arXiv:2303.03195v1 [cs.LG] 3 Mar 2023

有序多终端二叉决策图的查询学习算法

Atsuyoshi Nakamura

[a]北海道大学信息科学技术研究生院，日本北海道札幌市北区北*14*条西*9*丁目，邮编 *060-0814*

摘要

我们通过扩展有序二元决策图（OBDD）的算法，提出了一种用于有序多终端二元决策图（OMTBDD）的查询学习算法，该算法最多使用 n 次等价查询和 $2n(\lceil \log_2 m \rceil + 3n)$ 次成员查询。我们通过使用合成生成的目标 OMTBDD 进行实验，检验了所获上界的紧确性。我们还通过使用 UCI机器学习资料库中的数据集进行其他实验，指出了将该算法应用于分类问题的可能性。

关键词： 查询学习，样本复杂度，OMTBDD

1. 引言

二元决策图（BDD）是一种布尔函数的表示方式，已知它对许多函数而言都很紧凑，且易于操作 [1]。它是一种有向无环图（DAG），具有一个根节点和两个分别标记为 0 和 1 的汇点。每个非汇点都标记一个布尔变量 x_i，而对变量的赋值 $x_i = a_i$ 会通过在标记为x_i的节点处选择标记为 a_i的出边，确定一条从根节点到某个汇点的路径。如果从根节点到某个汇点的任一路径上，沿途经过的变量标签序列与某个变量顺序一致，则称其为有序 BDD（OBDD）。OBDD 因其良好性质而广受欢迎：对于任意固定的变量顺序，任何布尔函数都可以表示为唯一的约简有序二叉决策图（reduced OBDD）。

电子邮箱地址：atsu@ist.hokudai.ac.jp (Atsuyoshi Nakamura)

提交至爱思唯尔的预印本 *2023*年*3*月*7*日

多终端二元决策图（MTBDD） [2]是 BDD 的扩展，用于表示布尔变量的多值函数。其结构上的区别仅在于汇点的数量；MTBDD 可以有两个以上的汇点。有序 MTBDD（OMTBDD）是指非汇点的变量标签受到与 OBDD 相同限制的 MTBDD。已知 OMTBDD 继承了 OBDD 的良好性质，例如具有唯一约简形式。

本文将 OBDD 的查询学习算法扩展为 OMTBDD 的查询学习算法。本文采用的查询学习是利用等价查询和成员查询进行的学习 [3]。在函数类 $\mathcal{F}$ 的查询学习研究中，我们开发了一种高效算法，能够利用允许提出的查询识别 $\mathcal{F}$ 中的未知目标函数 f。针对学习者所选择的假设 $h \in \mathcal{F}$，等价查询 EQ(h) 是询问 $h = f$ 是否成立的查询；如果 $h = f$，则对该查询的回答为“是”（’YES’），否则为“否”（’NO’）。当回答为“否”时，学习者还会获得反例 e，使得 $h(e) \neq f(e)$。成员查询是针对学习者所选择的赋值 a，询问函数值 $f(a)$ 的查询，并返回值 $f(a)$。回答成员查询的预言机可以通过黑盒函数实现，但回答等价查询的预言机则难以轻易实现。对于随机抽样的 x，随机测试 $h(x) = f(x)$ 已知足以实现概率近似正确（PAC）学习，而非识别 [3]。

我们提出的用于 OMTBDD 的查询学习算法称为 QLearn-OMTBDD，它是 QLearn-π-OBDD [4] 的扩展，通过等价查询和成员查询来识别目标 OBDD。在这些算法中，我们使用（节点）分类树 T_i 对部分赋值 $x_1 = a_1$，$x_2 = a_2$，……，$x_{i-1} = a_{i-1}$ 进行分类，将其归入假设 OMTBDD 中带有标签 x_i 的内部节点，或归入 μ，以判断该部分赋值到达了哪个节点，或者是否没有可到达的节点（在其中含有 μ 的情况下）。分类树根据成员查询的回答作出判断。由于在具有 K 个汇点的 OMTBDD 中，成员查询可能的回答数量从两个增加到 K，因此必须修改树的结构，这也促使我们对其更新过程作出相应修改。特别是，在分类树的某个节点处，可能不存在与成员查询回答相对应的已有边标签，而这种情况在 OBDD 中从未发生。在这种情况下，必须更新树，使其包含这条边。

我们证明，使用 QLearn-OMTBDD算法，任意目标约简 OMTBDD 都可以通过至多 $2n(\lceil \log_2 m \rceil + 3n)$ 次成员查询和至多 n 次等价查询来学习，这与

文献 [4] 中给出的 OBDD 上界完全相同。我们利用合成生成的 OMTBDD，在实验中检验了这些查询复杂度上界的紧确性。我们还检验了该算法在分类问题中的适用性。我们的多终端扩展使 OMTBDD 能够表示多类分类器。实值特征也可以转换为二值特征，其取值表示高于或低于某个固定阈值。用于分类器的 OMTBDD表示具有实用价值，因为可以对表示某些条件的 OMTBDD 执行各种操作。通过在 UCI 机器学习知识库中的 12 个实值特征数据集上进行实验，我们展示了一种构造 OMTBDD 的方法：利用查询学习，从基于树的分类器及其训练数据中学习，而这些训练数据均被该分类器正确预测；其中，分类器用于回答成员查询，并以等价查询识别取代与训练数据的一致性检验。对于基于树的分类器，我们使用决策树和随机森林。在决策树分类器上，除一个数据集外，未观察到明显的准确率下降；在 12 个数据集中，有 5 个的节点数至多保持不变。对于两个数据集，即使在原始决策树中共享具有相同标签的叶节点，学习得到的 OMTBDD 节点数仍少于原始决策树。对于随机森林分类器，除两个数据集外均观察到准确率下降，但在 10 个数据集中有 5 个实现了节点数的大幅减少[1]。在这 5 个成功缩减的数据集中，有两个数据集上，从随机森林学习得到的 OMTBDD 准确率高于决策树。我们在这些基准数据集上的分类问题实验结果表明，该查询学习算法具有应用潜力。

相关工作。 查询学习由 Angluin [3], 提出，用于学习确定性有限自动机（DFAs）的算法是最著名的查询学习算法之一，它使用等价查询和成员查询。在针对 DFAs 的查询学习中，Kearns 和 Vazirani [5] 使用分类树取代了 Angluin 算法中的观察表。Gavald`a 和 Guijarro [6]将 Angluin 针对 DFAs 的查询学习算法扩展为用于 OBDDs 的算法。Nakamura [4] 通过使用分类

[1]对于癫痫发作数据集和魔术数据集，由于节点数量过多，无法从随机森林中学习 OMTBDD。

树，将学习一个 OBDD 所需的成员查询数量减少了 () 倍，其中 m 是变量数量。Mizumoto等人 [7]开发了 Nakamura算法的 ZDD 版本。

2. 预备知识

多终端二元决策图（MTBDD）是二元决策图（BDD）的扩展，可表示取值域为 $\{0,1\}^m$ 且值数多于 2 的函数。令 $\{0,\ldots,K-1\}$ for $K \geq 2$ 表示 K 值函数的取值集合。表示 K 值函数的 MTBDD 是一个有向无环图，包含一个根节点和至多 K 个汇点，节点的标签为 0，...，$K-1$。最简单的 MTBDD 由一个汇点构成，该汇点同时也是根节点，表示一个常数函数。其他 MTBDD 至少有两个汇点和一个内部（非汇点）节点。每个内部节点都标记一个布尔变量，并有两条出边，分别是带 0 标记的边和带 1 标记的边。有序 MTBDD（OMTBDD）指的是一种 MTBDD，其中从根节点到任一汇点的路径上，各节点所标记变量的序列必须符合某个预设顺序。本文将变量顺序固定为 x_1，x_2，...，x_m。

给定一个赋值 $x_1 = a_1, \ldots, x_m = a_m$，OMTBDD 在该赋值下表示的函数值按如下方式计算：从其根节点开始，在标签为 x_i 的节点处选择标记为 a_i 的出边，并取最终到达的汇点的标签 $j \in \{0,\ldots,K-1\}$ 作为函数值。赋值 $x_1 = a_1, \ldots, x_m = a_m$ 也可用二进制字符串 $a_1 a_2 \ldots a_m$ 表示。

我们用粗体字母（如 $\boldsymbol{a}$、$\boldsymbol{b}$）表示字符串，任意字符串 $\boldsymbol{a}$ 的长度记作 $|\boldsymbol{a}|$。$\boldsymbol{a}$ 与 $\boldsymbol{b}$ 的拼接串记作 $\boldsymbol{a} \cdot \boldsymbol{b}$，有时简写为 $\boldsymbol{ab}$。对于字符串 $\boldsymbol{a}$，pre($\boldsymbol{a}, i$) 和 suf($\boldsymbol{a}, i$) 分别表示 $\boldsymbol{a}$ 的长度为 i 的前缀和后缀字符串。对于两个长度为 i 的字符串 $\boldsymbol{a}$、$\boldsymbol{b}$ 以及一个自然数 $j < i$，cro($\boldsymbol{a}, \boldsymbol{b}, j$) 表示通过拼接 pre($\boldsymbol{a}, i-j$) 和 suf($\boldsymbol{b}, j$) 构造的长度为 i 的字符串。我们稍微滥用一下记号，OMTBDD D 表示的函数也记作 D，因此 $D(a_1, a_2, \ldots, a_m)$ 是 D 在赋值 $x_1 = a_1, \ldots, x_m = a_m$ 下的函数值，也记作 $D(\boldsymbol{a})$，其中 $\boldsymbol{a} = a_1 a_2 \cdots a_m$。

对于 OMTBDD D 中的节点 N，节点 N 的访问字符串是一个字符串 $a_1 a_2 \cdots a_{i-1}$，使得对于赋值 $x_1 = a_1$，$x_2 = a_2$，....., $x_{i-1} = a_{i-1}$，D 上的路径恰好到达节点 N。对于带标签为 x_i 的节点，访问字符串的长度为 $i-1$；对于汇点，长度为 m。我们用 nodes$_i$(D) 表示 D 中节点的长度为 $i-1$ 的访问字符串集合，并令 nodes(D) $= \bigcup_{i=1}^{m+1}$ nodes$_i$(D)。

然后，对于 $\boldsymbol{a}$， $\boldsymbol{b} \in$ nodes(D)，定义等价关系“$\overset{D}{=}$”如下：

$$\boldsymbol{a} \overset{D}{=} \boldsymbol{b} \overset{def}{\Leftrightarrow} \boldsymbol{a} \text{ 和 } \boldsymbol{b} \text{ 是 } D\text{.中同一节点的访问字符串}$$

令 $[\boldsymbol{a}]$ 表示 $\boldsymbol{a}$的等价类。若节点子集 V （D）中每个节点在 D中恰有一个访问字符串，则称 V 为 D 的节点标识符集合。对于 D的节点标识符集合 V，令 V_i 表示 V 中长度为（$i-1$）的访问字符串集合。于是 $V=\bigcup_{i=1}^{m+1} V_i$，节点$_i$（$D$）可划分为$\{[\boldsymbol{v}] \mid \boldsymbol{v} \in V_i\}$，而节点（$D$）也可划分为 $\{[\boldsymbol{v}] \mid \boldsymbol{v} \in V\}$。我们用 $\boldsymbol{v} \in V$作为节点标识符，并令“节点 $\boldsymbol{v}$”表示访问字符串为 $\boldsymbol{v}$的节点。

我们考虑的学习框架是 Angluin [3]提出的查询学习。在该框架中，使用等价查询和成员查询来识别未知目标函数 f。等价查询会询问 f 是否等价于假设 h；若是，则返回“是”；若否，则返回“否”以及一个反例 e（f（e）$\neq h$（e））。我们将等价查询视为一个函数 EQ，它将假设 h 映射为由答案和反例组成的一对（Ans e，其中 Ans 为 $\in$ {“是”，“否”），并令 EQ(h)表示（Ans e）。成员查询询问目标函数f 在赋值 a上的取值 f（a）。

3. 节点分类树

对于每个 $i=1$、2、.....、$m+1$，令 $\mathcal{S}_i$ 为长度为 $i-1$ 的 $\{0,1\}$-字符串集合。令 D 为一个 OMTBDD，V 为 D 的节点标识符集合。那么，$\mathcal{S}_i$ 可以划分为 $\mathcal{S}_i=\bigcup_{\boldsymbol{v}\in V_i}[\boldsymbol{v}] \cup (\mathcal{S}_i \setminus \bigcup_{\boldsymbol{v}\in V_i}[\boldsymbol{v}])$，其中 $\mathcal{S}_i \setminus \bigcup_{\boldsymbol{v}\in V_i}[\boldsymbol{v}]$ 是长度为 ($i-1$) 的字符串集合，这些字符串不会到达 D 中的任何节点。

如果给定一个 OMTBDD D 及其节点标识符集合 V，我们可以轻松判断给定的 $\boldsymbol{a}=a_1a_2\cdots a_{i-1} \in \mathcal{S}_i$ 是否会到达节点 $\boldsymbol{v} \in V_i$，或者是否不会从赋值 $x_1=a_1$、$x_2=a_2$、.....、$x_{i-1}=a_{i-1}$ 对应的 D 中的路径到达 $\boldsymbol{v} \in V_i$ 中的任何节点。那么，如果未给定 D，但我们可以针对 D 提出成员查询，是否也能回答同样的问题? 如果 D 是约简的，答案是肯定的。原因如下。

考虑 OMTBDD $D_{\boldsymbol{v}}$，它是从节点$\boldsymbol{v} \in V_i$可达的子图。OMTBDD $D_{\boldsymbol{v}}$可视为定义在($x_i, x_{i+1}, \ldots, x_m$)上的一个 K值函数 $\in\{0,1\}^{m-i+1}$，也就是说，对于 $\boldsymbol{v}=v_1v_2\cdots v_{i-1}$，$D_{\boldsymbol{v}}(x_i, x_{i+1}, \ldots, x_m) = D(v_1, v_2, \ldots, v_{i-1}, x_i, x_{i+1}, \ldots, x_m)$。令节点 $\boldsymbol{v}_a \in V$为从节点 $\boldsymbol{v}$ 出发的 a标签边所到达的节点。那么，$D_{\boldsymbol{v}}(0, x_{i+1}, \ldots, x_m)$和 $D_{\boldsymbol{v}}(1, x_{i+1}, \ldots, x_m)$分别是由 $D_{\boldsymbol{v}_0}$ 和 $D_{\boldsymbol{v}_1}$表示的函数。如果 D 是约简的，

$D_{\boldsymbol{v}}(0, x_{i+1}, \ldots, x_m)$和 $D_{\boldsymbol{v}}(1, x_{i+1}, \ldots, x_m)$必须不同，否则就可以进一步约简；节点 $\boldsymbol{v}$ 可以移除，其入边可以直接连到节点 $\boldsymbol{v}_0$ （或 $\boldsymbol{v}_1$）。因此，必定存在一个字符串 $a_{i+1}a_{i+2}\cdots a_m$，使得 $D_{\boldsymbol{v}}(0, a_{i+1}, \ldots, a_m) \neq D_{\boldsymbol{v}}(1, a_{i+1}, \ldots, a_m)$。令 $\boldsymbol{r}^{(v)}$ 为这样的字符串 $0a_{i+1}a_{i+2}\cdots a_m$ 和 $1a_{i+1}a_{i+2}\cdots a_m$中的一个，并令$\dot{\boldsymbol{r}}^{(v)}$ 表示将 $\boldsymbol{r}^{(v)}$的首位比特翻转后得到的字符串，也就是它们中的另一个。如果 $\boldsymbol{a} \in \mathcal{S}_i \setminus \bigcup_{\boldsymbol{v}\in V_i}[\boldsymbol{v}]$，那么对于所有 $\boldsymbol{v} \in V_i$，$D(\boldsymbol{a}\boldsymbol{r}^{(v)}) = D(\boldsymbol{a}\dot{\boldsymbol{r}}^{(v)})$均成立，因此对于所有$\boldsymbol{v} \in V_i$，$D(\boldsymbol{a}\boldsymbol{r}^{(v)}) \neq D_{\boldsymbol{v}}(\boldsymbol{r}^{(v)})$或 $D(\boldsymbol{a}\dot{\boldsymbol{r}}^{(v)}) \neq D_{\boldsymbol{v}}(\dot{\boldsymbol{r}}^{(v)})$均成立。根据上述事实，

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

成立。这意味着，如果我们知道对于所有 $\boldsymbol{v} \in V_i$，$\boldsymbol{r}^{(v)}, D(\boldsymbol{v}\boldsymbol{r}^{(v)})$ 和 $D(\boldsymbol{v}\dot{\boldsymbol{r}}^{(v)})$，就可以通过对所有 $\boldsymbol{v} \in V_i$ 查询 $\boldsymbol{a}\boldsymbol{r}^{(v)}$ 和 $\boldsymbol{a}\dot{\boldsymbol{r}}^{(v)}$ 的成员关系，来检查 $\boldsymbol{a} \in \mathcal{S}_i \setminus \bigcup_{\boldsymbol{v}\in V_i}[\boldsymbol{v}]$ 是否成立。注意，对于满足 $\boldsymbol{v}' \overset{D}{\neq} \boldsymbol{v}$ 的 $\boldsymbol{a} \in [\boldsymbol{v}']$，$D(\boldsymbol{a}\boldsymbol{r}^{(v)}) = D(\boldsymbol{v}\boldsymbol{r}^{(v)})$ 和 $D(\boldsymbol{a}\dot{\boldsymbol{r}}^{(v)}) = D(\boldsymbol{v}\dot{\boldsymbol{r}}^{(v)})$ 可能成立，并且对于 $\boldsymbol{v}' \neq \boldsymbol{v}$，$\boldsymbol{r}^{(v')_D}$ 可以等于 $\boldsymbol{r}^{(v)}$。即使对于满足 $\boldsymbol{v}' \overset{D}{\neq} \boldsymbol{v}$ 的 $\boldsymbol{a} \in [\boldsymbol{v}']$，$D(\boldsymbol{a}\boldsymbol{r}^{(v)}) = D(\boldsymbol{v}\boldsymbol{r}^{(v)})$ 和 $D(\boldsymbol{a}\dot{\boldsymbol{r}}^{(v)}) = D(\boldsymbol{v}\dot{\boldsymbol{r}}^{(v)})$ 成立，只要 D 是约简的，我们就可以通过成员查询确定 $\boldsymbol{a} \in [\boldsymbol{v}]$ 是否成立。如果 D 是约简的，那么对于任意两个节点 $\boldsymbol{v}$、$\boldsymbol{v}' \in V_i$，都存在一个字符串 $a_i a_{i+1}\cdots a_m$，使得 $D_{\boldsymbol{v}}(a_i, a_{i+1}, \ldots, a_m) \neq D_{\boldsymbol{v}'}(a_i, a_{i+1}, \ldots, a_m)$，因为否则 $D_{\boldsymbol{v}} = D_{\boldsymbol{v}'}$ 就成立，从而还可以进一步约简；节点 $\boldsymbol{v}'$ 可以被移除，其所有入边都可以接到节点 $\boldsymbol{v}$ 上。令 $r^{(\boldsymbol{v},\boldsymbol{v}')}$ 表示满足 $D_{\boldsymbol{v}}(a_i, a_{i+1}, \ldots, a_m) \neq D_{\boldsymbol{v}'}(a_i, a_{i+1}, \ldots, a_m)$ 的字符串 $a_i a_{i+1}\cdots a_m$。然后，对于所有满足 $D(\boldsymbol{v}'\boldsymbol{r}^{(v)}) = D(\boldsymbol{v}\boldsymbol{r}^{(v)})$ 和 $D(\boldsymbol{v}'\dot{\boldsymbol{r}}^{(v)}) = D(\boldsymbol{v}\dot{\boldsymbol{r}}^{(v)})$ 的 $\boldsymbol{v}' \overset{D}{\neq} \boldsymbol{v}$，我们可以通过在 $\boldsymbol{a}\boldsymbol{r}^{(\boldsymbol{v},\boldsymbol{v}')}$ 处查询成员关系，来检查 $\boldsymbol{a} \in [\boldsymbol{v}]$ 是否成立。

根据上述讨论，我们可以构造节点分类树 $T_i (i = 1, \ldots, m)$，将 $\boldsymbol{a} \in \mathcal{S}_i$ 分类为 $\boldsymbol{v} \in V_i$ 或 μ，其中 μ 表示$\boldsymbol{a} \in \mathcal{S}_i \setminus \bigcup_{\boldsymbol{v}\in V_i}[\boldsymbol{v}]$。图1最左侧的图展示了节点分类树 T_i的形式。它是一棵有根树，由两种类型的内部节点和叶节点组成。一种内部节点是双测试节点，另一种是单测试节点。双测试节点带有标签 $\boldsymbol{r}^{(v)}$，其中$\boldsymbol{v} \in V_i$，并针对 $\boldsymbol{a}\boldsymbol{r}^{(v)}$ 和 $\boldsymbol{a}\dot{\boldsymbol{r}}^{(v)}$ 提出两个成员查询，以对字符串 $\boldsymbol{a} \in \mathcal{S}_i$进行分类。每个双测试节点有两条出边，一条标记为 $(D(\boldsymbol{v}\boldsymbol{r}^{(v)})\ D(\boldsymbol{v}\dot{\boldsymbol{r}}^{(v)}))$，并指向单测试节点或叶节点；另一条

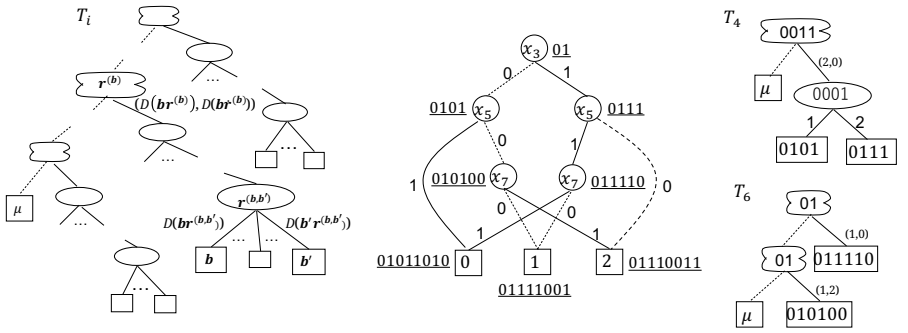


图1：节点分类树（左）及其针对一个示例 OMTBDD 的实例（右），中间为节点的访问字符串。

不带标签，并指向双测试节点或标记为 μ的叶节点。单测试节点带有标签 $\boldsymbol{r}^{(v,v')}$，其中 $\boldsymbol{v}, \boldsymbol{v}' \in V_i$，并针对$\boldsymbol{ar}^{(v,v')}$ 提出一个成员查询，以对字符串 $\boldsymbol{a} \in \mathcal{S}_i$进行分类。每个单测试节点最多有$K$ 条出边，标记为 $\ell \in \{0, 1, \ldots, K-1\}$，并指向单测试节点或标记为某个 $\boldsymbol{v}'' \in V_i$的叶节点。如果该边指向标记为 $\boldsymbol{v}''$的叶节点，则边的标签必须为 $D(\boldsymbol{v}''\boldsymbol{r}^{(v,v')})$。树 T_i 最多有 $|V_i|+1$ 个叶节点，每个叶节点都标记为 $\boldsymbol{v} \in V_i$ 或 μ。不同的叶节点必须具有不同的标签。如果存在一个标记为 μ的叶节点，则从根节点到该标记为 μ的叶节点的路径上的所有节点都必须是双测试节点，并且所有双测试节点都必须出现在该路径上，这保证了满足 (1) 右侧的条件。

对 $\boldsymbol{a} \in \mathcal{S}_i$ 进行 $V_i \cup \{\mu\}$ 分类，可以按如下方式使用节点分类树 T_i。从根节点开始。在标记为 $\boldsymbol{r}^{(v)}$ 的双测试节点处，对 $\boldsymbol{ar}^{(v)}$ 和 $\boldsymbol{a\dot{r}}^{(v)}$进行两次成员查询；如果存在标记为 $(D(\boldsymbol{ar}^{(v)})\ D(\boldsymbol{a\dot{r}}^{(v)}))$ 的边，则选择该边，否则选择无标记边。在标记为 $\boldsymbol{r}^{(v,v')}$ 的单测试节点处，对 $\boldsymbol{ar}^{(v,v')}$进行一次成员查询，并选择标记为 $D(\boldsymbol{ar}^{(v,v')})$ 的边。由 T_i分类的 $\boldsymbol{a}$ 的标签，记作 $T_i(\boldsymbol{a})$，就是最终通过重复上述操作所到达的叶节点的标签。

示例1。考虑图*1*中央所示的 *OMTBDD* D。每个节点旁带下划线的字符串是其在某个节点标识符集合 V 中的访问字符串，V 属于 D。该 *OMTBDD* 的节点分类树 T_4 和 T_6 显示在图的右侧。T_4 由一个双测试节点、一个单测试节点和三个叶节点组成，其中包括带有 μ标签的叶节点。T_6 由两个

双测试节点和三个叶节点组成。字符串 $1100 \in \mathcal{S}_4$ 在 D中到达节点 ***0101***。由于 D***(11000011)*** $= 2$、 D***(11001011)*** $= 0$ 以及 D***(11000001)*** $= 1$，它被 T_4分类为 ***0101***，这与赋值 ***(***x_1, x_2, x_3, x_4***) =(1,1,0,0)*** 在 D 中到达的节点一致。字符串 $101001 \in \mathcal{S}_6$在 D中未到达任何节点。由于 D***(10100101)*** $= D$ ***(10100111)*** $= 2$，它被 T_6分类为 μ，这也与赋值 ***(***$x_1, x_2, x_3, x_4, x_5, x_6$***) =(1, 0,1,0,0,1)*** 所到达节点的不存在相一致。

注***1***。***OMTBDD*** 的节点分类树与 ***OBDD*** 的分类树在边标签方面有所不同。对于 ***OBDD*** 的分类树，从带标签为 $\boldsymbol{r}$ 的双测试节点引出的边，其标签为 ***0*** 或 ***1***；当且仅当 $D(\boldsymbol{ar}) = 1$ 且 $D(\boldsymbol{a\dot{r}}) = 0$ 时，才会为测试字符串 $\boldsymbol{a} \in \{0,1\}^{i-1}$ 选择标签为 ***1*** 的边。对于 ***OMTBDD***，两种不同函数值的组合数量可能不止一种，因此我们采用标签 ***(****j, j′****)*** $\in \{0, \ldots, K-1\}^2$ 和无标签，分别取代 ***1*** 和 ***0***；当且仅当 ***(****D****(ar)*** $D(\boldsymbol{a\dot{r}})$ ***) =(****j, j′****)*** 时，才会为 $\boldsymbol{a}$ 选择 ***(****j, j′****)-***边。单测试节点引出的边的标签自然从 $j \in \{0,1\}$ 扩展为 $j \in \{0, \ldots, K-1\}$；当且仅当 $D(\boldsymbol{ar}) = j$ 时，才会选择标签为 j 的边。

4. 算法

我们将用于 OBDD 的算法 QLearn-π-OBDD [4] 扩展为一种用于 OMTBDD 的算法。从一个简单的假设 OBDD 开始，QLearn-π-OBDD 会针对当前假设 OBDD 反复提出等价查询，并利用查询得到的反例更新该假设，直到等价查询返回“是”为止。扩展后算法的基本结构与 QLearn-π-OBDD 相同。本节将说明如何扩展 QLearn-π-OBDD。

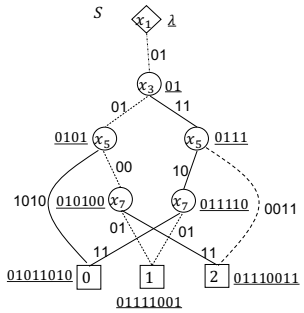
4.1. 假设数据结构

在 QLearn-π-OBDD 中，当前假设会与额外信息一同存储，以便轻松更新。它将当前假设保存为带有访问字符串的 ***OBDD***（***OBDDAS***）和（节点）分类树。为了处理两个以上的函数值，必须修改节点分类树；这些树已在上一节中扩展。接下来，我们将介绍如何把 OBDD 的 OBDDAS 扩展为 OMTBDD 的 OBDDAS，除汇点数量外无需修改。

一种带访问字符串的 *OMTBDD*（*OMTBDDAS*） S 表示某个 OMTBDD, 记为 $\mathcal{D}(S)$，并在边和节点中存储额外信息。它与 OMTBDD 的区别如下：

- 其根节点始终是一个标签为 x_1的节点，该节点可能是只有一条出边的虚拟节点。

- 边的标签是二进制字符串，而不是 0 或 1。标签为 x_i 和 x_j 的节点之间边标签字符串的长度为 $|i-j|$。如果一条边从标签为 x_i 的节点指向汇点，则其长度为 $m+1-i$。从同一节点发出的两条边的首位比特必须不同。



- 每个节点都有一个标识字符串：如果该节点不是虚拟节点，则它属于所表示的 OMTBDD $\mathcal{D}(S)$ 的某个节点标识符集合 $V(S)$；如果该节点是虚拟节点，则其标识字符串为 λ（空字符串）。

所表示的 OMTBDD $\mathcal{D}(S)$ 是通过从 OMTBDDAS S 中移除虚根及其出边, 并从所有边的标签字符串中删除除首位比特以外的所有比特（同时丢弃所有节点所带的标识字符串）而得到的。我们将 $E(S)$ 定义为 S中的边集合, 该集合表示为 $V(S) \times V(S)$ 的子集。 S 中边 $(\boldsymbol{u}, \boldsymbol{v}) \in E(S)$ 的标签字符串记为 $l(\boldsymbol{u}, \boldsymbol{v})$。

示例 2. 上图展示了一个 *OMTBDDAS* S 。 S 的根节点是一个虚拟节点，每个节点的标识符是写在节点旁边并带有下划线的字符串，它属于节点标识符集合 $V(S)$ ，该集合是 $\mathcal{D}(S)$ 的子集，但不包含虚拟节点的节点标识符 λ。这里， λ 表示一个空字符串。图*1* 中央展示了由 *OBDDAS* S 表示的、节点标识符集合为 $V(S)$ 的 *OMTBDD*$\mathcal{D}(S)$ 。

与 QLearn-π-OBDD 对未知目标 OMTBDD D 的做法相同[2], ，我们的扩展会构建假设 OMTBDDAS S 和节点分类树T_1， T_2，……， T_m 以满足以下条件 CN、CT 和 CE。注意

[2]条件 CN、CT 和 CE 与 [4] 中的条件 C1、C2 和 C3 相同。

，使用假设节点分类树进行分类时，是通过成员查询而非 $\mathcal{D}(S)$ D 来完成的，尽管非叶标签是$V(S)$中的字符串μ 。因此，对于任意分类树 T_i,

P1. $\forall \boldsymbol{v}_1$, $\forall \boldsymbol{v}_2 \in$ 满足条件的节点(D)。

CN. [节点条件](1)$V(S) \subseteq$ 节点(D)；(2)
$\forall \boldsymbol{v} \in V(S)_m[\mathcal{D}(S)(\boldsymbol{v}) = D(\boldsymbol{v})]$, 以及$_D(3)\forall \boldsymbol{v}_1$,
$\forall \boldsymbol{v}_2 \in V(S)[\boldsymbol{v}_1 \neq \boldsymbol{v}_2 \Rightarrow \boldsymbol{v}_1 \neq \boldsymbol{v}_2]$。

CT. [节点分类树条件]
(1)$\forall \boldsymbol{v} \in V(S)[T_{|\boldsymbol{v}|}(\boldsymbol{v}) = \boldsymbol{v}]$, 以及 (2)$\forall i \in \{1, ..., m\}$,
$\forall \boldsymbol{a} \in \{0,1\}^i[\boldsymbol{a} \notin$ 节点$(D) \Rightarrow T_i(\boldsymbol{a}) = \mu]$。

CE. [边条件] 对于所有 $(\boldsymbol{u}, \boldsymbol{v}) \in E(S)$, (1)$T_{|\boldsymbol{v}|}(\boldsymbol{u} \cdot l(\boldsymbol{u}, \boldsymbol{v}))$
$= \boldsymbol{v}$, 以及 (2)$|\boldsymbol{u}| < \forall j < |\boldsymbol{v}|$, $T_j(\boldsymbol{u} \cdot \text{pre}(l(\boldsymbol{u}, \boldsymbol{v}) j - |\boldsymbol{u}|))$
$= \mu$。

条件 CN 是针对节点标识符集合 $V(S)$ 的条件：(1) $V(S)$ 中的任何元素都必须是 D 中某个节点的访问字符串；(2) 对于 $V(S)$ 中所有长度为 m 的字符串， $\mathcal{D}(S)$ 必须与 D 具有相同的值；(3) $V(S)$ 中任意不同的字符串必须到达 D 中不同的节点。条件 CT 是针对假设节点分类树 T_1、T_2、.....、T_m 的条件：(1) 任何节点标识符都必须被分类为其自身；(2) 任何非访问字符串都必须被分类为 μ。条件 CE 是针对 $\mathcal{D}(S)$ 中边的条件：对于任意边，(1) 其起始节点标识符与其标签字符串的拼接串必须被分类为其目标节点标识符；(2) 其起始节点标识符与其标签字符串的任意前缀的拼接串必须被分类为 μ。

请注意，假设节点分类树可能不完整。因此，对于 $T_i (i = 1, .., m)$ 中某个单测试节点的标签 $\boldsymbol{r}$，对于某些 $\boldsymbol{a} \in \{0,1\}^i$，可能不存在标签为 $D(\boldsymbol{ar})$ 的边。在这种情况下，我们将 $T_i(\boldsymbol{a})$ 定义为从根节点开始，在双测试节点和单测试节点处反复执行边选择操作后到达的最后一个单测试节点的标签。

算法 1QLearn-OMTBDD()

输出： 目标函数的约简有序多终端二元决策图

1: (Ans e') $\leftarrow$ 等于(0)
2: **如果** Ans =是 **则返回 0**
3: $\ell' \leftarrow D(e')$, (Ans e) $\leftarrow$ EQ(ℓ')
4: **如果** Ans =是 **则返回** ℓ'
5: $\ell \leftarrow D(e)$、(S、T_1、T_2、..... T_m) $\leftarrow$ Initial-Hypothesis((e'、ℓ') e)
6: **循环**
7: **如果** $S(e) = \ell$ **则**
8: (答案 e) $\leftarrow$ EQ($\mathcal{D}(S)$)
9: **如果** 答案 =是 **则返回** $\mathcal{D}(S)$
10: $\ell \leftarrow D(e)$
11: (S, T_1, T_2, ..., T_m) $\leftarrow$ 更新假设(S, T_1, T_2, ..., T_m, e)

算法 2 更新假设(S, T_1, T_2, ..., T_m, e)

输出： 更新后的 OMTBDDAS S和节点分类树 T_1, T_2, ..., T_m

1: (p_1, p_2, ..., p_k) $\leftarrow$ 反例经过的 S 节点 ID 序列
e 按经过顺序。
2: $i \leftarrow$ 满足条件的索引 i s.t. $1 \le i < k$ 和 $D(p_i\text{suf}(e, m - |p_i|)) \neq D(p_{i+1}\text{suf}(e, m - |p_{i+1}|)) = D(p_k)$
3: **如果** $D(p_i l(p_i, p_{i+1})\text{suf}(e, m - |p_{i+1}|)) \neq D(p_k)$ **那么**
4: $(S, T_1, T_2, ..., T_m) \leftarrow \text{NodeSplit}(S, T_1, T_2, ..., T_m, e)$
5: **否则**
6: $(S, T_1, T_2, ..., T_m) \leftarrow$ 新分支节点($S, T_1, T_2, ..., T_m, e, p_i, p_{i+1}$)
7: **返回** $(S, T_1, T_2, ..., T_m)$

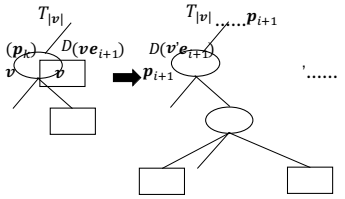
4.2. 伪代码

算法 QLearn-π-OBDD 的 OMTBDD 版本扩展称为 QLearn-OMTBDD，其伪代码见算法 1–5。主要扩展如下：

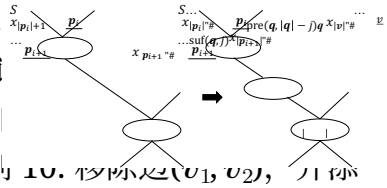
- 与 QLearn-π-OBDD 一样，一开始就找到所有汇点并不容易，因此扩展算法最初只找到两个汇点，之后再在 AddEdge（算法 5）的第 10 行添加必要的汇点。
- 当算法向某个节点分类树添加一个单测试节点时，像 QLearn-π-OBDD 那样当即查找并添加它的所有子叶节点似乎效率不高，因此扩展后的算法在 NodeSplit 的第 3 行只向这个新单测试节点添加两个子叶节点

算法3 NodeSplit($S, T_1, T_2, ..., T_m, \boldsymbol{e}, \boldsymbol{p}_i, \boldsymbol{p}_{i+1}$) 输出：更新后的 O

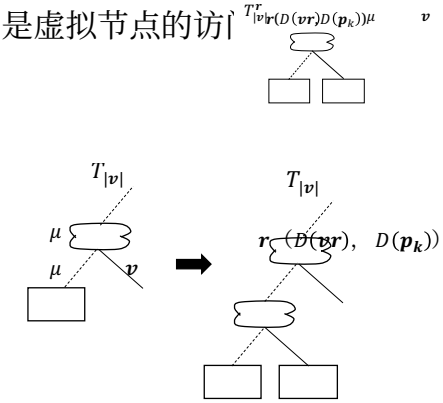
MTBDDAS S 以及节点分类树$T_1, T_2, ..., T_m$1： $\boldsymbol{e}_{i+1} \leftarrow \text{suf}(\boldsymbol{e}, m - |\boldsymbol{p}_{i+1}|)$，$\boldsymbol{v} \leftarrow \boldsymbol{p}_i l(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$ 2：添加一个标记为 $x_{|\boldsymbol{v}|+1}$ 的节点 $\boldsymbol{v}$，并添加一条标记为 $l(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$、连接$\boldsymbol{p}_i$和 $\boldsymbol{v}$的边，然后从 S中移除边 $(\boldsymbol{v}, \boldsymbol{p}_{i+1})$。 3：创建一棵树 $T_{|\boldsymbol{v}|}^{\boldsymbol{e}_{i+1}}$，它由一个标记为 $\boldsymbol{e}_{i+1}$ 的单测试根节点和两个子节点组成，这两个子节点分别标记为 $\boldsymbol{p}_{i+1}$ 和 $\boldsymbol{v}$，并分别通过 的边连接。 4： $\boldsymbol{t} \leftarrow$ 从根节点到标记为$\boldsymbol{p}_{i+1}$的叶节点的路径上， 中最后一个双测试节点的标签。 5：通过执行 AddEdge($\boldsymbol{v}, \boldsymbol{t}$) 和 Adc 从 $\boldsymbol{v}$ 添加两条新出边，其中 $\dot{\boldsymbol{t}}$ 表示将 $\boldsymbol{t}$的首位比特翻转后得到的 6：对所有满足 $(\boldsymbol{v}_1, \boldsymbol{p}_{i+1}) \in E(S)$ 的 $\boldsymbol{v}_1 \in V(S)$ 执行 7：对 $\boldsymbol{v}_1\boldsymbol{q}'\boldsymbol{e}_{i+1}$ 发起成员查询，并获得 $D(\boldsymbol{v}_1\boldsymbol{q}'\boldsymbol{e}_{i+1})$，其中$\boldsymbol{q}' = l(\boldsymbol{v}_1, \boldsymbol{p}_{i+1})$。 8：如果 $T_{|\boldsymbol{v}|}^{\boldsymbol{e}_{i+1}}$ 有一条标记为 $D(\boldsymbol{v}_1 l(\boldsymbol{v}_1, \boldsymbol{p}_{i+1})\boldsymbol{e}_{i+1})$ 的边，则 9：如果 $D(\boldsymbol{v}_1\boldsymbol{q}'\boldsymbol{e}_{i+1}) \neq D(\boldsymbol{p}_k)$，则 10：移除边 $(\boldsymbol{v}_1, \boldsymbol{p}_{i+1})$ 并添加边 $(\boldsymbol{v}_1, \boldsymbol{u})$，其中 $\boldsymbol{u}$ 是 $T_{|\boldsymbol{v}|}^{\boldsymbol{e}_{i+1}}$ 中入边标记为 $D(\boldsymbol{v}_1\boldsymbol{q}'\boldsymbol{e}_{i+1})$的叶节点的标签。 11：否则 12：添加一个标记为 $x_{|\boldsymbol{v}'|+1}$ 的节点 $\boldsymbol{v}' = \boldsymbol{v}_1\boldsymbol{q}'$，并添加一条标记为 $\boldsymbol{q}'$、连接$\boldsymbol{v}_1$和 $\boldsymbol{v}'$的边，然后从 S中移除边 $(\boldsymbol{v}_1, \boldsymbol{p}_{i+1})$。 13：向 $T_{|\boldsymbol{v}|}^{\boldsymbol{e}_{i+1}}$ 添加一个标记为 $\boldsymbol{v}'$ 的叶节点，作为根节点的子节点，并添加一条标记为 $D(\boldsymbol{v}_1\boldsymbol{q}'\boldsymbol{e}_{i+1})$的边。 14：通过执行 AddEdge($\boldsymbol{v}', \boldsymbol{t}$) 和 AddEdge($\boldsymbol{v}', \dot{\boldsymbol{t}}$)，从$\boldsymbol{v}'$ 添加两条新出边。 15：将树$T_{|\boldsymbol{v}|}$ 中标记为$\boldsymbol{p}_{i+1}$ 的叶节点替换为树 $T_{|\boldsymbol{v}|}^{\boldsymbol{e}_{i+1}}$ 。



算法 4 NewBranchingNode($S, T_1, T_2, ..., T_m, \boldsymbol{e}, \boldsymbol{p}_i, \boldsymbol{p}_{i+1}$) 输出：更新后的 OMTBDDAS S 以及节点分类树$T_1, T_2, ..., T_m$1: $\boldsymbol{e}_i \leftarrow$ suf($\boldsymbol{e}, m - |\boldsymbol{p}_i|$), $\boldsymbol{e}_{i+1} \leftarrow$ suf($\boldsymbol{e}, m - |\boldsymbol{p}_{i+1}|$), $\boldsymbol{q} \leftarrow l(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$, $\boldsymbol{f} \leftarrow$pre($\boldsymbol{e}_i, |\boldsymbol{q}|$) 2: $j \leftarrow j$满足 $1 \leq j \leq |\boldsymbol{q}|$ 使得 $D(\boldsymbol{p}_i$cro($\boldsymbol{q}, \boldsymbol{f}, j)\boldsymbol{e}_{i+1}) \neq D(\boldsymbol{p}_i$cro($\boldsymbol{q}, \boldsymbol{f}, j-1)\boldsymbol{e}_{i+1}) = D(\boldsymbol{p}_k)$ 3: $\boldsymbol{v} \leftarrow \boldsymbol{p}_i \cdot$ pre($\boldsymbol{q}, |\boldsymbol{q}| - j$), $\boldsymbol{r} \leftarrow$ suf($\boldsymbol{e}, m - |\boldsymbol{v}|$) 4: 如果 $j \neq |\boldsymbol{q}|$ 则 5: 向 S添加一个标签为$x_{|\boldsymbol{v}|+1}$ 的节点 $\boldsymbol{v}$ ，以及标签为 pre($\boldsymbol{q}, |\boldsymbol{q}| - j$) 的边($\boldsymbol{p}_i$, $\boldsymbol{v}$)和标签为 suf($\boldsymbol{q}, j$) 的边($\boldsymbol{v}, \boldsymbol{p}_{i+1}$)，并从 S中移除边($\boldsymbol{p}_i, \boldsymbol{p}_{i+1}$)。 6: 创建一棵树 $T^{\boldsymbol{r}}_{|\boldsymbol{v}|}$，由一个标签为 r 的双测试节点及其



子节点组成。将标签($D(\boldsymbol{vr})$ $D(\boldsymbol{p}_k)$)附加到双测试
之间的边上。 7: 对所有满足 $|\boldsymbol{v}_1| < |\boldsymbol{v}| < |\boldsymbol{v}_2|$
pre($l(\boldsymbol{v}_1, \boldsymbol{v}_2)$ $|\boldsymbol{v}| - |\boldsymbol{v}_1|$) 9: 如果 $T^{\boldsymbol{r}}_{|\boldsymbol{v}|}(\boldsymbol{v}_1\boldsymbol{g}) = \boldsymbol{v}$ 则 10: 移除边($\boldsymbol{v}_1, \boldsymbol{v}_2$)，并添加一条标签为 $\boldsymbol{g}$的边($\boldsymbol{v}_1, \boldsymbol{v}$)。 11: 将树 $T_{|\boldsymbol{v}|}$ 中标签为 μ 的叶节点替换为树 $T^{\boldsymbol{r}}_{|\boldsymbol{v}|}$。 12: 否则 13: 不执行任何操作。（$\boldsymbol{p}_i$ 是虚拟节点的访问



14: 通过执行 AddEdge($\boldsymbol{v}, \boldsymbol{r}$)，从 $\boldsymbol{v}$ 添加一条新的出边。
15: **返回** $(S, T_1, T_2, ..., T_m)$

（算法 3），并在尝试访问其他子叶节点时将其添加（NodeSplit 的第 13行和 AddEdge 的第8行）。

- 每当向分类树中的某个单测试节点添加一个子叶节点时，都会伴随着发现并添加 D 中的一个节点（NodeSplit 第 12 行和 AddEdge 第 12 行）。因此，在一次执行更新假设 S 期间，可以向假设带访问字符串的有序多值终端二叉决策图D 中添加多个节点

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

对于 $j = 1$ 到 $|t|$ **执行**
  $\boldsymbol{u} \leftarrow T_{|\boldsymbol{v}|+j}$($\boldsymbol{v}$前置条件($\boldsymbol{t}$, j))
  如果 $\boldsymbol{u} \neq \mu$ **则**
    如果 标记为 $\boldsymbol{u}$ 的节点是 $T_{|\boldsymbol{v}|+j}$ 的叶节点**则**
      添加一条边（$\boldsymbol{v}$, $\boldsymbol{u}$），其标签为 pre($\boldsymbol{t}$, j)。
    else
      $\boldsymbol{u}' \leftarrow \boldsymbol{v}$前置条件($\boldsymbol{t}, j$)
      添加一个标记为 $\boldsymbol{u}'$ 的叶节点，作为标记为 $\boldsymbol{u}$ 的节点的子节点 $T_{|\boldsymbol{v}|+j}$，并添加一条标记为 D 的边（$\boldsymbol{u}'\boldsymbol{u}$）。
      如果 $j = m - |\boldsymbol{v}|$，**则**
        添加一个新的汇点 $\boldsymbol{u}'$，标记为 D（$\boldsymbol{u}'$），并添加一条边（$\boldsymbol{v}$, $\boldsymbol{u}'$），其标签为 pre($\boldsymbol{t}$, j) 到 S。
      else
        添加一个新节点 $\boldsymbol{u}'$，并添加一条边（$\boldsymbol{v}$, $\boldsymbol{u}'$），其标签为 pre（$\boldsymbol{t}$, j）至 S。
        执行 AddEdge（$\boldsymbol{u}'$, $\boldsymbol{t}$）和 AddEdge（$\boldsymbol{u}'$, $\dot{\boldsymbol{t}}$），其中 $\dot{t}$ 表示将 t 的首位比特翻转后得到的字符串。
  返回

，而这种情况在 QLearn-π-OBDD 中从未发生。

首先，QLearn-OMTBDD 针对一个简单的 OMTBDD 发起等价查询（EQ），该 OMTBDD 记为 **0**，仅由一个标记为 0 的汇点组成。如果查询返回“是”，算法就输出假设 **0** 并停止。如果答案为“否”，且等价查询返回反例 $\boldsymbol{e}'$，那么算法会针对赋值 $\boldsymbol{e}'$ 发起成员查询，并得到 $\ell' = D(\boldsymbol{e}')$，其中 D 是目标 OMTBDD。接下来，算法针对另一个简单的 OMTBDD 发起等价查询$\boldsymbol{\ell}'$，该 OMTBDD 仅由一个标记为 ℓ' 的汇点组成。如果查询返回“是”，算法就输出假设 $\boldsymbol{\ell}'$ 并停止。如果答案为“否”且返回反例 $\boldsymbol{e}$，那么算法会通过发起成员查询得到 $\ell = D(\boldsymbol{e})$。然后，算法通过 Initial-Hypothesis 算法，根据两个反例 $\boldsymbol{e}'$ 和 $\boldsymbol{e}$，以及 $D(\boldsymbol{e}') = \ell'$ 和 $D(\boldsymbol{e}) = \ell$，构造初始 OMTBD-DAS 和节点分类树。

Initial-Hypothesis 算法的工作方式如下。由于 $\boldsymbol{e}' = \text{cro}(\boldsymbol{e}', \boldsymbol{e}, 0)$,$\boldsymbol{e} = \text{cro}(\boldsymbol{e}', \boldsymbol{e}, m)$ 和 $D(\boldsymbol{e}') = \ell' \neq D(\boldsymbol{e})$，因此存在满足 $0 < i \leq m$ 的 i，使得 $D(\text{cro}(\boldsymbol{e}', \boldsymbol{e}, i-1)) = \ell' \neq D(\text{cro}(\boldsymbol{e}', \boldsymbol{e}, i))$，而这样的 i 可以

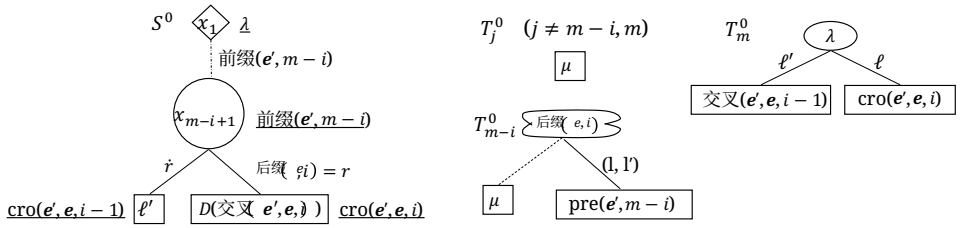


图2：初始 OMTBDDAS S^0和针对 $j = 1$、.....、m 的初始分类树 T_j^0： T_{m-i}^0有一个双测试节点， T_m^0 有一个单测试节点，其余 T_j^0仅由一个带有标签 μ 的叶节点组成。

通过使用 $\lceil \log_2 m \rceil$ 次成员查询进行二分搜索找到。图2展示了过程构造的初始 OMTBDDAS S^0 以及针对 $j = 1$、.....、m 的初始节点分类树 T_j^0。注意， S^0 和 $\{T_1^0, ..., T_m^0\}$ 满足条件 CN、CT 和 CE。

假设算法当前有 OMTBDDAS S 以及针对 $i = 1$、..、m 的当前节点分类树 T_i。令 $\boldsymbol{e}$ 为最后一个反例，并令 $\ell = D(\boldsymbol{e})$。如果 $S(\boldsymbol{e}) \neq \ell$，那么 $\boldsymbol{e}$ 仍是当前假设 OMTBDDAS S的一个反例。否则，算法会针对 $\mathcal{D}(S)$ 发起等价查询，并在返回‘是’时输出它。返回‘否’时，可以得到一个新的反例 $\boldsymbol{e}$ ，然后针对$\boldsymbol{e}$ 发起成员查询，以获取 $\ell = D(\boldsymbol{e})$。算法使用反例 $\boldsymbol{e}$ 执行算法 2 中所示的 Update-Hypothesis 算法。重复这一过程，直到等价查询返回‘是’。

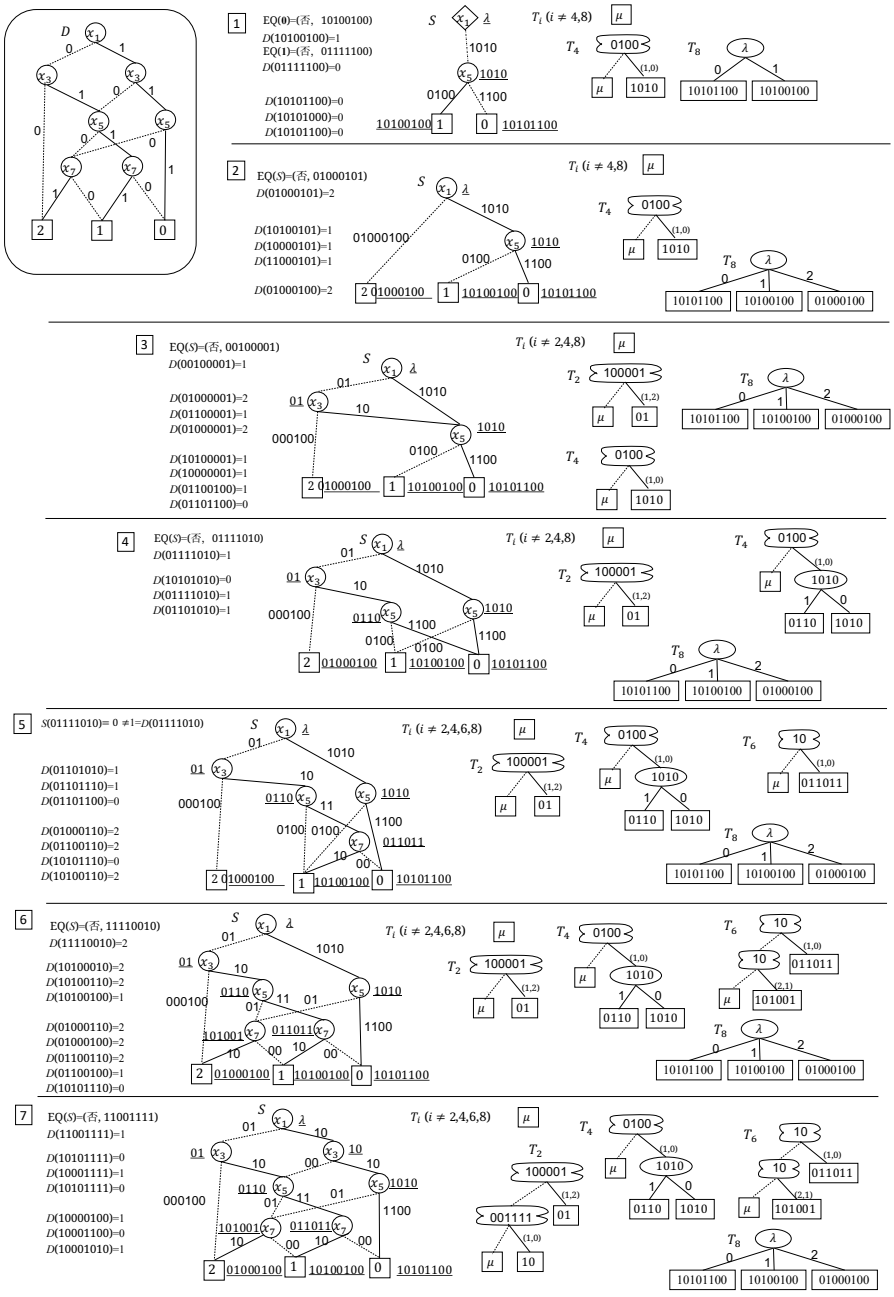
每次执行更新假设过程，都会找到目标约简 OMTBDD 中至少一个节点，并更新当前假设。考虑由给定反例 $\boldsymbol{e}$ 在 S 中形成的路径。假设该路径上有 k 个节点，并令 $\boldsymbol{p}_i$ 为从根节点起该路径上第 i 个节点的标识字符串。由于 $\boldsymbol{e}$ 是 S 的反例，因此沿该路径到达的叶节点 $\boldsymbol{p}_k$ 不正确，也就是说， $D(\boldsymbol{p}_k) \neq D(\boldsymbol{e})$。令$\boldsymbol{e}_i = \text{suf}(\boldsymbol{e}, m - |\boldsymbol{p}_i|)$。由于 $\boldsymbol{p}_k = \boldsymbol{p}_k\boldsymbol{e}_k$ 且 $\boldsymbol{e} = \boldsymbol{p}_1\boldsymbol{e}_1$，必定存在 i，使得 $1 \leq i < k$ 且 $D(\boldsymbol{p}_i\boldsymbol{e}_i) \neq D(\boldsymbol{p}_{i+1}\boldsymbol{e}_{i+1}) = D(\boldsymbol{p}_k)$。通过成员查询，可在第 2 行计算出这样的 i 。对于这个 i，令 $\boldsymbol{q}$ 表示边 $(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$ 的标签，即 $\boldsymbol{q} = l(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$。根据 $D(\boldsymbol{p}_i\boldsymbol{q}\boldsymbol{e}_{i+1})$ 的值，可分为两种情况。当 $D(\boldsymbol{p}_i\boldsymbol{q}\boldsymbol{e}_{i+1}) \neq D(\boldsymbol{p}_{i+1}\boldsymbol{e}_{i+1})$ 时， $\boldsymbol{p}_i\boldsymbol{q}$ 和 $\boldsymbol{p}_{i+1}$ 必须到达 D 中不同的节点；算法在 NodeSplit 算法（算法 3）中处理这种情况，并在其中一个分类树中至少添加一个单测试节点

。当 $D(\boldsymbol{p}_i\boldsymbol{q}\boldsymbol{e}_{i+1}) = D(\boldsymbol{p}_{i+1}\boldsymbol{e}_{i+1}) = D(\boldsymbol{p}_k)$ 时， $\boldsymbol{p}_i$ 和 $\boldsymbol{p}_{i+1}$ 之间必定存在一个节点， $\boldsymbol{p}_i\boldsymbol{e}_i$ 的路径和 $\boldsymbol{p}_i\boldsymbol{q}\boldsymbol{e}_{i+1}$ 的路径从该节点处分支；算法在新分支节点算法（算法 4）中处理这种情况，并在其中一个分类树中至少添加一个双测试节点。这两种算法都会向当前 OMTBDDAS 添加一个新节点 $\boldsymbol{v}$ ，更新 $T_{|\boldsymbol{v}|}$，更改所有必须进入 $\boldsymbol{v}$ 的边，并添加从 $\boldsymbol{v}$ 发出的边。

4.3. 算法执行示例

图3展示了 QLearn-OMTBDD 的执行示例。由 Initial-Hypothesis((10100100, 1), 01111100)) 构造的 OMTBDDAS 和节点分类树（$S, T_1, \ldots, T_8$）显示在 1 中。由于 S (01111100) $= 0$，QLearn-OMTBDD 在第 8 行发起等价查询，并假设获得了一个反例 01000101，且 D(01000101) $= 2$ 。将 01000101 作为访问字符串序列传入后，在 Update-Hypothesis 的第 1 行得到 ($\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3$) =(λ, 1010, 10100100)。在 Update-Hypothesis 的第 2 行， i 被设为 1，因为 $2 = D$(01000101) $= D(\boldsymbol{p}_1$ suf(01000101, 8) $\neq D(\boldsymbol{p}_2$suf(01000101, 4) $= D$(10100101) $= 1$ 成立。由于 $1 = D$ (10100101) $= D(\boldsymbol{p}_1 l(\boldsymbol{p}_1, \boldsymbol{p}_2)$suf(01000101, 4)) $= D(\boldsymbol{p}_3)$ 成立，因此在第 6 行执行算法 NewBranchingNode。在 NewBranchingNode 的第 2 行， j 被设为 $4 = |\boldsymbol{q}|$，因为 $2 = D$(01000101) $= D(\lambda$cro(1010, 0100, 4)0101) $\neq D(\lambda$cro(1010, 0100, 3)0101) $= D$ (11000101) $= 1$。因此执行第13行，虚根变为非虚根。由于在第 3 行将 $\boldsymbol{r}$ 设为 01000101，通过执行算法 AddEdge(λ, 01000101)（算法 5），从根节点引出的一条边 λ 被添加。（参见 2。）在算法 AddEdge 中，对于所有 $j = 1, \ldots, 7$， $T_{|\lambda|+j}(\lambda$ pre(01000101, j)) $= T_j$(pre(01000101, j)) $= \mu$，并且 T_8(01000101) $= \lambda$，因为从 T_8 的根节点引出的边没有标记为 2。随后，在 T_8 中，将标记为 01000101 的叶节点添加为标记为 λ 的节点的子节点（AddEdge 中的第 8 行），并将标记为 2 的汇点以及一条标记为 01000101 的边 (λ, 01000101) 添加到 S （AddEdge 中的第 10 行）。

NodeSplit 算法在第 4 行执行。对于 3 中所示的 S，假定获得了一个反例 01111010。01111010 经过的访问字符串序列为 ($\boldsymbol{p}_1$， $\boldsymbol{p}_2$， $\boldsymbol{p}_3$， $\boldsymbol{p}_4$) =(λ，01，1010，10101100)，并且 i 设为 2，因为在更新假设的第 2 行， $1 = D$(01111010) $= D(\boldsymbol{p}_2$ suf(01111010，6) $\neq D(\boldsymbol{p}_3$suf(01111010，4) $= D$(10101010)。在这种情况下， $1 = D$ (01101010) $= D(\boldsymbol{p}_2 l(\boldsymbol{p}_2$， $\boldsymbol{p}_3)$suf(01111010，4)) $\neq D(\boldsymbol{p}_4) = 0$ 成立，因此 NodeSplit 算法在第 4 行执行。在 NodeSplit 算法中，节点 0110 从节点 1010 分裂出来（第 2 行），T_4 中标记为 1010 的叶节点被替换（第 15 行），替换为



Fi图 3：QLearn-OMTBDD算法构建 OMTBDDAS 的示例

一个标签为 1010 的单测试节点，它有一个标签为 0110 的子节点，通过标签为 1 的边与之相连；另一个标签为 1010 的子节点则通过标签为 0 的边与之相连（第 3 行）。拆分该节点后，最后一个反例 01111010 对图 4 中更新后的 S 仍然是反例，也就是说，$0 = S(01111010) \neq \ell = D(01111010) = 1$，因此 QLearn-OMTBDD 中的第 8-10 行不会执行，而会针对相同的 (e, ℓ) 执行 Update-Hypothesis。在从图 5 更新到图 6 的过程中，算法 NewBranchingNode 将标签为 x_7 的节点 101001 添加到 S中，并将边 (1010, 10100100) 替换为边 (1010，101001)。这是在执行第 7-10 行的 FOR循环期间完成的。

由于图7所示的 OMTBDDAS S 对应的 OMTBDD 等价于 D，因此在 QLearn-OMTBDD 第9行，对 S 的等价查询的答案为“是”，并输出 $\mathcal{D}(S)(= D)$。

4.4. 正确性与效率

定理1. 对于任意目标约简 *OMTBDD* D，算法 *QLearn-OMTBDD* 至多使用 n 次等价查询和至多$2n(\lceil \log_2 m \rceil + 3n)$ 次成员查询，即可精确学习 D，其中 $m(\geq 1)$ 是变量数，n是 D中的节点数。

证明。 见附录A。 □

假设对长度为 m 的字符串进行所有操作最多需要$O(m)$步，则容易证明，运行时间至多为 $O(nm(\log m+n))$，即比查询次数大 $O(m)$ 倍。

5. 实验

我们使用一个合成数据集以及 UCI机器学习资料库中的若干基准数据集开展实验，以展示我们算法的有效性。

5.1. 合成数据集

我们使用合成数据集研究了算法的经验样本复杂度。我们针对不同的（n，m，K），随机生成了节点数为 n、变量数量为 m、叶节点数量为 K 的OMTBDD。对于每个（n，m，K），生成10个OMTBDD：(1) 取 $n = 1$、2、4、8、16、32、64、128、256、512（$\times 10^2$），并固定 $m = 3200$ 和$K = 32$；(2) 取 $m = 1$、2、4、8、16、32、64、128、256、512（$\times 10^2$），并固定

$n = 3200$和 $K = 32$；以及(3) 取 $K = 2$、4、8、16、32、64、128、256、512，并固定 n、$m = 3200$，采用与OBDD生成过程 [4] 类似的过程（见附录B）。参数列表为（n，m，K）的OMTBDD的查询次数取这10个OMTBDD的平均值。

结果如图4所示。左列中的表格和折线图展示成员查询的结果，右列中的表格和折线图展示等价查询的结果。两个坐标轴均采用对数刻度。表格中的数值均四舍五入至三位有效数字。图中还展示了理论上界（成员查询为 $2n(\lceil \log_2 m \rceil + 3n)$，等价查询为 n，其中 n、m 分别表示节点数和变量数量），以供比较。对于节点数，成员查询和等价查询次数的理论上界分别为 $O(n^2)$ 和 $O(n)$，图中第 (1) 行可以观察到这两种查询次数按相应阶数增长。对于变量数量，成员查询和等价查询的增长阶数分别为 $O(\log m)$ 和 $O(1)$，但如图中第 (2) 行所示，等价查询次数和成员查询次数看起来都几乎保持不变。成员查询次数的上界为 $6400\lceil \log_2 m \rceil + 6 \times 6400^2$，因此变量数量翻倍时需要额外进行 6400 次查询，而与 6×6400^2 相比，6400 是一个较小的数值。如图中第 (3) 行所示，两类查询的次数都不受叶节点数量影响，不过在叶节点数量非常少（$K = 2$，4）的情况下，成员查询次数似乎有所减少。不同函数的数量会随着叶节点数量增加而增多，这意味着函数类的复杂度也随之增加。这或许就是这一现象的原因。

5.2. 基准数据集

一种应用是将已学习的分类器转换为 OMTBDD。如果能够获得已学习分类器的紧凑 OMTBDD表示，那么它们更适合在资源受限设备上进行硬件实现，而且由于 OMTBDD 支持函数间的各种运算，因此更易于处理。

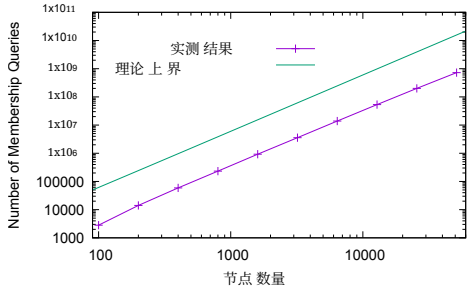
我们使用了 UCI机器学习资料库 [8]中登记的 12 个数据集，这些数据集仅由实值特征组成。（见表 1）我们针对每个数据集进行实验的流程如下。

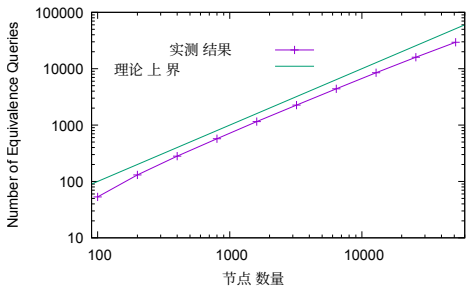
1. 使用给定的数据集学习基于树的分类器。2. 使用分支条件共享算法 Min DBN [9] 对组成决策树的分支条件进行约简。该基于树的

(1)

节点数/10^2	1	2	4	8	16
查询/10^4	0.281	1.41	5.90	23.2	92.6
	32	64	128	256	512
	357	1390	533020000	72600	

节点数/10^2	1	2	4	8	16
查询	53.2	131	281	573	1150
	32	64	128	256	512
	2260	4400	848016000	29400	

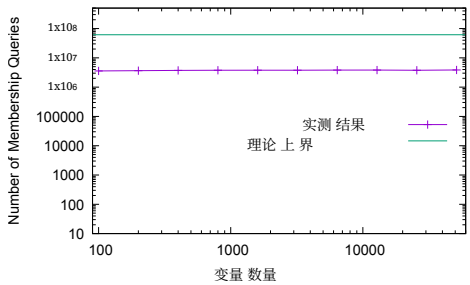


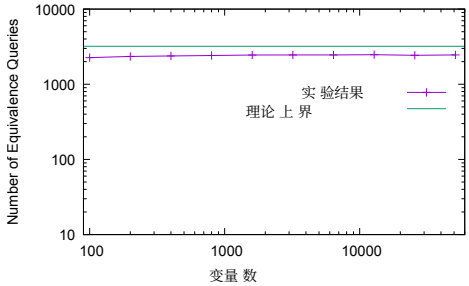


(2)

变量/10^2	1	2	4	8	16
查询/10^4	357	365	373	377378	
	32	64	128	256	512
	378	382	383377	386	

变量/10^2	1	2	4	8	16
查询/10	226	234	238	241245	
	32	64	128	256	512
	245	246	247243	246	

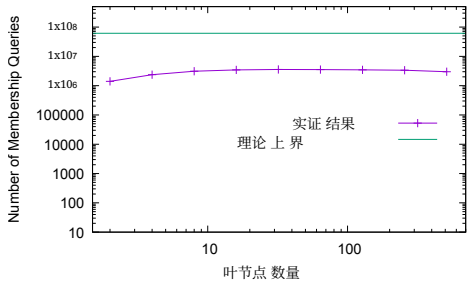




(3)

叶节点	2	4	8	16	32
查询/10^4	140	237	311345	357	
	64	128	256	512	
	354	347	337299		

叶节点	2	4	8	16	32
查询/10	245	198	205	219	226
	64	128	256	512	
	228	223	219203		



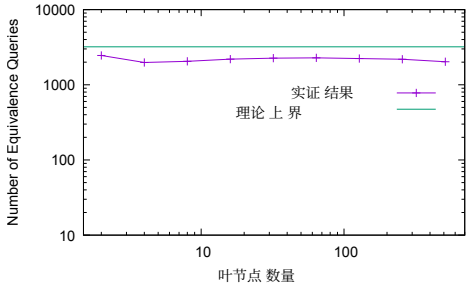


图4：不同节点数下的成员查询和等价查询数量（1）、不同变量数量下的成员查询和等价查询数量（2），以及不同叶节点数量下的成员查询和等价查询数量（3）。

分类器通过共享分支条件转换为更简单的分类器。

表 1：我们实验中使用的 UCI机器学习资料库数据集 [8] 。

数据集	#数据	#特征	#类别	详情
iris	150	4	3	Iris
帕金森病	195	22	2	帕金森病
乳腺癌	569	30	2	威斯康星乳腺癌（诊断）
血液	748	4	2	输血服务中心
RNA测序泛癌数据集	801	20531	5	癌症基因表达RNA测序
红葡萄酒质量	1599	11	11	葡萄酒质量
白葡萄酒质量	4898	11	11	葡萄酒质量
波形	5000	40	3	波形数据库生成器（第2版）
机器人	5456	24	4	循墙机器人导航
musk	6598	166	2	麝香（第2版）
癫痫发作	11500	178	5	癫痫发作识别
magic	19020	10	2	MAGIC伽马射线望远镜

3. 使用转换后分类器的每个不同分支条件作为二值特征，将每份训练数据转换为二元特征数据

。4. 二值特征的变量排序按如下方式确定。考虑一个由与二值特征对应的顶点组成的有向图。对于每一对二值特征（x_i，x_j），统计标签为 x_i 的节点是标签为 x_j 的节点的祖先的次数，以及反过来成立的次数。将x_i 和 x_j 对应顶点之间的边方向定义为：从祖先出现次数较多的特征对应的顶点指向另一特征对应的顶点。还将有向边的权重定义为两者祖先出现次数之差。从权重最小的边开始移除，直到整个顶点集合可以完成拓扑排序。将对应顶点的顺序定义为二值特征顺序。

5. QLearn-OMTBDD 使用转换后的带标签二元特征数据集学习 OMTBDD，并将转换后的基于树的分类器用作成员查询预言机。在等价查询中，若其与所有给定数据一致，则视为与目标函数等价；否则返回一条不一致的给定数据作为反例。请注意，我们只使用转换后的基于树的分类器预测正确的数据。

作为基于树的分类器，我们在实验中使用了决策树和随机森林。我们采用 scikit-learn 1.1.dev0 版本中的 DecisionTreeClassifier 和 RandomForest-

表2：从决策树学习得到的 OMTBDD 的节点数和准确率。

数据集	决策树					OMTBDD	
	节点数	（l. share）	准确率	#DC	#RDC	节点数	准确率
iris	14.6	(9.8)	0.947	6.8	6.8	9.8	0.947
帕金森病	25.4	(14.2)	0.856	12.2	11.8	14.2	0.856
乳腺癌	37.4	(20.2)	0.924	18.2	18.2	19.8	0.924
血液	320	(162)	0.713	106	72.2	342	0.722
RNA测序泛癌数据集	14.6	(11.8)	0.974	6.8	6.8	11.8	0.974
红葡萄酒质量	668	(345)	0.650	300	179	3811	0.652
白葡萄酒质量	2088	(1050)	0.604	769	370	29700	0.598
波形	797	(401)	0.735	397	304	3883	0.743
机器人	57.4	(32.7)	0.995	28.0	22.4	49.0	0.996
musk	252	(128)	0.965	126	121	123	0.964
癫痫发作	3690	(1850)	0.472	1830	1290	722000	0.452
magic	3200	(1600)	0.818	1600	771	49800	0.820

Classifier。RandomForestClassifier 中的组件树数量（n estimators）设为 100。除 n jobs 和 random state 外，RandomForestClassifier 的其他参数均采用默认值：n jobs$=-1$ （并行运行的任务数设为处理器数量），rāndom state$=0$ （随机选择的种子设为 0）。DecisionTreeClassifier 中的参数 random state（每次分割时特征随机排列的种子）设为 0，其他参数均采用默认值。我们通过节点数和准确率，评估所学习 OMTBDD 的紧凑性及其在测试数据集上的准确率；这些测试数据集通过五折交叉验证与训练数据集分离。最大的两个数据集 epileptic seizure 和 magic 过于庞大，我们的查询学习算法无法从使用它们训练出的随机森林分类器中学习 OMTBDD，因此我们在随机森林分类器的实验中将其排除。

决策树的结果见表2，随机森林的结果见表3。在表中，“#node”表示节点数，“(l.share)”表示其叶共享树分类器的节点数，其中相同标签的叶节点共享一个叶节点。OMTBDD 中相同标签仅有一个叶节点，因此，与原始基于树的分类器比较节点数时，应比较其叶共享形式的节点数。“#DC”表示原始基于树的分类器中不同分支条件的数量，“#RDC”表示经分支条件共享算法 Min DBN 约简后的分类器中的数量。所有数值均四舍五入至三位有效数字。对于简单分类器问题（原始决策树的节点数少于 30），其叶共享决策树已经是 OMTBDD，查询学习

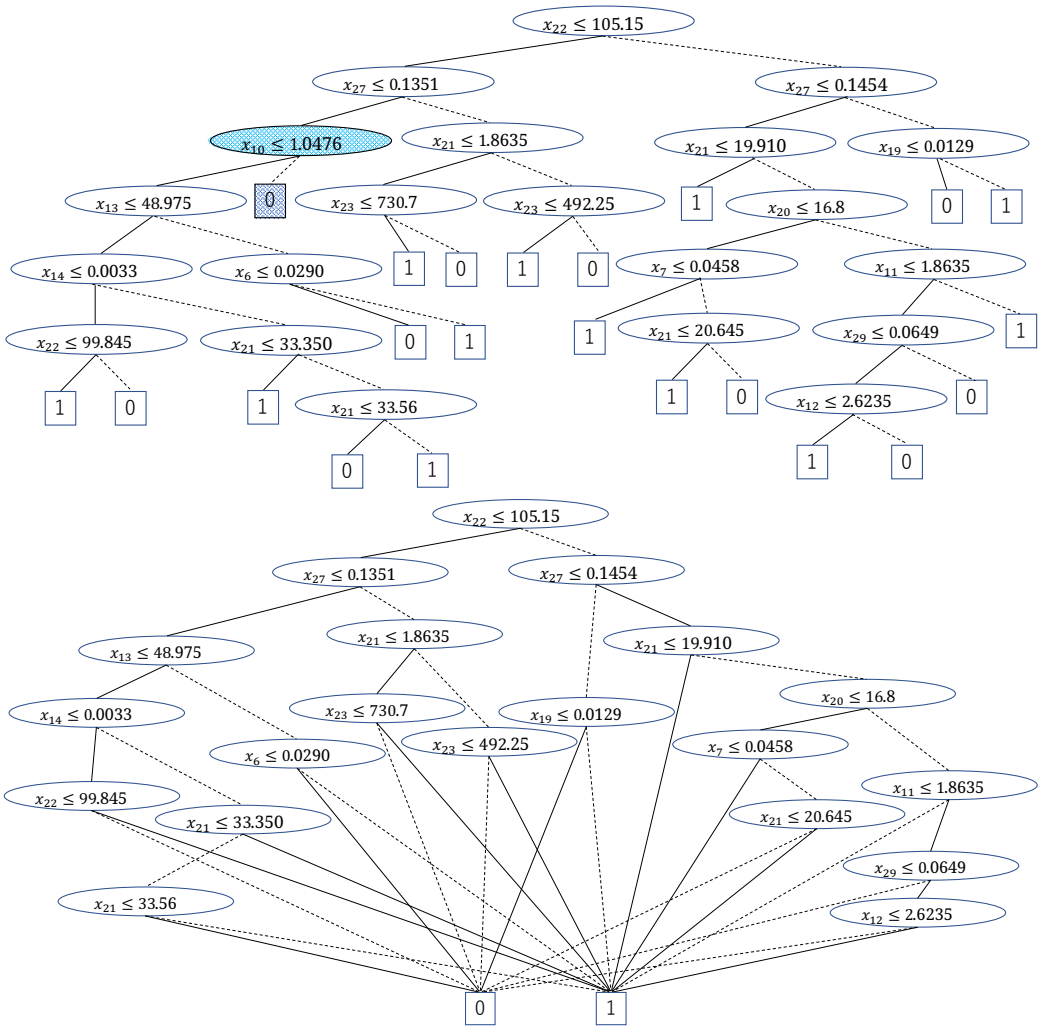


图5：乳腺癌数据集的决策树及从中学习得到的 OMTBDD .

算法能够精确地学习它们。（参见鸢尾花、帕金森病和 RNA-Seq PANCAN 数据集的结果。）乳腺癌和蘑香数据集的结果颇为有趣：对于这些数据集，OMTBDD 的规模小于原始叶共享决策树的规模。图5展示了我们对乳腺癌数据集进行五折交叉验证的一次运行结果。上图中决策树对应的叶共享树已经是 OMTBDD，而下图中的 OMTBDD 是从

表3：从随机森林学习得到的 OMTBDD 的节点数和准确率。

数据集	随机森林				OMTBDD	
	节点数（l. share）	准确率	#DC	#RDC	节点数	准确率
iris	1430 (965)	0.947	104	43.6	18.2	0.947
帕金森病	2660 (1480)	0.892	1010	404	147	0.872
乳腺癌	3570 (1940)	0.961	1420	594	203	0.944
血液	23200 (11800)	0.749	322	145	1520	0.723
RNA测序泛癌数据集	3960 (2430)	0.996	1920	1840	3540	0.885
红葡萄酒质量	55900 (29000)	0.692	4120	900	271000	0.598
白葡萄酒质量	181000 (91600)	0.679	7290	1400	2080000	0.594
波形	83800 (42200)	0.854	32500	5870	2010000	0.689
机器人	25000 (12900)	0.995	9900	3140	1010	0.996
musk	34600（17500）	0.975	12200	6240	574000	0.908

它。它们之间的区别在于阴影部分。原始决策树以自顶向下方式构建，在这种方式下，$x_{10} \leq 1.0476$ 是一个很好的分类条件。构建完所有后代节点后，发现所有被该条件分类为 0 的训练数据，在后代部分中也都被分类为 0。因此，这个条件并非必需。对于除癫痫发作数据集以外的其他较大型分类器，OMTBDD 的规模不到原始决策树叶共享决策树规模的 30 倍，且没有观察到明显的准确率下降。对于癫痫发作数据集，OMTBDD 的规模扩大了 165 倍，准确率也有所下降。与其他数据集相比，在给定变量排序下，癫痫发作数据集的决策树决策边界被认为更为复杂，而与给定数据一致的 OMTBDD 可能无法很好地近似原始决策树。至于随机森林分类器的结果，所有数据集上的原始分类器准确率均有所提高或保持不变。然而，从这些分类器学习得到的 OMTBDD，其准确率则保持不变或有所下降。对于通过学习得到的 OMTBDD 成功缩减规模的数据集——鸢尾花、帕金森病、乳腺癌、血液和机器人数据集——准确率下降幅度较小。特别是对于帕金森病和乳腺癌数据集，从随机森林分类器学习得到的 OMTBDD，其准确率优于从决策树分类器学习得到的 OMTBDD。这些 OMTBDD 的意义在于，其准确率优于决策树，且规模小于随机森林。对于其他 5 个数据集，准确率显著下降，规模则增大至原来的 1.5–48 倍。这 5 个数据集出现这种下降的原因，似乎与上述

使用癫痫发作数据集时决策树准确率下降的原因相同。

6. 结论与未来工作

我们通过将用于 OBDDs 的 QLearn-π-OBDD [4] 扩展到 OMTBDD，开发了用于 OMTBDD 的查询学习算法 QLearn-OMTBDD。在我们的算法中，对于长度为 $i-1$ 的赋值，长度为 $(i-1)$ 的前缀通过节点分类树，利用成员查询被分类为标签为 x_i 的节点，或分类为 μ （没有节点对应于该赋值前缀）；同时，每次成员查询的可能答案数量多于两个这一事实，阻碍了分类树及其运行过程的直接扩展。在使用基准数据集进行的实验中，我们通过为部分数据集构建紧凑且准确的 OMTBDD，展示了该算法应用于分类问题的可行性。另一方面，对于另一些数据集，学习得到的 OMTBDD包含大量节点且准确率较低。我们希望弄清楚对于此类数据集是否存在紧凑且准确的 OMTBDD。

致谢

本研究得到日本 JST CREST 项目（编号：JPMJCR18K3）的资助。

参考文献

[1]R. E. Bryant, 《使用有序二叉决策图进行符号布尔运算》, ACM计算调查 24（1992）293–318。[2] M. Fujita、P. C. McGeer、J. C. Y. Yang, 《多终端二元决策图：用于矩阵表示的高效数据结构》, 系统设计形式化方法 10（1997）149–169。[3] D. Angluin, 《查询与概念学习》, 机器学习 2（1988）319–342。[4] A. Nakamura, 《一种用于有序二元决策图的高效查询学习算法》, 信息与计算 201（2005）178–198。[5] M. J. Kearns、U. V. Vazirani, 《计算学习理论导论》, 麻省理工学院出版社, 剑桥，马萨诸塞州，美国，1994年。

[6] R. Gavald`a、D. Guijarro，《学习有序二元决策图》，载于：算法学习理论，1995年，第228–238页。

[7] H. Mizumoto, S. Todoroki, Diptarama, R. Yoshinaka, A. Shinohara，《零抑制二元决策图的高效查询学习算法》，载于：第28届国际算法学习理论会议论文集，第76卷，2017年，第360–371页。

[8] D. Dua, E. Karra Taniskidou，UCI机器学习资料库，2017年。网址：`http://archive.ics.uci.edu/ml`。

[9] A. Nakamura, K. Sakurada，《减少决策森林中不同分支条件数量的算法》，载于：数据库中的机器学习与知识发现——欧洲会议 ECML PKDD 2019，德国维尔茨堡，2019年9月16日至20日，论文集，第一部分，2019年，第578–589页。

附录A. 定理1的证明

下面给出 OMTBDD 的引理1、引理2和定理1的证明，这些证明是 [4] 中 OBDD 的引理3、引理4和定理5证明的扩展。为确保内容自足，我们写出了完整的详细证明。

引理1. 对于约简 *OMTBDD* D，假设 *OMTBDDAS* S 和节点分类树 T_i ($i = 1, ..., m$) 满足条件*CN*、*CT*和*CE*。那么， $\mathcal{D}(S) = D$ ，当且仅当 $V(S)$ 的基数恰好等于D中的节点数。

证明。令 $N(\mathcal{D}(S))$ 和 $N(D)$ 分别为 $\mathcal{D}(S)$ 和 D 中节点的集合。定义映射 M: $N(\mathcal{D}(S)) \to N(D)$，将 $\mathcal{D}(S)$ 中的节点 $\boldsymbol{v} \in V(S)$ 映射到 D 中访问字符串为 $\boldsymbol{v}$ 的节点。为了证明 $\mathcal{D}(S) = D$，我们证明 $\langle 1 \rangle$ M 定义良好、一一对应且为满射，$\langle 2 \rangle$ M 保留节点标签，并且 $\langle 3 \rangle$ M 保留边关系和标签。$\langle 1 \rangle$ 很容易证明；根据 CN (1)，映射 M 定义良好；根据 CN (3)，它是一一映射；又根据假设，$|V(S)|$ 等于 D 中的节点数，因此它是满射。$\mathcal{D}(D)$ 和 D 都是相同变量排序下的 OMTBDD，且节点映射到访问字符串相同的节点，因此内部节点标签相同。CN (2) 保证汇点标签相同。因此，$\langle 2 \rangle$成立。

证明 $\langle 3\rangle$ 如下。由于 $\langle 1\rangle$ 和 $\langle 2\rangle$ 成立， $\mathcal{D}(S)$ 和 D 的非汇点数相同，因此边数也相同。所以，只需证明：对于任意 $\boldsymbol{v}_1$， $\boldsymbol{v}_2 \in V(S)$，如果 D 中访问字符串为 $\boldsymbol{v}_1$ 和 $\boldsymbol{v}_2$ 的节点之间存在一条带 b 标签的边，那么 $(\boldsymbol{v}_1, \boldsymbol{v}_2) \in E(S)$，且 $l(\boldsymbol{v}_1, \boldsymbol{v}_2)$ 的首位比特为 S 中的 b。

假设在 D 中，访问字符串为 $\boldsymbol{v}_1$ 和 $\boldsymbol{v}_2$ 的节点之间存在一条标记为 b 的边，其中 $\boldsymbol{v}_1, \boldsymbol{v}_2 \in V(S)$。令 $(\boldsymbol{v}_1, \boldsymbol{v})$ 为 S 中从节点 $\boldsymbol{v}_1$ 出发、标记为首位比特为 b 的字符串的边。假设 $|\boldsymbol{v}| < |\boldsymbol{v}_2|$。由于 $\boldsymbol{v}_1 \cdot l(\boldsymbol{v}_1, \boldsymbol{v}) \notin$ 节点(D)，根据 CT (2)， $T_{|v|}(\boldsymbol{v}_1 \cdot l(\boldsymbol{v}_1, \boldsymbol{v})) = \mu$，因此根据 CE (1)，$(\boldsymbol{v}_1, \boldsymbol{v}) \notin E(S)$，这与假设矛盾。因此，$|\boldsymbol{v}| \geq |\boldsymbol{v}_2|$。假设 $|\boldsymbol{v}| > |\boldsymbol{v}_2|$。由于 $\boldsymbol{v}_1 \cdot \text{pre}(l(\boldsymbol{v}_1, \boldsymbol{v})\ |\boldsymbol{v}_2| - |\boldsymbol{v}_1|) \overset{D}{=} \boldsymbol{v}_2$，根据 CT (1) 和 P1，$T_{|\boldsymbol{v}_2|}(\boldsymbol{v}_1 \cdot \text{pre}(l(\boldsymbol{v}_1, \boldsymbol{v})\ |\boldsymbol{v}_2| - |\boldsymbol{v}_1|)) = \boldsymbol{v}_2$，这与 CE (2) 矛盾。因此，$|\boldsymbol{v}| = |\boldsymbol{v}_2|$。由于 $\boldsymbol{v}_1 \cdot l(\boldsymbol{v}_1, \boldsymbol{v}) \overset{D}{=} \boldsymbol{v}_2$，根据 CT (1) 和 P1，$T_{|\boldsymbol{v}_2|}(\boldsymbol{v}_1 \cdot l(\boldsymbol{v}_1, \boldsymbol{v})) = \boldsymbol{v}_2$。另一方面，根据 CE (1)，$T_{|\boldsymbol{v}_2|}(\boldsymbol{v}_1 \cdot l(\boldsymbol{v}_1, \boldsymbol{v})) = \boldsymbol{v}$。因此，$\boldsymbol{v} = \boldsymbol{v}_2$。这意味着 $(\boldsymbol{v}_1, \boldsymbol{v}_2) \in E(S)$，并且 S 中 $l(\boldsymbol{v}_1, \boldsymbol{v}_2)$ 的首位比特为 b。因此， $\langle 3\rangle$ 成立。 □

引理 2。对于目标 *OMTBDD* D，假设 *OMTBDDAS* S 和针对 $i = 1$、...、m 的分类树 T_i 满足 ***CN***、***CT*** 和 ***CE***，且 S 至少有两个汇点。令 $\boldsymbol{e}$ 为 D 针对 $\mathcal{D}(S)$ 的一个反例。令 $(S'$、T'_1、...、$T'_m)$表示对 $(S$、T_1、...、T_m、$\boldsymbol{e})$执行 ***Update-Hypothesis*** 算法的输出。那么，$(S'$、T'_1、...、$T'_m)$ 满足 ***CN***、***CT*** 和 ***CE***，并且 $|V(S')| \geq |V(S)| + 1$。

证明。首先请注意，如果 $T_{|\boldsymbol{v}|}(\boldsymbol{v}) \neq \mu$，那么在算法 NodeSplit 和 NewBranchingNode 中对 $T_{|\boldsymbol{v}|}$ 进行任何更新后， $T_{|\boldsymbol{v}|}(\boldsymbol{v}) \neq \mu$ 仍然成立。这是因为，从根到 $T_{|\boldsymbol{v}|}$ 中标记为 μ 的叶节点的路径，除该 μ标记的叶节点本身外，仅包含双测试节点；双测试节点从不更新， μ标记的叶节点从不添加到单测试节点中，并且在这些算法中，任何非 μ标记的叶节点都不会被替换为带有 μ标记叶节点的子树。

一次执行 Update-Hypothesis 算法可能会添加多个节点。首先，我们通过断言 1 证明，添加第一个节点及其伴随的节点分类树更新后，满足 CN 和 CT 这两个条件的性质仍然成立。然后，我们也证明，之后每次添加一个节点及其伴随的节点分类树更新，都能保持满足 CN 和 CT 的性质。最后，我们证明，所有节点添加及其伴随的节点分类树更新完成后，必定满足 CE。

断言 1. 设 $(S, T_1, \ldots, T_m)$ 为在算法 *Update-Hypothesis* 中首次添加节点 $\boldsymbol{v}$ 并相应更新$T_{|\boldsymbol{v}|}$之后的 *OMTBDDAS* 和节点分类树。该算法针对$(S, T_1, ..., T_m, \boldsymbol{e})$运行。则 *CN* 和 *CT* 由$(S, T_1, \ldots, T_m)$满足。

断言 1 的证明。在 NodeSplit 中，第一个新节点 $\boldsymbol{v} = \boldsymbol{p}_i l(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})$ 在第 2 行被添加到 S。对于这些 $\boldsymbol{v}$, $\boldsymbol{v} \in$ 节点，D 由 CT (2) 推出，因为 CE (1) 表明 $T_{|\boldsymbol{v}|}(\boldsymbol{v}) = T_{|\boldsymbol{v}|}(\boldsymbol{p}_i l(\boldsymbol{p}_i, \boldsymbol{p}_{i+1})) = \boldsymbol{p}_{i+1} \neq \mu$ 成立，因此 CN (1) 对 成立。节点 $\boldsymbol{v}$ 不可能是汇点；否则，由 CE (1) 可得 $D(\boldsymbol{v}) = D(\boldsymbol{p}_{k-1} l(\boldsymbol{p}_{k-1}, \boldsymbol{p}_k)) = D(\boldsymbol{p}_k) \neq D(\boldsymbol{e})$，这与执行 NodeSplit 这一事实矛盾。因此，$V_m(S) = V_m(S)$，从而根据关于 S 的假设，CN (2) 对 S 成立。对于任意 $\boldsymbol{v}' \in V(S)$，我们可以如下证明 $\boldsymbol{v} \neq \boldsymbol{v}'$。若 $|\boldsymbol{v}| \neq |\boldsymbol{v}'|$，则结论显然成立；因此假设 $\boldsymbol{v}' \in V_{|\boldsymbol{v}|}(S)$。如果 $\boldsymbol{v}' \neq \boldsymbol{p}_{i+1}$，则因为 $T_{|\boldsymbol{v}|}(\boldsymbol{v}) = \boldsymbol{p}_{i+1} \neq \boldsymbol{v}'$，所以对于 $\boldsymbol{t} = \boldsymbol{r}$ 或 $\boldsymbol{t} = \dot{\boldsymbol{r}}$，$D(\boldsymbol{vt}) \neq D(\boldsymbol{v}'\boldsymbol{t})$ 成立，其中 $\boldsymbol{r}$ 是 $T_{|\boldsymbol{v}|}$ 中分别标记为 $\boldsymbol{v}'$ 和 $\boldsymbol{p}_{i+1}$ 的叶节点的最近公共祖先的标签。对于 $\boldsymbol{v}' = \boldsymbol{p}_{i+1}$，$D(\boldsymbol{v}\boldsymbol{e}_{i+1}) \neq D(\boldsymbol{p}_{i+1}\boldsymbol{e}_{i+1})$ D 成立，因此 $\boldsymbol{v} \neq \boldsymbol{p}_{i+1}$ 成立。由此，CN (3) 对 S 也成立。因此，CN 由 $(S, T_1, \ldots, T_m)$ 满足。至于节点分类树，只有标记为 $\boldsymbol{p}_{i+1}$ 的叶节点会被替换为标记为 $\boldsymbol{e}_{i+1}$ 的单测试节点，其子节点分别标记为 $\boldsymbol{v}$ 和 $\boldsymbol{p}_{i+1}$。因此，对于所有 $\boldsymbol{u} \in V_{|\boldsymbol{v}|}(S)$，$T_{|\boldsymbol{v}|}(\boldsymbol{u}) = \boldsymbol{u}$ 成立。由于 $\{\boldsymbol{a} \mid T_i(\boldsymbol{a}) = \mu\} = \{\boldsymbol{a} \mid T_i(\boldsymbol{a}) = \mu\}$，CT (2) 对 $T_i\,(i = 1, 2, \ldots, m)$ 仍然成立。因此，CT 也由 $(S, T_1, \ldots, T_m)$ 满足。

在算法新分支节点中，第一个新节点 $\boldsymbol{v}$ 会在第5行或第 13 行被添加到 S。在这两种情况下，$\boldsymbol{v} \in$ 节点(D)，因为对于 $\boldsymbol{r} = \text{suf}(\boldsymbol{e}, m - |\boldsymbol{v}|)$，$D(\boldsymbol{vr}) \neq D(\boldsymbol{v}\dot{\boldsymbol{r}})$成立；这由第 2 行中对 j at的选择所保证，因此 CN (1) 对于 S成立。由于 $\boldsymbol{v}$ 不能是汇点，因为 $j \geq 1$， CN (2) 对于 S仍然成立。根据 CE (2)，$T_{|\boldsymbol{v}|}(\boldsymbol{v}) = \mu$ 成立，因此对于任何 $\boldsymbol{v}' \in V(S)$，$\boldsymbol{v} \neq \boldsymbol{v}'$，因为对于标记为 μ 和 $\boldsymbol{v}'$的节点的最近公共祖先的标签$\boldsymbol{r}$，$D(\boldsymbol{vr}) \neq D(\boldsymbol{v}'\boldsymbol{r})$或 $D(\boldsymbol{v}\dot{\boldsymbol{r}}) \neq D(\boldsymbol{v}'\dot{\boldsymbol{r}})$成立。因此，CN (3) 对于 S成立。显然，$T_{|\boldsymbol{v}|}(\boldsymbol{v}) = \boldsymbol{v}$ 成立，因为 CE (2) 保证了 $T_{|\boldsymbol{v}|}(\boldsymbol{v}) = \mu$，并且 $T_{|\boldsymbol{v}|}$ 中标记为 μ 的叶节点被替换为一个标记为 $\boldsymbol{r}$ 的双测试节点，该节点有一条标记为 $(D(\boldsymbol{vr})\ D(\boldsymbol{v}\dot{\boldsymbol{r}}))$ 的出边，通向标记为$\boldsymbol{v}$的叶节点。由于未对 $T_{|\boldsymbol{v}|}$进行其他更新，因此对于 $\boldsymbol{v}' \in V_{|\boldsymbol{v}|}(S)$，$T_{|\boldsymbol{v}|}(\boldsymbol{v}') = T_{|\boldsymbol{v}|}(\boldsymbol{v}') = \boldsymbol{v}'$ 成立。因此，CT (1) 对于 T_1，T_2，...，T_m成立。对于除某些满足 $\mu = T_{|\boldsymbol{v}|}(\boldsymbol{a}) \neq T_{|\boldsymbol{v}|}(\boldsymbol{a}) = \boldsymbol{v}$的 $\boldsymbol{a} \in \{0,1\}^{|\boldsymbol{v}|}$之外的所有 $\boldsymbol{a} \in \{0,1\}^i$，$i = 1$，...，$m$，$T_{|\boldsymbol{a}|}(\boldsymbol{a}) = T(\boldsymbol{a})$成立；这些 $\boldsymbol{a} \in \{0,1\}^{|\boldsymbol{v}|}$属于节点$(D)$，因为 $D(\boldsymbol{ar}) \neq D(\boldsymbol{a}\dot{\boldsymbol{r}})$

对于 $T_{|\boldsymbol{v}|}$中从根到标记为 $\boldsymbol{v}$ 的˜ ˜叶节点的路径上的双测试节点的标签 $\boldsymbol{r}$ 成立。因此，CT (2) 对于 $T_{|\boldsymbol{v}|}$仍然成立。因此，CT 对于 (S，T_1，...，T_m)成立。 □

断言 2。假设当前 *OMTBD-*˜ ˜ ˜*DAS* 以及算法 *Update-Hypothesis*˜ ˜ ˜中的分类树 (S、T_1、.....、T_m) 已经至少完成了一次节点添加及其相应的节点分类树更新。另假设 (S、T_1、.....、T_m) 满足 *CN* 和 *CT*。˜ ˜ ˜令 (S'、T'_1、.....、T'_m) 表示完成一次节点添加及其相应的节点分类树更新后的 *OMTBDDAS* 和节点分类树。˜ ˜ ˜则 (S'、T'_1、.....、T'_m)满足 *CN* 和 *CT*。

断言 2 的证明。考虑在 NodeSplit 的第 12 行和第 13 行执行的节点添加及其相应的节点分类树更新。添加的节点 $\boldsymbol{v}'$ 是 $\boldsymbol{v}_1 l(\boldsymbol{v}_1, \boldsymbol{p}_{i+1})$，其中 $\boldsymbol{v}_1 \in V(S)$。于是，根据 CT (2)，$\boldsymbol{v}' \in$ 节点(D)，因为 $T(\boldsymbol{v}') = T(\boldsymbol{v}_1 l(\boldsymbol{v}_1, \boldsymbol{p}_{i+1})) = \boldsymbol{p}_{i+1} \neq \mu$，依据 CE (1)。因此，CN (1)˜ ˜对 S'成立。CN (2) 对 S' 成立，因为节点 $\boldsymbol{v}'$ 不可能是汇点，因为D ˙$|\boldsymbol{v}'| = |\boldsymbol{v}|$，且 $\boldsymbol{v}$ 不是汇点。此外，对于任意 $\boldsymbol{u} \in V(S)$˜，$\boldsymbol{v}' \neq \boldsymbol{u}$ 成立，因为 $T_{|\boldsymbol{v}'|}(\boldsymbol{v}') = \boldsymbol{e}_{i+1}$（从标记为 $\boldsymbol{e}_{i+1}$ 的单测试节点没有发出任何带 $D(\boldsymbol{v}'\boldsymbol{e}_{i+1})$ 标签的边），这意味着，对于叶节点的任意标签 $\boldsymbol{u}(\neq \mu)$，$\boldsymbol{u}$-标记叶节点与 $\boldsymbol{e}_{i+1}$-标记单测试节点的最近公共祖先的标签 $\boldsymbol{r}$（或 $\dot{\boldsymbol{r}}$）满足 $D(\boldsymbol{v}'\boldsymbol{r}) \neq D(\boldsymbol{u}\boldsymbol{r})$（或 $D(\boldsymbol{v}'\dot{\boldsymbol{r}}) \neq D(\boldsymbol{u}\dot{\boldsymbol{r}})$）。˜ ˜ ˜因此，CN (3) 对 S'成立。CT (1) 对 T'_1、.....、T'_m 成立，因为 $T_{|\boldsymbol{v}'|}$ 已被更新，以便通过添加一个标记为 $\boldsymbol{v}'$ 的叶节点以及一条从标记为 $\boldsymbol{e}_{i+1}$ 的单测试节点指向该叶节点、标签为 $D(\boldsymbol{v}')$ 的边来满足 $T'_{|\boldsymbol{v}'|}(\boldsymbol{v}') = \boldsymbol{v}'$。由于 ˜ ˜ ˙$\{\boldsymbol{a} \mid T_i(\boldsymbol{a}) = \mu\} = \{\boldsymbol{a} \mid T'_i(\boldsymbol{a}) = \mu\}$，CT (2) 对 $T'_i(i = 1$、.....、$m)$ 仍然成立。

我们来考虑在 NodeSplit 的第 5 行和第 14 行，以及 NewBranchingNode 的第 14 行执行的其他节点添加和分类树更新；这些操作可能在算法 AddEdge 中重复执行。在˜AddEdge 中，第 10 行或第 12 行会向 S 添加一个新节点 $\boldsymbol{u}'$。节点 $\boldsymbol{u}'$ 属于˜nodes(D)，因为 $T_{|\boldsymbol{u}'|}(\boldsymbol{u}') = \boldsymbol{u} \neq \mu$（不存在从标记为 $\boldsymbol{u}$ 的单测试节点出发、带有标签 $D(\boldsymbol{u}'\boldsymbol{u})$ 的边），所以 $|\boldsymbol{u}'| = m$ 或 $D(\boldsymbol{u}'\boldsymbol{r}) \neq D(\boldsymbol{u}'\dot{\boldsymbol{r}})$，其中 $\boldsymbol{r}$ 是从根节点到标记为 $\boldsymbol{u}$ 的单测试˜节点的路径上某个双测试节点的标签。因此，CN (1) 对 S'成立。当 $\boldsymbol{u}'$ 是一个汇点时，会更新 S，使其满足 $\mathcal{D}(S')(\boldsymbol{u}') = D(\boldsymbol{u}')$（第 10 行），因为节点 $\boldsymbol{u}'$ 标记为 $D(\boldsymbol{u}')$，并且保证 $S(\boldsymbol{v}) = \boldsymbol{v}$ 对满足˙$\boldsymbol{v}$pre($\boldsymbol{t}$, $|\boldsymbol{u}'| - |\boldsymbol{v}|$) $= \boldsymbol{u}'$的 $\boldsymbol{v}$ 成立。因此，S'满足 CN (2)。此外，

D ~ ~$\boldsymbol{u}' \neq \boldsymbol{v}'$ 对任意 $\boldsymbol{v}' \in V(S)$ 均成立，因为 $T_{|\boldsymbol{u}'|}(\boldsymbol{u}') = \boldsymbol{u}$，且 $\boldsymbol{u}$ 是单测试内部节点的标签；这意味着，对于叶节点的任意标签 $\boldsymbol{v}'(\neq \mu)$，$\boldsymbol{v}'$-标记节点与标记为 $\boldsymbol{u}$ 的节点的最近公共祖先的标签 $\boldsymbol{r}$（或 $\dot{\boldsymbol{r}}$）满足 $D(\boldsymbol{u}'\boldsymbol{r}) \neq D(\boldsymbol{v}'\boldsymbol{r})$（或 $D(\boldsymbol{u}'\dot{\boldsymbol{r}}) \neq D(\boldsymbol{v}'\dot{\boldsymbol{r}})$）。因此，$S'$ 满足 CN (3)。通过添加一个标记为 $\boldsymbol{u}'$ 的叶节点，以及一条从标记为 $\boldsymbol{u}$ 的单~ ~测试节点指向该叶节点、标签为 $D(\boldsymbol{u}'\boldsymbol{u})$ 的边，更新 $T_{|\boldsymbol{u}'|}$，使其满足 $T'_{|\boldsymbol{u}'|}(\boldsymbol{u}') = \boldsymbol{u}'$。因此，CT (1) 对 $T'_1, \ldots, T'_m$ 成立。由于 ~ ~ ~ ~$\{\boldsymbol{a} \mid T_i(\boldsymbol{a}) = \mu\} = \{\boldsymbol{a} \mid T'_i(\boldsymbol{a}) = \mu\}$，CT (2) 对 $T'_1, \ldots, T'_m$ 仍然成立。 □

我们证明 CE 对 (S'，T'_1，……，T'_m) 成立。

首先，考虑 NodeSplit 情况。在 NodeSplit 中，对于任意 T_i (i =1, . . . , m)，从根节点到叶节点的路径上，标记为 μ 的叶节点和双测试节点不会改变，这意味着 $T'_i(\boldsymbol{u}) = \mu$ 当且仅当 $T_i(\boldsymbol{u}) = \mu$ 对所有 $\boldsymbol{u} \in \{0,1\}^i, i = 1, \ldots, m$ 均成立。因此，CE (2) 对 $(\boldsymbol{u}, \boldsymbol{v}) \in E(S') \cap E(S)$ 成立。令 $\boldsymbol{v}$ 为在第 2 行添加到 S 的新节点。那么，NodeSplit 中发生变化的 T_i 是 $T_{|\boldsymbol{v}|}$ 和 T_j，其中某个 $j > |\boldsymbol{v}|$ 是通过 AddEdge 添加的。在 $T_{|\boldsymbol{v}|}$ 中，标记为 $\boldsymbol{p}_{i+1}$ 的叶节点在第 15 行被替换为 $T^{e_{i+1}}_{|\boldsymbol{v}|}$。因此，必须删除边 $(\boldsymbol{v}', \boldsymbol{p}_{i+1})$，其中 $T_{|\boldsymbol{v}|}(\boldsymbol{v}' l(\boldsymbol{v}', \boldsymbol{p}_{i+1})) = \boldsymbol{p}_{i+1}$，以及 $T'_{|\boldsymbol{v}|}(\boldsymbol{v}' l(\boldsymbol{v}', \boldsymbol{p}_{i+1})) = \boldsymbol{v} \neq \boldsymbol{p}_{i+1}$，并添加新边 $(\boldsymbol{v}', \boldsymbol{v})$。所有此类边删除和添加均在第 2、10 和 12 行完成。因此，所有不满足 CE (1) 的边 $(\boldsymbol{v}', \boldsymbol{p}_{i+1})$ 都会被删除，而所有添加的边 $(\boldsymbol{v}', \boldsymbol{v})$ 都满足 CE (1)。由于添加边 $(\boldsymbol{v}', \boldsymbol{v})$ 对应的字符串 $l(\boldsymbol{v}', \boldsymbol{v})$ 与删除边 $(\boldsymbol{v}', \boldsymbol{p}_{i+1})$ 对应的字符串 $l(\boldsymbol{v}', \boldsymbol{p}_{i+1})$ 相同，并且 $T'_i = T_i$ 对于 $i < |\boldsymbol{v}|$，CE (2) 对新边 $(\boldsymbol{v}', \boldsymbol{v})$ 仍然成立。其他新边 $(\boldsymbol{v}', \boldsymbol{u}')$ 通过 AddEdge($\boldsymbol{v}', \boldsymbol{t}$) 添加，这些边的构造保证满足 CE (1) 和 CE (2)。因此，CE 对 $(S', T'_1, \ldots, T'_m)$ 成立。

接下来，考虑 NewBranchingNode 的情况。令 $\boldsymbol{v}$ 表示在第5行添加到 S 的新节点。然后，通过 AddEdge，NewBranchingNode 中发生变化的 T_j 是$T_{|\boldsymbol{v}|}$ 和 T_j，其中 $j > |\boldsymbol{v}|$ 为某个值。在 $T_{|\boldsymbol{v}|}$中，第11行用 $T^{r}_{|\boldsymbol{v}|}$ 替换标签为 μ的叶节点。因此，必须删除满足 $|\boldsymbol{u}| < |\boldsymbol{v}| < |\boldsymbol{v}'|$、$T_{|\boldsymbol{v}|}(\boldsymbol{u}\text{pre}(l(\boldsymbol{u}, \boldsymbol{v}')\ |\boldsymbol{v}| - |\boldsymbol{u}|)) = \mu$ 和 $T'_{|\boldsymbol{v}|}(\boldsymbol{u}\text{pre}(l(\boldsymbol{u}, \boldsymbol{v}')\ |\boldsymbol{v}| - |\boldsymbol{u}|)) = \boldsymbol{v}$ 的边 $(\boldsymbol{u}, \boldsymbol{v}')$，并添加新边 $(\boldsymbol{u}, \boldsymbol{v})$。所有此类边删除和添加操作都在第5行和第10行完成。因此，所有不满足 CE (2) 的边 $(\boldsymbol{u}, \boldsymbol{v}')$ 都会被删除，且所有添加的边 $(\boldsymbol{u}, \boldsymbol{v})$ 都满足 CE (1)。由于添加边 $(\boldsymbol{u}, \boldsymbol{v})$ 对应的字符串 $l(\boldsymbol{u}, \boldsymbol{v})$ 与被删除边 $(\boldsymbol{u}, \boldsymbol{v}')$ 对应的字符串 $\text{pre}(l(\boldsymbol{u}, \boldsymbol{v}')\ |\boldsymbol{v}| - |\boldsymbol{u}|)$ 相同，并且对于 $j < |\boldsymbol{v}|$，$T'_j = T_j$，因此添加的边 $(\boldsymbol{u}, \boldsymbol{v})$ 仍满足 CE (2)。边 $(\boldsymbol{v}, \boldsymbol{p}_{i+1})$

也在第5行添加。由于 S' 中的 $\boldsymbol{v}l(\boldsymbol{v},\boldsymbol{p}_{i+1})$ 等于S 中的 $\boldsymbol{p}_i l(\boldsymbol{p}_i,\boldsymbol{p}_{i+1})$，并且在 NewBranchingNode 中，从根节点到 $T_{|\boldsymbol{p}_{i+1}|}$中标签为 $\boldsymbol{p}_{i+1}$的叶节点的路径上的节点不会改变，因此对于 S'， $T'_{|\boldsymbol{p}_{i+1}|}(\boldsymbol{v}l(\boldsymbol{v},\boldsymbol{p}_{i+1}))$ 等于对于 S的 $T_{|\boldsymbol{p}_{i+1}|}(\boldsymbol{p}_i l(\boldsymbol{p}_i,\boldsymbol{p}_{i+1}))$，所以根据 CE (1)，$(S,T_1,\ldots,T_m)$ 满足 $T'_{|\boldsymbol{p}_{i+1}|}(\boldsymbol{v}l(\boldsymbol{v},\boldsymbol{p}_{i+1})) = \boldsymbol{p}_{i+1}$。由于添加边 $(\boldsymbol{v},\boldsymbol{p}_{i+1})$ 对应的字符串 $l(\boldsymbol{v},\boldsymbol{p}_{i+1})$ 与被删除边 $(\boldsymbol{p}_i,\boldsymbol{p}_{i+1})$ 对应的字符串 $\text{suf}(l(\boldsymbol{p}_i,\boldsymbol{p}_{i+1})$ $|\boldsymbol{p}_{i+1}| - |\boldsymbol{v}|)$ 相同，并且对于 $j > |v|$， T_j 中从根节点到标签为 μ的叶节点的路径上没有节点发生变化，因此添加的边 $(\boldsymbol{v},\boldsymbol{p}_{i+1})$ 仍满足 CE (2)。第14行还通过 AddEdge$(\boldsymbol{v},\boldsymbol{r})$ 添加了其他新边 $(\boldsymbol{v}',\boldsymbol{u}')$，其中 $|\boldsymbol{v}| \le |\boldsymbol{v}'| < |\boldsymbol{u}'|$ 为某个值；这些边的构造使其满足 CE (1) 和 CE (2)。因此，$(S',T'_1,\ldots,T'_m)$ 满足 CE。

□

定理*1*的证明。我们假设 D 至少有两个汇点，因为只有一个汇点的情况是显而易见的。首先，我们证明 QLearn-OMTBDD 至多通过 n次等价查询输出D。显然， S^0和 T_1^0，……， T_m^0满足条件 CN、CT 和 CE。根据引理 2，算法所维护的 OMTBDDAS S 中的非虚拟节点数，在每次执行 Update-Hypothesis 算法时至少增加 1；该算法会更新 S和节点分类树 T_1，……， T_m，使其满足条件 CN、CT 和 CE。因此，执行 Update-Hypothesis 算法至多 $n-3$次后， $|V(S)|$ 达到 D 中的节点数。接着，引理1保证 $\mathcal{D}(S) = D$。因此，QLearn-OMTBDD 针对 $n' \le n$ 发出的第 n'次等价查询得到“是”的答复，因为在首次执行 Update-Hypothesis 之前会提出三次等价查询，而每次执行 Update-Hypothesis 之后至多提出一次等价查询。

接下来，我们考虑成员查询的数量。为了构造 S^0和 $T_1^0,\ldots,T_m^0$，算法使用 $\lceil \log_2 m \rceil$ 次成员查询。让我们考虑每向 S添加一个节点时，执行一次更新假设期间会提出多少次成员查询。

执行更新假设时包含 NodeSplit 的情况

在更新假设的第 2 行，最多会提出 $\lceil \log_2 m \rceil$ 次成员查询。在 NodeSplit 的第 7 行，最多会提出 n 次成员查询，因为对于 S 中的每个节点，最多提出一次成员查询。在 NodeSplit 的每次第 5 行执行中，如果 AddEdge 中没有添加节点，则最多会提出 $4n$ 次成员查询，因为对于 $T_1,\ldots,T_m$ 中的每个内部节点，最多提出两次成员查询，且内部节点数最多为 n。在这种情况下，如果第 12 行和 AddEdge 中没有节点添加，则该算法总共最多提出 $\lceil \log_2 m \rceil + 5n$ 次成员查询。在

第 12 行每添加一个节点，最多会额外提出 $4n$ 次成员查询；在 AddEdge 中每添加一个节点，最多会额外提出 $2n$ 次成员查询。因此，对于 S 的每次节点添加，最多会提出 $\lceil \log_2 m \rceil + 5n$ 次成员查询。

Update-Hypothesis 中执行新分支节点的情况

在 Update-Hypothesis 的第 2 行和新分支节点的第 2 行中，通过使用二分搜索，至多进行 $2\lceil \log_2 m \rceil$ 次成员查询。在新分支节点的第 9 行中，至多进行 $4n$ 次成员查询，因为对于 S中的每条边，至多进行两次成员查询。在新分支节点的第 14 行中，若由于上述原因 AddEdge 中未添加节点，则至多进行 $2n$ 次成员查询。在这种情况下，若 AddEdge 中没有节点添加，算法总共至多进行 $2\lceil \log_2 m \rceil + 6n$ 次成员查询。即使存在节点添加，对于向 S添加一个节点，每次节点添加至多会额外进行 $2n$ 次成员查询。因此，对于向 S添加一个节点，总共至多进行 $2\lceil \log_2 m \rceil + 6n$ 次成员查询。

因此，QLearn-OMTBDD 最多会提出 $2n(\lceil \log_2 m \rceil + 3n)$ 个成员查询。□

附录B. 随机 OMTBDD 生成过程

在我们的合成数据集实验中，OMTBDD 由算法 6 生成。

算法 6 OMTBDD 生成算法

输入： m：变量数量，n：节点数，K：汇点数

输出： D：具有 n 个节点和（最多）K 个汇点的约简有序多终端二元决策图，适用于 m 排序 $x_1 < x_2 < \cdots < x_m$ 的变量。

$n', n'' \leftarrow n$

重复

  $n' \leftarrow n' +$(n − n″)

  从 $\{x_1, \ldots, x_m\}$ 中随机选择 $n' - K$ 个变量并排序，使其成为 $x_{i_1}, x_{i_2}, \ldots, x_{i_{n'-K}}$ 以满足 $i_1 \le i_2 \le \cdots \le i_{n'-K}$。创建节点 v_j 为 $j = 1, \ldots, n' - K$ 创建标签为 x_{i_j} 的节点，并为 $j = n' - K + 1, \ldots, n'$ 创建汇点 v_j。

  对于 $j = 1$ **到** $n' - K$，**执行**

    如果 $j > 1$ 且 v_j 没有入边 **则**

      删除节点 v_j。继续处理 FOR循环中的下一个 j 。

    随机将 ℓ 设为 0 或 1。

    对于 $k = \ell$ **到** $(\ell + 1)\%2$，**执行**

      如果 $\exists v_j$ 使得 n var(j) $\le h <$n var(n var(j)) 且 v_h 没有入边 边 {n var(j) $=$ min($\{h \mid i_h > i_j\} \cup \{n' - K + 1\}$)} **则**

        Add 边 (v_j, v_h) 用于 h 从中随机选择的 {n var(j)…, n var(n var(j))} 作为带有 k标签的出边 v_j的。

      else

        Add 边 (v_j, v_h) 用于 h 从中随机选择的 {n var(j)…, n} 作为 v_j 的 k 标签出边。选择 h 如果 $k = (\ell + 1)\%2$，并且与 $k = \ell$ 相同的边为所选。

  将汇点 v_j($j = n' - K + 1, \ldots, n'$) 的标签设为 0, 1, … $K - 2$ 或 $K - 1$ 随机设置，使每个汇点标签都不同。

  将当前 OMTBDD 转换为唯一的 OMTBDD D，使其处于约简形式。将 n'' 设为 D 中的节点数。

直到 $n'' = n$

输出 D。