Using Binary Decision Diagrams to Enumerate Inductive Logic Programming Solutions

Hikaru Shindo*, Masaaki Nishino**, Akihiro Yamamoto*

September 4, 2018

* Graduate School of Informatics, Kyoto University
** NTT Communication Science Laboratories

Abstract

- We propose an efficient algorithm for enumerating solutions of **Inductive Logic Programming** problem with **Binary Decision Diagram**s.
  - Basic formalization of ILP allows many potential solutions, and we might miss important solutions.
    $\Rightarrow$ Enumeration is fundamental technique to avoid such missing.
- Key idea: We use Binary Decision Diagram for enumeration.
  - **Binary Decision Diagram (BDD)** is a directed acyclic graph representing compactly a Boolean function.
- We show how to build recursively a Binary Decision Diagram that represents the set of solutions.

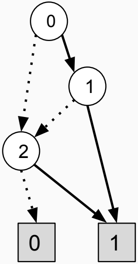


Table of contents

Introduction

Motivation

- ILP system generate solutions for given positive examples and negative examples. On the view point of logic, a lot of candidates of solutions might be generated.
- Every ILP system choose some appropriate solutions based on some criteria or its search method.

Example

$$\mathcal{E}^+ = \{p(a)\}, \quad \mathcal{E}^- = \{p(b)\}, \quad \mathcal{B} = \{\} \quad \Rightarrow \quad \Sigma = \{p(a)\}, \quad \Sigma = \{p(x) \leftarrow q(x), q(a)\}, \quad \vdots$$

We call the solution of ILP problem as **hypothesis**.

Fundamental idea: Enumeration of hypotheses

Enumeration of hypotheses is keeping all hypotheses.

Merits of the enumeration:

1. **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.
2. **Hypothesis selection**
   Users can select a hypothesis or compare some hypotheses using an evaluation function.
3. **Online-learning**
   We can efficiently perform online leaning, i.e., updating the current set of hypothesis when new examples are added.

Approach

- We assume that a finite set of clauses that can be an element of hypotheses is given explicitly.
  - Even in that finite space, enumerating all hypotheses naively is an implausible task because there are a serious amount of candidate hypotheses.
- To treat such large scale sets of hypotheses, we use **Binary Decision Diagram (BDD)**s that give compressed representation of hypotheses for enumeration.
- In this work, we developed an efficient recursive algorithm for constructing a BDD.

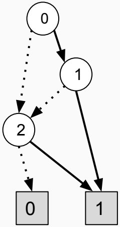
Contribution

- An efficient algorithm for enumerating hypotheses using BDDs.
- The class of ILP problems that we can apply our algorithm.
- An efficient algorithm to get the best hypothesis with an evaluation function.
- We empirically show that our method can be applied to real data.

Binary Decision Diagram and Enumeration of Solutions

Binary Decision Diagrams

A Binary Decision Diagram (BDD) is a directed acyclic graph that represents a Boolean function.



BDD that represents $\boldsymbol{F(x_0, x_1, x_2) = (x_0 \wedge x_1) \vee x_2}$

Binary operations between BDDs can be executed efficiently.

For example, given two BDDs representing logical functions $\boldsymbol{F}$ and $\boldsymbol{G}$, then the BDD representing $\boldsymbol{H = F \wedge G}$ can be computed in time linear to $\boldsymbol{F}$ and $\boldsymbol{G}$ sizes.

Inductive Logic Programming

In **Inductive Logic Programming (ILP)**, all data, background knowledge, and hypotheses are represented by first-order logic.

ILP Problem

Input Finite sets $\mathcal{E}^+$, $\mathcal{E}^-$, and $\mathcal{B}$ of ground atoms

Output A set of definite clauses Σ such that

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

Example

$$\mathcal{E}^+ = \{p(a)\}, \mathcal{E}^- = \{p(b)\}, \mathcal{B} = \{\}$$

$$\Sigma = \{p(a)\}, \{p(x) \leftarrow q(x), q(a)\}, \ldots$$

Using BDDs for enumerating ILP solutions

- To enumerate ILP hypotheses with BDDs, we introduce Boolean variables, because BDD is a representation of a Boolean function.
- Boolean variables make the **hypothesis enumeration problem** equivalent to the **problem of identifying a Boolean function**.
- **Hypothesis space** $\mathcal{H}$ is a finite set of clauses that can be an element of the hypothesis. We assume that $\mathcal{H}$ is given **explicitly**.

For each clause $C \in \mathcal{H}$, we introduce a propositional variable $v_{C \in \Sigma}$ that becomes true if and only if clause $C \in \Sigma$.
For readability, we represent $[C \in \Sigma]$ instead of $v_{C \in \Sigma}$,

$$C \in \Sigma \Leftrightarrow [C \in \Sigma] = T. \quad (1)$$

Building a BDD that represents hypotheses

We define F_A as a BDD that represents the Boolean function that becomes true if and only if $\Sigma \cup \mathcal{B} \models A$.

Then, a BDD that represents the set of hypotheses is

$$\bigwedge_{A \in \mathcal{E}^+} F_A \wedge \bigwedge_{A \in \mathcal{E}^-} \neg F_A.$$

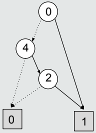
Example

Given:

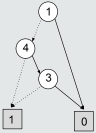
$$\mathcal{E}^+ = \{p(a)\}, \mathcal{E}^- = \{p(b)\}, \mathcal{B} = \{\},$$

The BDD to be built:

$$F_{p(a)} \wedge \neg F_{p(b)} =$$



$\wedge$



Solving ILP problem on the BDD

$\boldsymbol{I_C}$: the BDD that represents the Boolean variable $[\boldsymbol{C} \in \boldsymbol{\Sigma}]$
$\boldsymbol{BK_A}$: the BDD that represents a constant that becomes true if and only if $\boldsymbol{A} \in \boldsymbol{\mathcal{B}}$.

Then $\boldsymbol{F_A}$ for $\boldsymbol{A} \in \boldsymbol{\mathcal{E}}^{+} \cup \boldsymbol{\mathcal{E}}^{-}$ is recursively defined as

$$\boldsymbol{F_A} = \boldsymbol{BK_A} \vee \bigvee_{\substack{\boldsymbol{C} \in \boldsymbol{\mathcal{H}} \\ \exists \boldsymbol{\theta} \\ \boldsymbol{C\theta} = \boldsymbol{A} \leftarrow \boldsymbol{B_1} \wedge \ldots \wedge \boldsymbol{B_n}}} \left(\boldsymbol{I_C} \wedge \bigwedge \boldsymbol{F_{B_i}} \right). \tag{2}$$

The right side of equation (2) represents the fact that $\boldsymbol{\Sigma} \cup \boldsymbol{\mathcal{B}} \models \boldsymbol{A}$ if

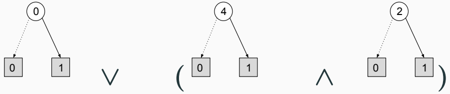
1. $\boldsymbol{A} \in \boldsymbol{\mathcal{B}}$, or
2. $\boldsymbol{A}$ is deduced by a substitution.

Example

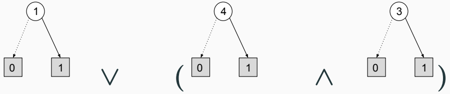
Introduced variables:

$$⓪[\boldsymbol{p(a) \in \Sigma}],\quad ①[\boldsymbol{p(b) \in \Sigma}],$$
$$②[\boldsymbol{q(a) \in \Sigma}],\quad ③[\boldsymbol{q(b) \in \Sigma}],\quad ④[\boldsymbol{p(x) \leftarrow q(x) \in \Sigma}]$$

$$\boldsymbol{F_{p(a)} = I_{p(a)} \lor (I_{p(x)\leftarrow q(x)} \land F_{q(a)})}$$



$$\boldsymbol{F_{p(b)} = I_{p(b)} \lor (I_{p(x)\leftarrow q(x)} \land F_{q(b)})}$$



Solving ILP problem on the BDD

Problem

$$\mathcal{E}^+ = \{p(a)\}, \mathcal{E}^- = \{p(b)\}, \mathcal{B} = \{\},$$

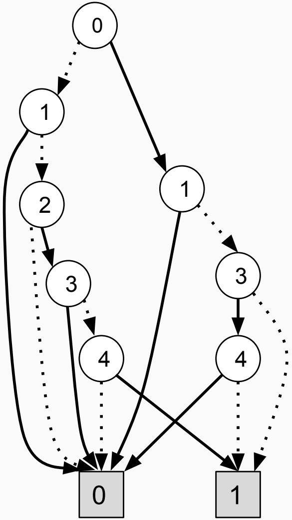
$$\mathcal{H} = \left\{ \begin{array}{lll} p(a), & p(b), & \\ q(a), & q(b), & p(x) \leftarrow q(x) \end{array} \right\}.$$

Introduced variables:

$$⓪[p(a) \in \Sigma]\ ①[p(b) \in \Sigma]$$
$$②[q(a) \in \Sigma]\ ③[q(b) \in \Sigma]$$
$$④[p(x) \leftarrow q(x) \in \Sigma]$$

Enumerated hypotheses:

$$\Sigma = \{p(a)\}$$
$$\Sigma = \{q(a), p(x) \leftarrow q(x)\}$$
$$\vdots$$

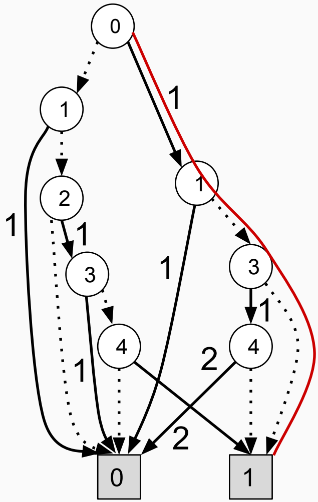


$F_{p(a)} \wedge \neg F_{p(b)}$

Applications

Introduced variables:

$$⓪[\boldsymbol{p(a) \in \Sigma}]$$
$$①[\boldsymbol{p(b) \in \Sigma}]$$
$$②[\boldsymbol{q(a) \in \Sigma}]$$
$$③[\boldsymbol{q(b) \in \Sigma}]$$
$$④[\boldsymbol{p(x) \leftarrow q(x) \in \Sigma}]$$



$\boldsymbol{F_{p(a)} \wedge \neg F_{p(b)}}$

Example

The hypothesis with minimum number of atoms:

$$\boldsymbol{\Sigma_{best} = \{p(a)\}}$$

This corresponds to the minimum-weight path colored red.

Experiments

Classification of natural numbers

When n is even,

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

When n is odd,

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

Example

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

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

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

Results

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

Table 1: The results of the natural number problem

Classification of real data

(1) Soybean(small)[1] and (2) Shuttle Landing Control[2] from UCI Machine Learning Repository[3].
Target concept: $D1$, no_auto respectively.

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 results of real data problem

One of the best hypotheses found in problem of Soybean(small) is,

$$\Sigma_{best} = \{class(x, D1) \leftarrow stem\_canker(x, above\_soil)\}.$$

[1] https://archive.ics.uci.edu/ml/datasets/soybean+(small)
[2] https://archive.ics.uci.edu/ml/datasets/Shuttle+Landing+Control
[3] http://archive.ics.uci.edu/ml/index.php

Conclusion and Future work

Conclusion and Future work

Conclusion

- We proposed a BDD-based method to enumerate hypotheses of an ILP.
- We showed that users can get the best hypothesis following an evaluation function from the constructed BDD.

Future Work

- Enumerating hypotheses that have some errors
- Combination with other ILP approaches
- Enumeration with other data structures

Requirements

Hypothesis space is a finite set of clauses that can be an element of the hypothesis.

We assume that the hypothesis space is given **explicitly**,
and it satisfies the following two requirements.

Requirement 1
The hypothesis space does not contain any **mutually recursive clauses**.

Requirement 2
The hypothesis space is **variable-bounded**.

Mutually recursive clauses

Mutually recursive clauses

Let $\mathcal{H}$ is a hypothesis space that is finite set of definite clauses. If a series of definite clauses $\{C_i \in \mathcal{H}\}_{i=0,\ldots,n}$ and substitutions $\theta_1, \ldots, \theta_n$ 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 $C_1, C_2, \ldots, C_n$ are **mutually recursive clauses**.

Having no mutually recursive clauses ensures that we can trace all literals present in the hypothesis space in a finite number of steps.

Variable-bounded

Variable-bounded

Definite clause $A \leftarrow B_1 \wedge \ldots \wedge B_n$ is **variable-bounded** if $v(A) \supseteq v(B_i)$ $(i = 1, \ldots, n)$, 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.

Being variable-bounded 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 $(i = 1, \ldots, n)$ also has no variables.