suf}(r, m - |p_{i+1}|))$ for $(p_i, q, p_{i+1}) \in E$;
    end
end
```

4.3. EdgeSplit

The function EdgeSplit takes three strings $p \in V$, $q \in \{0,1\}0^*$ and $r \in 1\{0,1\}^*$ such that

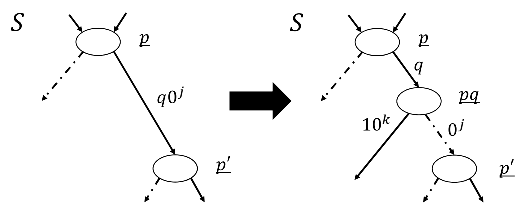
- $|pqr| = m$,
- $(p, q0^j, p') \in E$ for some $j > 0$ and $p' \in V$,
- $D_*(pqr) = \top$.

While pq does not reach a node in the current S, the fact $D_*(pqr) = \top$ witnesses $pq \in \mathrm{Reach}(D_*)$. EdgeSplit adds a new node pq to V and replaces $(p, q0^j, p')$ with (p, q, pq) in E (see Fig. 1(a)). We need to

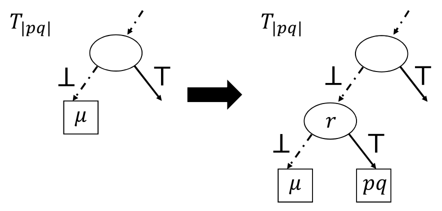
1. determine the nodes that the two edges of pq should point to,
2. redirect some edges towards the new node pq.

For the first point, we add a 0-edge $(pq, 0^j, p')$ and a 1-edge $(pq, 10^k, T(pq10^k))$ to E, for the smallest k such that $T(pq10^k) \neq \mu$.

Concerning the second point, S may have nodes whose edges should point to pq but did not before creating the node because of the absence of that node. We modify the



(a)



(b)

Figure 1: EdgeSplit(p, q, r) updates (a) ZDDAS S and (b) classification tree $T_{|pq|}$.

classification tree $T_{|pq|}$ by replacing the μ node with the tree of height 1, whose root, $\bot$-child and $\top$-child are labeled with r, μ and pq, respectively (see Fig. 1(b)).

Then for every $(s, t, u) \in E$ such that $|s| < |pq| < |st|$, we classify st' by $T_{|pq|}$ where t' is the prefix of t such that $|st'| = |pq|$. Since st' was classified into μ before updating T, it is enough to ask an MQ on $st'r$ to know the value of $T(st')$. If $D_*(st'r) = \top$, then we update E to be $(E \setminus \{(s, t, u)\}) \cup \{(s, t', pq)\}$.

```
Algorithm 3: EdgeSplit
Input: $p \in V$, $q \in \{0,1\}0^*$ and $r \in 1\{0,1\}^*$ such that $D_*(pqr) = \top$ and $(p, q0^j, p') \in E$ for
       some $j > 0$ and $p' \in V$
Add $pq$ to $V$;
Let $E := (E \setminus \{(p, q0^j, p')\}) \cup \{(p, q, pq), (pq, 0^j, p'), (pq, 10^k, T(pq10^k))\}$ for the smallest $k$
such that $T(pq10^k) \neq \mu$;
Replace the $\mu$-node of $T_{|pq|}$ with a tree of height 1 whose root, $\bot$-child and $\top$-child are
labeled with $r$, $\mu$ and $pq$, respectively;
for each $(s, t, u) \in E$ such that $|s| < |pq| < |st|$ do
    let $t' = \mathrm{pre}(t, |pq| - |s|)$;
    if $D_*(st'r) = \top$ then Let $E := (E \setminus \{(s, t, u)\}) \cup \{(s, t', pq)\}$;
end
```

4.4. NodeSplit

The function NodeSplit takes $p \in V \setminus \{\varepsilon\}$, $q \in \{0,1\}0^*$, $r \in \{0,1\}^+$ such that

- $|pqr| = m$,
- $(p, q, p') \in E$ for some non-terminal node p',
- $D_*(pqr) \neq D_*(p'r)$.

That is, currently S connects p and p' by q but the suffix r witnesses that $pq \not\approx_{D_*} p'$. NodeSplit solves this problem by splitting the node p' into p' and pq. The new node pq will be connected from p by an edge labeled by q, i.e., E is updated to $(E \setminus \{(p, q, p')\}) \cup \{(p, q, pq)\}$ (see Fig. 2(a)). To complete the modification, we need to

1. determine the nodes that the two outgoing edges of pq should point to,
2. reconnect incoming edges of p' to pq if necessary.

For the first issue, we should find the smallest j_0 and j_1 such that $T(pq00^{j_0}) \neq \mu$ and $T(pq10^{j_1}) \neq \mu$. We then add two edges $(pq, 00^{j_0}, T(pq00^{j_0}))$ and $(pq, 10^{j_1}, T(pq10^{j_1}))$ to E.

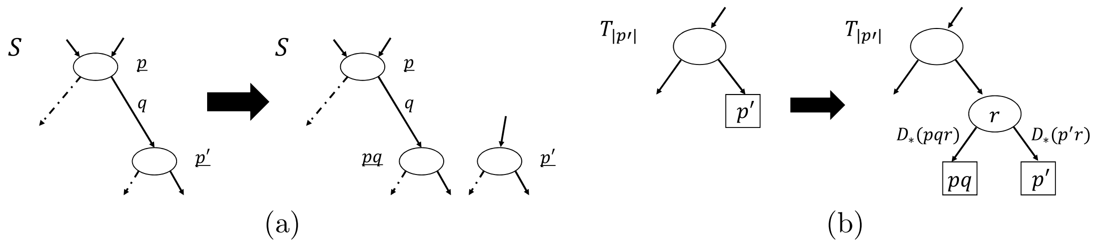


Figure 2: NodeSplit(p, q, r) updates (a) ZDDAS S and (b) classification tree $T_{|pq|}$.

To achieve the second, we modify the classification tree $T_{|p'|}$ and then reexamine every edge currently coming into p', which may be redirected towards pq. The leaf of $T_{|p'|}$ labeled by p' is now replaced by a subtree of height 1, whose root, $D_*(p'r)$-child and $D_*(pqr)$-child are labeled with r, p' and pq, respectively (see Fig. 2(b)). For each $(s, t, p') \in E$, since st was classified into p' before updating T, it is enough to ask an MQ on str to obtain $T_{|p'|}(st)$. If $D_*(str) \neq D_*(p'r)$, then we update E to be $(E \setminus \{(s, t, p')\}) \cup \{(s, t, pq)\}$.

Algorithm 4: NodeSplit

```
Input: $p \in V \setminus \{\varepsilon\}$, $q \in \{0,1\}0^*$ and $r \in \{0,1\}^{m-|pq|}$ such that $(p, q, p') \in E$ and
       $D_*(pqr) \neq D_*(p'r)$ for some non-terminal node $p'$
Add $pq$ to $V$;
Let $E := (E \setminus \{(p, q, p')\}) \cup \{(p, q, pq), (pq, 00^{j_0}, T(pq00^{j_0})), (pq, 10^{j_1}, T(pq10^{j_1}))\}$ for the
smallest $j_b$ such that $T(pqb0^{j_b}) \neq \mu$ for $b = 0, 1$;
Replace the node labeled with $p'$ of $T_{|p'|}$ with a tree of height 1 whose root, $D_*(p'r)$-child
and $D_*(pqr)$-child are labeled with $r$, $p'$ and $pq$, respectively;
for each $(s, t, p') \in E$ do
    if $D_*(str) = D_*(pqr)$ then Let $E := (E \setminus \{(s, t, p')\}) \cup \{(s, t, pq)\}$;
end
```

5. Example Run

Suppose that our learning target is the ZDD D_* in Fig. 3, where the universe is $\{x_1, x_2, x_3\}$. The learner asks the first MQ on 000. Since the answer is $D_*(000) = \perp$, the initial ZDDAS has a dummy node connected with the $\perp$-terminal (Fig. 3(a)). Against the EQ on $\tilde{S}_1$, the teacher may return a counterexample 110 for which $D_*(110) = \top \neq S_1(110) = \perp$. It reveals that $\varepsilon \in \text{Reach}(D_*)$ and thus ε is promoted to a non-dummy node. We need to give a 1-edge to ε. Since $T_1(1) = T_2(10) = \mu$ and $T_3(100) = \perp$, we decide to connect ε with $\perp$-terminal with the 1-edge labeled with 100 (Fig. 3(b)). Note that at this time the counterexample 110 is still misclassified by the learner's hypothesis. Hence the teacher may return the same counterexample again to the EQ on $\tilde{S}_2$. Then EdgeSplit$(\varepsilon, 1, 10)$ is called. We create a node 1 and edges $(1, 00, \perp), (1, 10, \top)$, since $T_3(100) = \perp$ and $T_3(110) = \top$. Here the suffix 10 witnesses $1 \in \text{Reach}(D_*)$. Accordingly we modify T_1 so that it classifies strings of length 1 using the witness 10 (Fig. 3(c)). When 111 is returned to the EQ on $\tilde{S}_3$, we call EdgeSplit$(1, 1, 1)$. We create a node 11 and edges $(11, 0, \top), (11, 1, \top)$, since

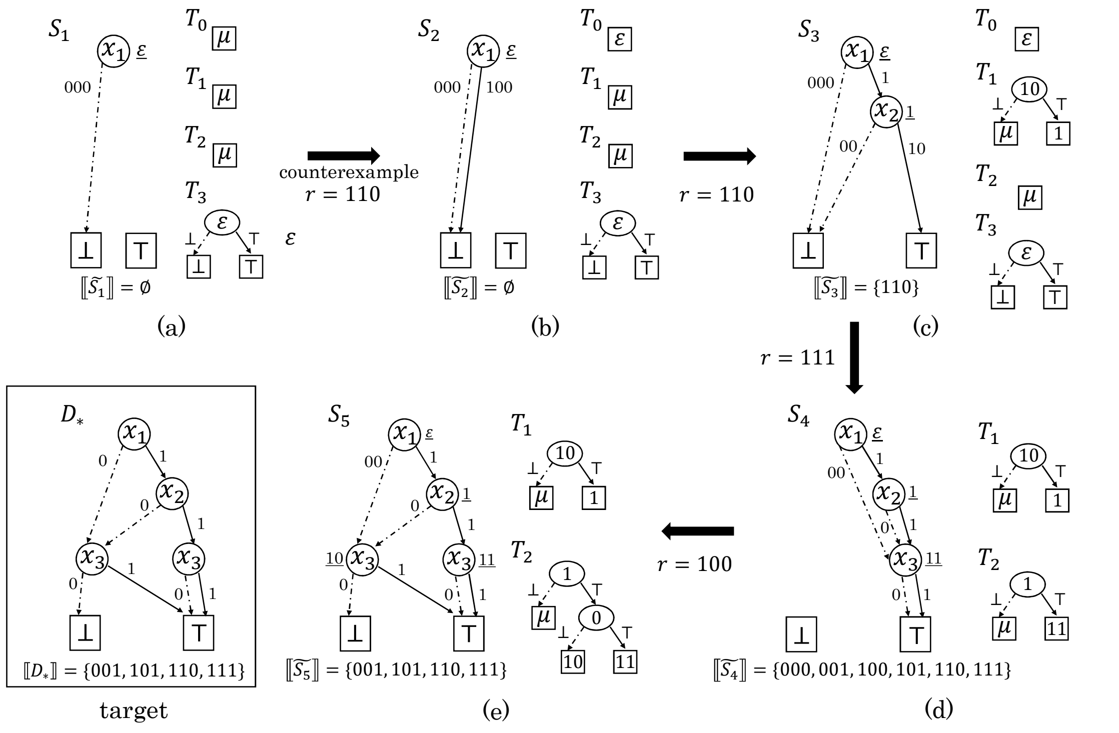

S_1
T_0
T_1
T_2
T_3
$[\![\tilde{S}_1]\!] = \emptyset$
(a)
counterexample
$r = 110$
S_2
$[\![\tilde{S}_2]\!] = \emptyset$
(b)
$r = 110$
S_3
$[\![\tilde{S}_3]\!] = \{110\}$
(c)
$r = 111$
S_4
$[\![\tilde{S}_4]\!] = \{000, 001, 100, 101, 110, 111\}$
(d)
$r = 100$
S_5
$[\![\tilde{S}_5]\!] = \{001, 101, 110, 111\}$
(e)
D_*
$[\![D_*]\!] = \{001, 101, 110, 111\}$
target

Figure 3: A running example. Access strings are underlined. Classification trees T_0 and T_3 are omitted in (d) and (e), because they do not change from the previous steps.

$T_3(110) = T_3(111) = \top$. Here the suffix 1 witnesses $11 \in \mathrm{Reach}(D_*)$. Accordingly we modify T_2 using 11. We reexamine edges that "skip" nodes labeled x_3 so far; namely, the 0-edges of the nodes ε and 1. According to the classification results $T_2(00) = T_2(10) = 11$, the learner reconnects those 0-edges towards the new node 11. We have obtained S_4 (Fig. 3(d)). The teacher may return 100 to the EQ on $\tilde{S}_4$. The suffix 0 witnesses $10 \not\approx_{D_*} 11$ by $D_*(100) = \bot \neq D_*(110) = \top$. We call NodeSplit$(1, 0, 0)$ and create a node 10 with edges $(10, 0, \bot)$ and $(10, 1, \top)$. The edges pointing to 11 should be reexamined using the key suffix 0. Those are $(\varepsilon, 00, 11), (1, 0, 11), (1, 1, 11)$. Since $D_*(000) = D_*(100) = \bot$ and $D_*(110) = \top$, the first two will be redirected to 10 (Fig. 3(e)). Then the teacher will answer Yes towards the EQ on $\tilde{S}_5$.

6. Correctness and Efficiency

This section gives a formal proof for the correctness and the efficiency of our algorithm. By Lemma 2, it is enough to show the following lemma for the correctness.

Lemma 3 *S and T always satisfy the conditions shown in Section 3.3.*

Proof For the initial ZDDAS S and classification forest T, all the conditions are satisfied, since $\hat{V} = \emptyset$ and each T_i is just a μ-leaf for all $i < m$. Suppose that we update S^{old} and T^{old} to S^{new} and T^{new}, respectively, where S^{old} and T^{old} satisfy the conditions in concern.

Suppose that a dummy node ε is promoted by getting a counterexample $r \in 1\{0,1\}^{m-1}$.

C1. (1) For $\varepsilon \in V^{\text{new}}$, we have $\varepsilon \in \text{Reach}(D_*)$, which is witnessed by $D_*(r) = \top$.
(2) The only q such that $\varepsilon \approx_{D_*} q$ is ε itself.

C2. (1) The label of the unique node of T_0 is changed to ε, so we have $T_0^{\text{new}}(\varepsilon) = \varepsilon$.
(2) We know that $\varepsilon \in \text{Reach}(D_*)$.

C3. The 1-edge of ε is determined so that it satisfies C3.

Suppose that EdgeSplit(p, q, r) is called, where $p \in V^{\text{old}}$, $q \in \{0,1\}0^*$, $r \in 1\{0,1\}^*$, $D_*(pqr) = \top$, $(p, q0^j, p') \in E^{\text{old}}$ for some $j > 0$ and $p' \in V^{\text{old}}$.

C1. (1) For $pq \in V^{\text{new}}$, we have $pq \in \text{Reach}(D_*)$, which is witnessed by $D_*(pqr) = \top$.
(2) Let $s \in \hat{V}^{\text{old}}_{|pq|}$. Then by $T^{\text{old}}_{|pq|}(s) = s$ (C2(1)) and $T^{\text{old}}_{|pq|}(pq) = \mu$ (C3), their lowest common ancestor in $T^{\text{old}}_{|pq|}$ is an internal node, which is the case for $T^{\text{new}}_{|pq|}$. Let r' be the label of that node. Then we have $D_*(sr') \neq D_*(pqr')$, which shows $pq \not\approx_{D_*} s$.

C2. (1) Since $T^{\text{old}}_{|pq|}(pq) = \mu$, the modification on $T_{|pq|}$ causes $T^{\text{new}}_{|pq|}(pq) = pq$.
(2) For $s \in \{0,1\}^{<m} \setminus \text{Reach}(D_*)$, we have $D_*(sr) = \bot$ and $T^{\text{old}}_{|pq|}(s) = \mu$. Hence $T^{\text{new}}_{|pq|}(s) = \mu$.

C3. We create new edges for the new node pq as C3 instructs. We reexamine existing edges in accordance with C3.

Suppose that NodeSplit(p, q, r) is called, where $p \in V^{\text{old}} \setminus \{\varepsilon\}$, $q \in \{0,1\}0^*$, $r \in \{0,1\}^+$, $(p, q, p') \in E^{\text{old}}$ for some $p' \in V^{\text{old}}$ and $D_*(pqr) \neq D_*(p'r)$.

C1. (1) For $pq \in V^{\text{new}}$, we have $T^{\text{old}}(pq) = p'$ (C3), which means $pq \in \text{Reach}(D_*)$.
(2) Let $s \in \hat{V}^{\text{old}}_{|pq|}$. If $s = p'$, we have $pq \not\approx_{D_*} s$ by $D_*(pqr) \neq D_*(p'r)$. For $s \neq p'$, by $T^{\text{old}}_{|pq|}(s) = s$ (C2(1)) and $T^{\text{old}}_{|pq|}(pq) = p'$ (C3), their lowest common ancestor in $T^{\text{old}}_{|pq|}$ is an internal node and it is the same for $T^{\text{new}}_{|pq|}$. Let r' be the label of that node. Then we have $D_*(sr') \neq D_*(pqr')$, which shows $pq \not\approx_{D_*} s$.

C2. (1) Since $T^{\text{old}}_{|pq|}(pq) = p'$, the modification on $T_{|pq|}$ causes $T^{\text{new}}_{|pq|}(pq) = pq$ and $T^{\text{new}}_{|pq|}(p') = p'$. Classification of the other nodes $s \notin \{pq, p'\}$ are not affected.
(2) $T^{\text{old}}_{|pq|}(s) = \mu$ implies $T^{\text{new}}_{|pq|}(s) = \mu$ for any $s \in \{0,1\}^{|pq|}$.

C3. We create new edges for the new node pq as C3 instructs. We reexamine existing edges in accordance with C3. ■

Theorem 4 *For an arbitrary reduced ZDD D_*, our algorithm learns ZDDs with n EQs and at most $n(4n + \lfloor \log m \rfloor)$ MQs with a MAT, where m is the cardinality of the universe and n is the number of internal nodes of D_*.*

Proof Algorithm creates one non-dummy node for each EQ (including promotion of the dummy to a non-dummy). By Lemma 2, the number of EQs required is exactly n.

Let us count the number of MQs. Suppose that a counterexample is given. If it promotes a dummy node to non-dummy, we determine the end of the 1-edge of the root by making at most n MQs to get the smallest k such that $T(10^k) \neq \mu$. Otherwise, we make an MQ to decide which of the procedures EdgeSplit (Case 1) and NodeSplit (Case 2) should be called.

(Case 1) EdgeSplit involves MQs for two kinds of tasks. To determine the nodes where the 1-edge of the new node should point requires at most n MQs. To redirect an existing edge requires one MQ. There can be at most $2(n-1)$ edges to examine, so $2(n-1)$ MQs in total. Hence, an execution of EdgeSplit makes at most $(3n-2)$ MQs.

(Case 2) To find the right argument of NodeSplit by binary search, we require $\lceil \log m \rceil$ MQs. NodeSplit has two kinds of tasks that require MQs. To determine the nodes where the two edges of the new node should point requires at most $2n$ MQs. To reconnect an existing edge requires one MQ. There can be at most $2(n-1)$ edges to examine, so $2(n-1)$ MQs in total. Hence, an execution of NodeSplit makes at most $(4n + \lceil \log m \rceil - 2)$ MQs.

All in all, the learner asks at most $n(4n + \lfloor \log m \rfloor)$ MQs. ■

7. Concluding Remarks

We have proposed an efficient MAT learning algorithm for ZDDs based on the algorithm for BDDs by Nakamura (2005). Nakamura's algorithm requires n EQs and at most $2n(\lceil \log m \rceil + 3n)$ MQs. The difference of the numbers of required MQs comes from the difference of the reduction rules of ZDDs and BDDs. While a ZDD suppresses a node v whose 1-child is the $\perp$-terminal from a decision diagram, a BDD suppresses a node v whose 0-child and 1-child are the same. Accordingly the condition on a string $p \in \{0,1\}^{<m}$ to be in $\mathrm{Reach}(C_*)$ for a BDD C_* is to admit $q \in \{0,1\}^{m-|p|-1}$ such that $C_*(p0q) \neq C_*(p1q)$. Thus, we need two MQs on $p0q$ and $p1q$ to be sure that $p \in \mathrm{Reach}(C_*)$. Recall that for a ZDD D_*, one fact $D_*(p1r) = \top$ is enough to see $p \in \mathrm{Reach}(D_*)$. This makes the required numbers of MQs to classify strings using the classification forest different.

References

D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

M. Kearns and U. Vazirani. *An introduction to computational learning theory.* The MIT Press, 1994.

D. E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams.* Addison-Wesley Professional, 12th edition, 2009.

S. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proceedings of the 30th Design Automation Conference*, pages 272–277. ACM, 1993.

A. Nakamura. An efficient query learning algorithm for ordered binary decision diagrams. *Information and Computation*, 201:178–198, 2005.

R. L. Rivest and R. E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, 1993.