Using Binary Decision Diagrams to Enumerate Inductive Logic Programming Solutions

Hikaru Shindo[1], Masaaki Nishino[2], and Akihiro Yamamoto[1]

[1] Graduate School of Informatics, Kyoto University, Kyoto, Japan
hikarushindo@iip.ist.i.kyoto-u.ac.jp
[2] NTT Communication Science Laboratories, NTT Corporation, Kyoto, Japan

Abstract. This paper proposes a method for efficiently enumerating all solutions of a given ILP problem. Inductive Logic Programming (ILP) is a machine learning technique that assumes that all data, background knowledge, and hypotheses can be represented by first-order logic. The solution of an ILP problem is called a hypothesis. Most ILP studies propose methods that find only one hypothesis per problem. In contrast, the proposed method uses Binary Decision Diagrams (BDDs) to enumerate all hypotheses of a given ILP problem. A BDD yields a very compact graph representation of a Boolean function. Once all hypotheses are enumerated, the user can select the preferred hypothesis or compare the hypotheses using an evaluation function. We propose an efficient recursive algorithm for constructing a BDD that represents the set of all hypotheses. In addition, we propose an efficient method to get the best hypothesis given an evaluation function. We empirically show the practicality of our approach on ILP problems using real data.

Keywords: Inductive Logic Programming · Binary Decision Diagram.

1 Introduction

This paper proposes a method that can efficiently enumerate all solutions of a given ILP problem. Inductive Logic Programming (ILP) is a machine learning technique that assuming that each datum, background knowledge, and hypothesis can be represented by first-order logic. Our work is based on the foundation of ILP established by Plotkin [17] and Shapiro [21]: inductive inference with clausal logic. The goal of an ILP system is to find a hypothesis that explains all positive examples while admitting no negative examples. A lot of methods for finding hypotheses have been proposed. Inverse entailment [13] and saturation [20] are well known methods of hypothesis discovery. Unfortunately, there are restrictions on the hypotheses that can be generated by these methods as they are designed to output only one hypothesis per problem. This restriction may prevent important hypotheses from being generated, i.e. discovered. To avoid this problem, we propose a method that can enumerate all hypotheses.

As most ILP problems have an infinite number of hypotheses in the absence of any restriction, we need a new algorithm that enumerates all hypotheses in

a bounded hypothesis space, i.e., every hypothesis is a subset of a finite set of clauses. Even with this restriction, it is implausible to naively enumerate all hypotheses because there are 2^n candidate hypotheses where n is the number of clauses in the hypothesis space. Our proposal is an enumeration method that exploits Binary Decision Diagrams (BDDs) [1]. A BDD is a very compact graph representation of a Boolean function, and allows several operations to be performed efficiently. The merits of enumerating hypotheses are following:

- **Preventing hypothesis omission** The importance of a hypothesis depends on the case, so algorithms that give only one hypothesis may not return the best hypothesis. By using a BDD to enumerate all hypotheses, the user is assured of getting the important hypothesis for the desired case as the BDD ensures that it will be enumerated.
- **Hypothesis selection** Users can select a hypothesis or compare some hypotheses using an evaluation function. Moreover, users can select the top-k best hypotheses. If there are multiple evaluation metrics for evaluating hypothesis importance, then finding just one hypothesis may insufficient. We show that this selection can be executed efficiently with BDDs.
- **Online-learning** BDDs support efficient binary operations. By using this feature, we can efficiently perform online leaning, i.e., updating the current set of hypothesis when new examples are added.

We introduce propositional variables that represent if a clause is contained in a hypothesis or not. This makes the hypothesis enumeration problem equivalent to the problem of identifying an n-ary Boolean function. We show an efficient way of constructing a BDD for Boolean function representation. As an application of the constructed BDD, we show how to get the best hypothesis in terms of minimizing an evaluation function. As an example, this paper introduces an evaluation function based on Minimum Description Length (MDL) [19]. We can find the best hypothesis in time linear to the number of nodes by dynamic programming. We apply these methods to real data sets and show that the resulting BDDs can compactly represent hypotheses.

2 Preliminaries

2.1 Inductive Logic Programming (ILP)

In clausal logic, formulae are constructed of five symbols [16]: constants, variables, function symbols, predicate symbols, and connectives. A *term* is a constant, a variable, or an expression $f(t_1, \ldots, t_n)$ where f is an n-ary function and $t_1, \ldots, t_n$ are terms. If P is an n-ary predicate symbol and $t_1, \ldots, t_n$ are terms, then $P(t_1, \ldots, t_n)$ is a formula called an *atom*. A *ground atom* is an atom that contains no variables. A *literal* is an atom or its negation. A *positive literal* is just an atom. A *negative literal* is the negation of an atom. A *clause* is a finite disjunction ($\vee$) of literals. A *definite clause* is a clause with exactly one positive literal. If $A, B_1, \ldots, B_n$ are atoms, then $A \vee \neg B_1 \vee \ldots \vee \neg B_n$ is a definite clause.

We write definite clauses in the form of the implication $A \leftarrow B_1 \wedge \ldots \wedge B_n$. Atom A in the clause is called the *head*, and the set of negative atoms $\{B_1, \ldots, B_n\}$ in the clause is called the *body*. We denote the constant *true* as $\mathbf{T}$, and the constant $false$ as $\mathbf{F}$.

ILP concerns learning a *theory* from given *examples*, possibly taking *background knowledge* into account. We can distinguish between two kinds of examples: *positive* and *negative*. Usually, we assume that a theory is represented as a finite set, Σ, of definite clauses, and the positive examples, the negative examples, and background knowledge are given respectively as finite sets $\mathcal{E}^+, \mathcal{E}^-, \mathcal{B}$ of ground atoms.

Formally, the ILP problem treated in this paper is defined as follows.

Input Finite set $\mathcal{E}^+$, $\mathcal{E}^-$, and $\mathcal{B}$ of ground atoms,
Output A set of definite clauses Σ such that

$$for\ all\ A \in \mathcal{E}^+\ \Sigma \cup \mathcal{B} \models A\ \ and\ \ for\ all\ A \in \mathcal{E}^-\ \Sigma \cup \mathcal{B} \not\models A. \tag{1}$$

In this paper, we call theory Σ that satisfies equation (1) a *hypothesis*.

Example 1.

$$\mathcal{E}^+ = \{e(0), e(s^2(0))\}, \mathcal{E}^- = \{e(s(0))\}, \mathcal{B} = \{\},$$

where e is a 1-ary predicate symbol that becomes true if its argument is an even number, 0 is a constant symbol interpreted as natural number 0, s is a 1-ary function symbol and $s^n(0)$ is interpreted as natural number n. Examples of correct hypotheses are as follows:

$$\Sigma = \{e(0), e(s^2(0))\},$$
$$\Sigma = \{e(0), e(s^2(x)) \leftarrow e(x)\}.$$

2.2 Binary Decision Diagram (BDD)

A Binary Decision Diagram (BDD) [1] is a directed acyclic graph that represents a Boolean function. Fig. 1 is a BDD that represents Boolean function $(x_0 \wedge x_1) \vee x_2$. The nonterminal node that has no parents is called root. Each nonterminal node is labeled with a variable, and has a low-branch, shown as a dashed line, and a high-branch, shown as a solid line. We denote the node whose index is i as node i. We start at the root and take the low-branch from node j when $x_j = 0$, the high-branch when $x_j = 1$. Each terminal node is labeled 0 or 1. A path from the root to the terminal node labeled 0 corresponds to an assignment for which the function returns false, and a path from the root to the terminal node labeled 1 corresponds to an assignment for which the function returns true. In this paper, the number of nodes means the number of nodes other than terminal nodes.

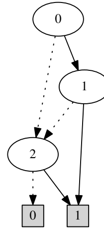

Fig. 1. BDD that represents $(x_0 \wedge x_1) \vee x_2$

A binary decision tree is obtained by applying Shannon decomposition recursively a Boolean function, and a BDD is given by deleting redundant nodes and sharing equivalent subgraphs of a binary decision tree. BDDs have the following features.

- The shape of the minimum BDD is uniquely determined by the order of input variables. If the order of input variables is given, this feature enables the equivalency of two Boolean functions to be checked in constant time if they are represented as BDDs.
- The number of nodes of a BDD that represents an m input Boolean function is $\mathcal{O}(\frac{2^m}{m})$ in the worst case [10]. However, many practical Boolean functions can be represented as a BDD, which is a comparatively small graph.
- Binary operations between BDDs can be executed efficiently [3]. For example, given two BDDs representing logical functions F and G, then the BDD representing $H = F \wedge G$ can be computed in time linear to input BDD sizes.
- Users can count the number of solutions, solve the Boolean optimization problem, and randomly sample a solution in $\mathcal{O}(n)$, where n is the number of the nodes of the BDD (see [9]).

In this paper, we assume that the index of terminal node $\boxed{0}$ is 0, that of terminal node $\boxed{1}$ is 1, and that of the root is $n+1$, where n is the number of nodes of the BDD. We also assume for all nodes i and node j, where $0 \leq i \leq n+1$ and $0 \leq j \leq n+1$,

$$i < j \Rightarrow \text{node } j \text{ is not the child of node } i.$$

3 Enumerating hypotheses by constructing BDDs

3.1 Restrictions

In general, there are an infinite number of hypotheses for an ILP problem if no restrictions are set. Furthermore, the algorithm proposed here does not finish in finite steps if there is a recursive structure in the hypothesis space. Let us introduce some concepts for bounding the hypothesis space.

Definition 1. *(Mutually recursive clauses) Let $\mathcal{H}$ be a hypothesis space that is finite set of definite clauses. If a series of definite clauses $\{C_i \in \mathcal{H}\}_{i=0,\dots,n}$ and substitutions $\theta_1, \dots, \theta_n$ exist, and they are expressed as*

$$\begin{aligned} C_1\theta_1 &= A \leftarrow \ldots \wedge X_1 \wedge \ldots, \\ C_2\theta_2 &= X_1 \leftarrow \ldots \wedge X_2 \wedge \ldots, \\ &\vdots \\ C_n\theta_n &= X_{n-1} \leftarrow \ldots \wedge A \wedge \ldots, \end{aligned}$$

then $C_1, C_2, \dots, C_n$ are ***mutually recursive clauses****.*

Definition 2. *(Variable-bounded [2]) Definite clause $A \leftarrow B_1 \wedge \ldots \wedge B_n$ is* ***variable-bounded*** *if $v(A) \supseteq v(B_i)$ $(i = 1, \dots, n)$, where $v(C)$ is the set of all variables in C. The hypothesis space $\mathcal{H}$ is variable-bounded if all $C \in \mathcal{H}$ are variable-bounded.*

The requirements that hypothesis space $\mathcal{H}$ should hold are as follows:

Requirement 1 The hypothesis space does not contain any mutually recursive clauses.
Requirement 2 The hypothesis space is variable-bounded.

Requirement 1 ensures that we can trace all literals present in the hypothesis space in a finite number of steps. **Requirement 2** ensures that for a clause $C\theta = A \leftarrow B_1 \wedge \ldots \wedge B_n \in \mathcal{H}$, if A has no variables, then B_i $(i = 1, \dots, n)$ also has no variables.

3.2 Introduction of propositional variables

If the hypothesis space is $\mathcal{H}$, every ILP problem is interpreted as a search problem of the subset of $\mathcal{H}$. The enumeration is understood as a problem involving the family of sets. This means we can formulate the enumeration problem as an identification problem of a Boolean function with appropriate propositional variables.

For each clause $C \in \mathcal{H}$, we introduce a propositional variable $v_{C\in\Sigma}$ which becomes true if and only if clause C is in hypothesis Σ. For readability, we use an expression $[C \in \Sigma]$ instead of $v_{C\in\Sigma}$, that is we let

$$C \in \Sigma \Leftrightarrow [C \in \Sigma] = \mathbf{T}. \tag{2}$$

We construct a BDD that represents the set of hypotheses with these propositional variables.

Example 2. When the hypothesis space $\mathcal{H}$ is defined as

$$\mathcal{H} = \left\{ \begin{array}{ccc} e(0), & e(x), & \\ e(s(0)), & e(s(x)), & e(s(x)) \leftarrow e(x), \\ e(s^2(0)), & e(s^2(x)), & e(s^2(x)) \leftarrow e(x), \\ \multicolumn{3}{l}{e(s^2(x)) \leftarrow e(s(x)),\ e(s^2(x)) \leftarrow e(s(x)) \wedge e(x)} \end{array} \right\},$$

an example of an assignment to the propositional variables generated is:

$$\begin{array}{ll} [e(0) \in \Sigma] = \mathbf{T}, & [e(s^2(0)) \in \Sigma] = \mathbf{F}, \\ [e(x) \in \Sigma] = \mathbf{F}, & [e(s^2(x)) \in \Sigma] = \mathbf{F}, \\ [e(s(0)) \in \Sigma] = \mathbf{F}, & [e(s^2(x)) \leftarrow e(x) \in \Sigma] = \mathbf{T}, \\ [e(s(x)) \in \Sigma] = \mathbf{F}, & [e(s^2(x)) \leftarrow e(s(x)) \in \Sigma] = \mathbf{F}, \\ [e(s(x)) \leftarrow e(x) \in \Sigma] = \mathbf{F}, & [e(s^2(x)) \leftarrow e(s(x)) \wedge e(x) \in \Sigma] = \mathbf{F}. \end{array}$$

We get the following hypothesis from this assignment:

$$\Sigma = \{e(0), e(s^2(x)) \leftarrow e(x)\}.$$

3.3 Construction of BDDs

In this section, we show how to construct a BDD that represents the set of hypotheses. We assume that positive examples $\mathcal{E}^+$, negative examples $\mathcal{E}^-$, background knowledge $\mathcal{B}$, and hypothesis space $\mathcal{H}$ are given. We also assume that $\mathcal{E}^+$, $\mathcal{E}^-$, and $\mathcal{B}$ consist of only ground atoms. Let us denote the ILP problem as $P = (\mathcal{E}^+, \mathcal{E}^-, \mathcal{B}, \mathcal{H})$. For $C \in \mathcal{E}^+ \cup \mathcal{E}^-$, we define F_C as a BDD that represents the Boolean function whose input corresponds to $\Sigma \subseteq \mathcal{H}$; it becomes true if and only if $\Sigma \cup \mathcal{B} \models C$ on problem P. We also define I_C as a BDD that represents the Boolean variable $[C \in \Sigma]$, and BK_C as a BDD that represents a constant which becomes true if and only if $C \in \mathcal{B}$. By using binary operations between BDDs, then F_C where $C \in \mathcal{E}^+ \cup \mathcal{E}^-$ is recursively defined as

$$F_C = BK_C \vee \bigvee_{\substack{D \in \mathcal{H} \\ \exists \theta \\ D\theta = C \leftarrow B_1 \wedge \ldots \wedge B_n}} \left(I_D \wedge \bigwedge F_{B_i} \right). \tag{3}$$

The right side of equation (3) represents the fact that $\Sigma \cup \mathcal{B} \models C$ if $C \in \mathcal{B}$, or there exists substitution θ for definite clause $D \in \mathcal{H}$ such that the head of $D\theta$ becomes C by the substitution θ, and $\Sigma \cup \mathcal{B} \models B_i$ $(i = 1, \ldots, n)$ where $B_1, \ldots, B_n$ are the atoms of the body of $D\theta$.

A BDD that represents the set of hypotheses of problem P is

$$\bigwedge_{C \in \mathcal{E}^+} F_C \wedge \bigwedge_{C \in \mathcal{E}^-} \neg F_C. \tag{4}$$

The goal is to construct a BDD that represents the expression (4). We calculate each F_C where $C \in \mathcal{E}^+ \cup \mathcal{E}^-$ following equation (3). The algorithm is written as **Algorithm 1**. Here we say that $F_{\mathbf{T}}$ is a BDD that represents the constant *true*, and $F_{\mathbf{F}}$ is a BDD that represents the constant $false$. The function **constructBDD** returns the BDD that represents the set of hypotheses by calling **constructF** for each clause $C \in \mathcal{E}^+ \cup \mathcal{E}^-$. The function **constructF** takes clause C as an input and returns a BDD that represents the Boolean function that becomes true if and only if $\Sigma \cup \mathcal{B} \models C$.

Algorithm 1 Construction of a BDD that represents the set of hypotheses of ILP

Input: positive examples $\mathcal{E}^+$, negative examples $\mathcal{E}^-$, background knowledge $\mathcal{B}$, hypothesis space $\mathcal{H}$
Output: a BDD that represents the set of hypotheses

1: **constructBDD**$(\mathcal{E}^+, \mathcal{E}^-, \mathcal{B}, \mathcal{H})$
2: $\quad L \leftarrow \emptyset$
3: $\quad$**for each** C **in** $\mathcal{E}^+$
4: $\quad\quad F_C \leftarrow$ **constructF**$(C, \mathcal{B}, \mathcal{H}, L)$
5: $\quad\quad$Add F_C to L
6: $\quad$**for each** C **in** $\mathcal{E}^-$
7: $\quad\quad F_C \leftarrow$ **constructF**$(C, \mathcal{B}, \mathcal{H}, L)$
8: $\quad\quad$Add F_C to L
9: $\quad$**return** $\bigwedge_{C\in\mathcal{E}^+} F_C \wedge \bigwedge_{C\in\mathcal{E}^-} \neg F_C$

10: **constructF**$(C, \mathcal{B}, \mathcal{H}, L)$
11: $\quad$**if** $F_C \in L$ **then**
12: $\quad\quad$**return** F_C
13: $\quad$**else if** $C \in \mathcal{B}$ then
14: $\quad\quad$**return** $F_{\mathbf{T}}$
15: $\quad$**else if** θ exists for $D \in \mathcal{H}$ such that $D\theta = C \leftarrow B_1 \wedge \ldots \wedge B_n$ then
16: $\quad\quad F_{B_i} \leftarrow$ **constructF**$(B_i, \mathcal{B}, \mathcal{H}, L)$
17: $\quad\quad$**return** $\bigvee_{\substack{D\in\mathcal{H} \\ \exists\theta \\ D\theta=C\leftarrow B_1\wedge\ldots\wedge B_n}} (I_D \wedge \bigwedge F_{B_i})$
18: $\quad$**else**
19: $\quad\quad$**return** $F_{\mathbf{F}}$

Example 3. We show how to construct a BDD that represents a set of hypotheses. We assume the positive examples $\mathcal{E}^+$, the negative examples $\mathcal{E}^-$ and background knowledge $\mathcal{B}$ in *Example 1*, and the hypothesis space $\mathcal{H}$ in *Example 2*. $F_{e(0)}$, $F_{e(s(0))}$, and $F_{e(s^2(0))}$ are constructed as follows,

$$\begin{aligned}
F_{e(0)} &= I_{e(0)} \vee I_{e(x)}, \\
F_{e(s(0))} &= I_{e(s(0))} \vee I_{e(x)} \vee I_{e(s(x))} \vee \left(I_{e(s(x))\leftarrow e(x)} \wedge F_{e(0)}\right), \\
F_{e(s^2(0))} &= I_{e(s^2(0))} \vee I_{e(x)} \vee I_{e(s(x))} \vee I_{e(s^2(x))} \vee \left(I_{e(s^2(x))\leftarrow e(x)} \wedge F_{e(0)}\right) \\
&\quad \vee \left(I_{e(s^2(x))\leftarrow e(s(x))} \wedge F_{e(s(0))}\right) \vee \left(I_{e(s^2(x))\leftarrow e(s(x))\wedge e(x)} \wedge F_{e(s(0))} \wedge F_{e(0)}\right).
\end{aligned}$$

The BDD that represents the set of hypotheses is

$$F_{e(0)} \wedge F_{e(s^2(0))} \wedge \neg F_{e(s(0))}.$$

Generated propositional variables and the constructed BDD are as shown in Fig. 2.

The BDD in Fig. 2 represents the set of 28 hypotheses with 8 nodes. We get individual hypotheses by tracing from the root to terminal node $\boxed{1}$. Examples

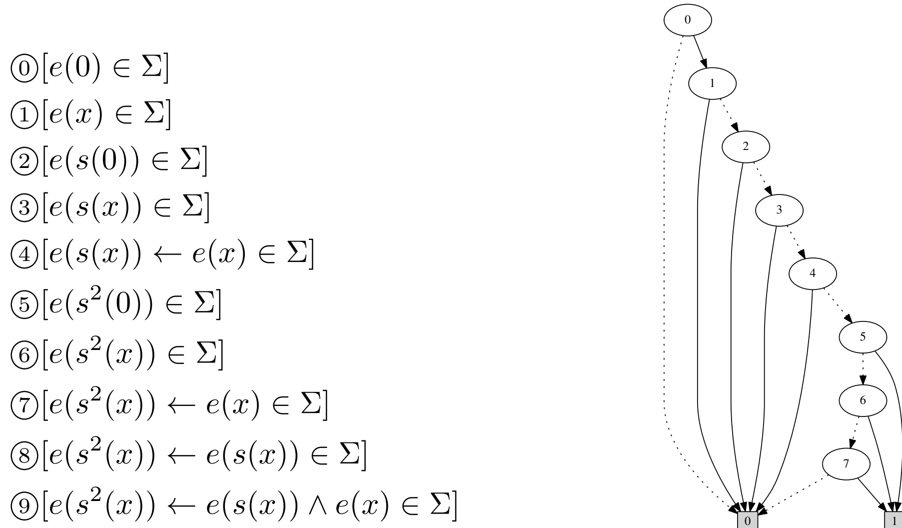

Fig. 2. Generated propositional variables and the BDD that represents the set of hypotheses

of the hypotheses of this problem are listed as follows,

$$\{e(0), e(s^2(0))\}, \{e(0), e(s^2(x))\}, \{e(0), e(s^2(x)) \leftarrow e(x)\}.$$

4 Applications of BDD-based enumeration

As described in Section 1, there are several benefits to using a BDD to enumerate hypotheses. Here we show how BDDs can be used for finding the hypothesis that maximizes a given evaluation function. Here we take the Minimum Description Length (MDL) principle as an example in a practical evaluation of the proposed method.

4.1 Minimum description length

Minimum description length (MDL) is a principle of model selection that states that the best model has the shortest description. The MDL principle was introduced to machine learning by Quinlan and Rivest [18]. MDL has been used in ILP in [13, 12, 4]. In Progol [14], the length of hypothesis Σ is measured as the number of atoms in the hypothesis [15] as follows

$$DL(\Sigma) = \sum_{C \in \Sigma} l(C), \tag{5}$$

where $l(C)$ is the number of atoms composing clause C.

4.2 Search for the best hypothesis

We can get the best hypothesis consistent with the data from the BDD by finding the minimum-weight path from the root to the terminal node. For each clause C in hypothesis space $\mathcal{H}$, we assign the number of atoms of C as the weight of the corresponding node's high-branch for a BDD written as (4). We denote the weight of the high-branch of node i as w_i with the value

$$w_i = l(C_i), \tag{6}$$

where C_i is the clause that corresponds to node i. Every low-branch has weight of 0.

We get the best hypothesis that minimize description length (5) by finding the minimum-weight path from the root to terminal node $\boxed{1}$ on a BDD whose weights are assigned following equation (6).

It is known that this type of optimization problem can be efficiently solved in time linear to BDD size by using dynamic programming (see [9]). The running time is $\mathcal{O}(n+B)$, where B is the number of BDD nodes. The algorithm is written as **Algorithm 2**. Here $high_i$ is the index of the node that is traced using the high-branch from node i, and low_i is the index of the node that is traced using the low-branch from node i. The value of $m[i]$ represents the minimal sum of the weights from the node i to a terminal node. The value of $t[i]$ represents which branch should be taken at node i to minimize the sum of the weights. Here $t[i] = 0$ means that the low-branch should be taken, $t[i] = 1$ means that the high-branch should be taken, and $t[i] = 2$ means that either branch can be taken.

Similarly, the problem of finding the top-k best hypothesis corresponds to finding the top-k shortest paths on the BDD. We omit detail due to the space limit, but it is straightforward to apply the algorithm for finding the top-k shortest paths [6] to obtain the top-k best hypotheses.

4.3 Enumeration of the best hypotheses

In some cases, some different hypotheses have the same best sore. In such cases, outputting just one of them is not effective. By using BDDs, it is easy to output a set of all such hypotheses. We call this the *top-tie BDD*. We denote F as a constructed BDD that represents the set of hypotheses. We can construct F by using the result of **Algorithm 2**. After calculating m_i, t_i for F in Section 4.2, we execute the following operations until the number of nodes of F becomes one. First, we set i as the index of the root. Then,

1. When $t_i = 0$, switch the target of the high-branch of the node i to terminal node $\boxed{0}$. Then call the procedure recursively for the partial BDD traced by the low-branch from node i.
2. When $t_i = 1$, switch the target of the low-branch of the node i to the terminal node $\boxed{0}$. Then call the procedure recursively for the partial BDD traced by the high-branch from node i.

Algorithm 2 Search for the best hypothesis

Input: BDD F, weights w, hypothesis space $\mathcal{H}$
Output: the best hypothesis Σ_{best}
1: $n \leftarrow$ the number of nodes of F
2: $m[0] \leftarrow \infty$
3: $m[1] \leftarrow 0$
4: **from** $i = 2$ **to** $n + 1$
5: **if** $m[low_i] < m[high_i] + w_i$
6: $m[i] \leftarrow m[low_i]$
7: $t[i] \leftarrow 0$
8: **if** $m[low_i] > m[high_i] + w_i$
9: $m[i] \leftarrow m[high_i] + w_i$
10: $t[i] \leftarrow 1$
11: **else**
12: $m[i] \leftarrow m[low_i]$
13: $t[i] \leftarrow 2$
14: $\Sigma \leftarrow \emptyset$
15: $j \leftarrow n + 1$
16: **while** $j \neq 1$ **then**
17: **if** $t_j = 1$
18: Add C_j to Σ
19: $j \leftarrow high_j$
20: **else**
21: $j \leftarrow low_j$
22: **return** Σ

3. When $t_i = 2$, call the procedure recursively for the partial BDDs traced by the high-branch and low-branch from node i, respectively.

We say F_T denotes the BDD constructed by these operations, and U is the set of all variables. S denotes the set of variables that appear on F, and $D = U - S$. F_D denotes the BDD that represents the logical conjunction of the negations of elements in D. The top-tie BDD is given by subjecting F_T and F_D to the logical "and" operation. The algorithm is schematized in **Algorithm 3**. Here *F.top* is a BDD that consists of only the root of F, *F.high* is a partial BDD that is traced using the high-branch from the root of F, and *F.low* is a partial BDD that is traced using the low-branch from the root of F.

Theorem 1. *The BDD yielded by **Algorithm 3** represents the set of the best hypotheses.*

5 Experiments

In order to show the efficiency of our method, we apply it to the following ILP problems,

Algorithm 3 Construction of top-tie BDD

Input: BDD F, set of variables U, array t
Output: a BDD that represents the set of the best hypotheses

```
 1: S ← the set of variables appearing on F
 2: D ← U − S
 3: F_D ← a BDD that represents the logical conjunction of the negations of elements
    of D
 4: F_T ← topTie(F, t)
 5: return F_T ∧ F_D

 6: topTie(F, t)
 7:   if the number of nodes of F = 1 then
 8:      return F
 9:   else
10:      j ← the index of the root
11:      if t[j] = 0
12:        return ¬ (F.top) ∧ topTie(F.low, t)
13:      else if t[j] = 1
14:        return F.top ∧ topTie(F.high, t)
15:      else
16:        return (¬F.top ∧ topTie(F.low, t)) ∨ (F.top ∧ topTie(F.high, t))
```

Classification problem of natural numbers We assumed that even numbers are given as positive examples, and odd numbers are given as negative examples.

Classification problem of real data We used a public data set. We assumed that instances whose class label corresponds to a target concept are given as positive examples, others are given as negative examples.

The experiment environment is as follows, OS: Ubuntu 17.10, CPU: Intel Xeon(R) E5-1650 v3 3.50GHz×12 , RAM: 125.8GB, Language: Scala 2.11.4, BDD implementation: JavaBDD[3].

5.1 Classification of natural numbers

We created an ILP problem involving natural numbers in which even numbers are given as positive examples, and odd numbers are given as negative examples. We constructed the BDD representing the set of hypotheses for various problem sizes. We assumed positive examples $\mathcal{E}^+$ and negative examples $\mathcal{E}^-$ are given as follows,
When n is even,

$$\mathcal{E}^+ = \{e(0), e(s^2(0)), \ldots, e(s^n(0))\},$$
$$\mathcal{E}^- = \{e(s(0)), e(s^3(0)), \ldots, e(s^{n+1}(0))\}.$$

[3] http://javabdd.sourceforge.net/

n	variables	nodes	hypotheses	BDD construction time	top-tie BDD construction time	best hypothesis search time
1	10	8	28	7.56msec	0.98msec	0.62msec
2	19	14	192	9.63msec	1.09msec	0.68msec
3	36	27	1.25×10^7	1.90×10msec	2.00msec	1.02msec
4	69	42	1.31×10^{13}	3.08×10msec	2.38msec	1.16msec
5	134	69	4.82×10^{32}	7.00×10msec	6.87msec	1.48msec
6	263	101	9.77×10^{63}	3.50×10^2msec	1.31×10msec	2.21msec
7	520	156	2.26×10^{141}	1.68×10^3msec	4.44×10msec	1.68msec
8	1033	219	$1.80 \times 10^{308}+$	1.20×10^4msec	1.88×10^2msec	2.66msec

Table 1. The results of the natural number problem

When n is odd,

$$\mathcal{E}^+ = \{e(0), e(s^2(0)), \ldots, e(s^{n+1}(0))\},$$
$$\mathcal{E}^- = \{e(s(0)), e(s^3(0)), \ldots, e(s^n(0))\}.$$

We assumed $\mathcal{B} = \emptyset$. Let $\mathcal{J}$ be the set consisting of definite clauses such as $C = A \leftarrow B_1 \wedge \ldots \wedge B_n$ which satisfy the following conditions. (i) C holds **Requirement 1** and **Requirement 2**, (ii) $|A| < \max_{D \in \mathcal{E}^+ \cup \mathcal{E}^-} |D|$, where $|A|$ is the sum of the number of constant symbols, variable symbols, and function symbols appearing in atom A. (iii) Predicate symbols, function symbols and constants appearing on C also appear on $\mathcal{E}^+ \cup \mathcal{E}^-$. (iv) If a variable appears on a clause, it also appears on all literals of the clause. We assume $\mathcal{H} = \mathcal{J} \cup \mathcal{E}^+ \cup \mathcal{E}^-$.

Example 4. In the case of $n = 1$, $\mathcal{E}^+$, $\mathcal{E}^-$, $\mathcal{B}$, and $\mathcal{H}$ are, respectively,

$$\mathcal{E}^+ = \{e(0), e(s^2(0))\},$$
$$\mathcal{E}^- = \{e(s(0))\},$$
$$\mathcal{B} = \emptyset, \text{ and}$$
$$\mathcal{H} = \left\{ \begin{array}{ccc} e(0), & e(x), & \\ e(s(0)), & e(s(x)), & e(s(x)) \leftarrow e(x), \\ e(s^2(0)), & e(s^2(x)), & e(s^2(x)) \leftarrow e(x), \\ e(s^2(x)) \leftarrow e(s(x)), & e(s^2(x)) \leftarrow e(s(x)) \wedge e(x) & \end{array} \right\}.$$

5.2 Results

The results are given in Table1 1. For each n, the number of nodes is smaller than that of hypotheses. $n = 8$ allows more than 1.80×10^{308} hypotheses to be represented by a BDD consisting of 219 nodes. This shows that many hypotheses can be compactly represented as a BDD. The BDD construction time is faster than searching for all hypotheses naively.

5.3 Classification of real data

We created an ILP problem from real data, and constructed a BDD that represents the set of the hypotheses. We also searched for the best hypothesis and constructed a BDD that represents the set of the best hypotheses. We assumed that the hypothesis space consists of just definite clauses that have body length under 2. We used data (1) Soybean(small)[4] and (2) Shuttle Landing Control[5] from UCI Machine Learning Repository[6]. We set the target concept to $D1$, no_auto.

5.4 Results

Problem	variables	nodes	hypotheses	BDD construction time
Soybean	2243	788498	$1.80 \times 10^{308}+$	13495msec
Shuttle	117	2345	6.76×10^{10}	30msec

Table 2. The result of constructing the BDD that represents the set of hypotheses

Problem	variables	nodes	best hypotheses	top-tie BDD construction time	best hypothesis search time
Soybean	2243	2251	2	1153msec	911msec
Shuttle	117	117	1	6msec	9msec

Table 3. The result of searching for the best hypothesis and construction of the top-tie BDD

In Table 2, the BDD has very few nodes compared to the number of hypotheses. We can conclude the set of the hypotheses can be expressed compactly as a BDD. Taking the number of hypotheses into account, we can get the best hypothesis faster by searching the BDD than by naive full search of the hypotheses. The best hypotheses found in problem (1) Soybean(small) are,

$$\Sigma_{best} = \{class(x, D1) \leftarrow stem_canker(x, above_soil)\},$$
$$\Sigma_{best} = \{class(x, D1) \leftarrow fruiting_bodies(x, absent)\}.$$

We can see that the best hypotheses consist of a simple definite clause that have body length of 1.

[4] https://archive.ics.uci.edu/ml/datasets/soybean+(small)
[5] https://archive.ics.uci.edu/ml/datasets/Shuttle+Landing+Control
[6] http://archive.ics.uci.edu/ml/index.php

In Table 3, the number of nodes of the top-tie BDD is close to the number of variables. This is because almost all variables have a constraint that should never be true. This yields a redundant BDD as almost all nodes have a high-branch to terminal node $\boxed{0}$ and low-branch to the next node. In this case, we can express the same set more compactly with a ZDD(Zero-suppressed Decision Diagram) [11].

6 Related Work

In the field of ILP, many hypothesis discovery methods that include inverse entailment [13] and saturation [20] have been proposed, but no hypothesis enumeration methods have been proposed. Yamamoto [22] proposed a general hypothesis finding method that applies *upward refinement* to the *residue hypothesis* that is given from $H \models \neg B \wedge E$, and also showed that it is a complete method for solving any hypothesis finding problem in clausal logic. In our method, all hypotheses are represented as a BDD.

BDDs and their variants are also widely used in the field of probabilistic logic programming for achieving efficient probabilistic inference and parameter learning [5, 8, 7]. The major difference from these approaches is that our algorithm exploits BDDs to solve enumeration problems of logic programs. One future work is to apply our enumeration algorithm to learning tasks of probabilistic relational rules [5]. Since our algorithm enables the set of all candidate logic programs to be stored, it will contribute to efficient search.

7 Conclusion

In this paper, we proposed a BDD-based method to enumerate hypotheses of an ILP. We pointed out that it is infeasible to enumerate all hypotheses without any restrictions, and we showed how to construct the BDD that represents the set of hypotheses in a bounded hypothesis space by applying BDD operations recursively. We also detailed our construction algorithm. As an application, we showed that users can get the best hypothesis following an evaluation function from the constructed BDD in time $\mathcal{O}(n)$, where n is the number of nodes. We also introduced a method to enumerate the best hypothesis as indicated by an evaluation function. We empirically showed that our method can be applied to real data.

Acknowledgements

This work was partly supported by JSPS KAKENHI Grant Number 17K19973.

References

1. Akers, S.B.: Binary decision diagrams. IEEE Trans. Comput. **27**(6), 509–516 (1978)

2. Arikawa, S., Shinohara, T., Yamamoto, A.: Learning elementary formal systems. In: Theoretical Computer Science. vol. 95, pp. 97–113 (1992)
3. Bryant, R.: Graph-based algorithms for boolean function manipulation. In: IEEE Trans. on Computers. vol. C–35, pp. 677–691 (1985)
4. Conklin, D., Witten, I.H.: Complexity-based induction. Machine Learning **16**(3), 203–225 (1994)
5. De Raedt, L., Kimmig, A., Toivonen, H.: Problog: A probabilistic prolog and its application in link discovery. In: Proc. IJCAI. pp. 2462–2467 (2007)
6. Eppstein, D.: Finding the k shortest paths. SIAM J. Comput. **28**(2), 652–673 (1999)
7. Firens, D., Van Den Broeck, G., Renkens, J., Shterionov, D., Gutmann, B., Thon, I., Jansssns, G., De Raedt, L.: Inference and learning in probabilistic logic programs using weighted boolean formulas. Theory and Practice of Logic Programming **15**(3), 358401 (2015)
8. Gutmann, B., Thon, I., De Raedt, L.: Learning the parameters of probabilistic logic programs from interpretations. In: Proc. ECML/PKDD. pp. 581–596 (2011)
9. Knuth, D.E.: The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams. Addison-Wesley Professional (2009)
10. Liaw, H.T., Lin, C.S.: On the obdd-representation of general boolean functions. IEEE Transactions on Computers **41**(6), 661–664 (1992)
11. Minato, S.: Zero-suppressed bdds for set manipulation in combinatorial problems. In: Proceedings of the 30th International Design Automation Conference. pp. 272–277. DAC '93, ACM (1993)
12. Muggleton, S., Srinivasan, A., Michael, B.: Compression, significance and accuracy. In: In Machine Learning: Proceedings of the Ninth International Workshop. pp. 338–347. Morgan Kaufmann (1992)
13. Muggleton, S.: A strategy for constructing new predicates in first order logic. In: In Proceedings of the Third European Working Session on Learning. pp. 123–130. Pitman (1988)
14. Muggleton, S.: Inverse entailment and progol. New Generation Computing **13**(3), 245–286 (Dec 1995)
15. Muggleton, S.: Learning from positive data. In: Muggleton, S. (ed.) Inductive Logic Programming. pp. 358–376. Springer Berlin Heidelberg (1997)
16. Nienhuys, C., Hwei, S., Wolf, R.D.: Foundations of Inductive Logic Programming (1997)
17. Plotkin, G.D.: A note on inductive generalization. Machine Intelligence **5**, 153–163 (1970)
18. Quinlan, J.R., Rivest, R.L.: Inferring decision trees using the minimum description length principle. Inf. Comput. **80**(3), 227–248 (1989)
19. Rissanen, J.: Paper: Modeling by shortest data description. Automatica **14**(5), 465–471 (1978)
20. Rouveriol, C.: Extensions of inversion of resolution applied to theory completion. In: Inductive Logic Programming. pp. 63–92 (1992)
21. Shapiro, E.Y.: An algorithm that infers theories from facts. In: Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 1. pp. 446–451. IJCAI'81, Morgan Kaufmann Publishers Inc. (1981)
22. Yamamoto, A.: Hypothesis finding based on upward refinement of residue hypotheses. Theor. Comput. Sci. **298**(1), 5–19 (2003)