Linear Algebraic Abduction with Partial Evaluation

Tuan Nguyen[1(✉)], Katsumi Inoue[1(✉)], and Chiaki Sakama[2(✉)]

[1] National Institute of Informatics, Tokyo, Japan
{tuannq,inoue}@nii.ac.jp
[2] Wakayama University, Wakayama, Japan
sakama@wakayama-u.ac.jp

Abstract. Linear algebra is an ideal tool to redefine symbolic methods with the goal to achieve better scalability. In solving the abductive Horn propositional problem, the transpose of a program matrix has been exploited to develop an efficient exhaustive method. While it is competitive with other symbolic methods, there is much room for improvement in practice. In this paper, we propose to optimize the linear algebraic method for abduction using partial evaluation. This improvement considerably reduces the number of iterations in the main loop of the previous algorithm. Therefore, it improves practical performance especially with sparse representation in case there are multiple subgraphs of conjunctive conditions that can be computed in advance. The positive effect of partial evaluation has been confirmed using artificial benchmarks and real Failure Modes and Effects Analysis (FMEA)-based datasets.

Keywords: Abduction · Linear algebra · Partial evaluation

1 Introduction

Abduction is a form of explanatory reasoning that has been used for Artificial Intelligence (AI) in diagnosis and perception [15] as well as belief revision [4] and automated planning [8]. Logic-based abduction is formulated as the search for a set of abducible propositions that together with a background theory entails the observations while preserving consistency [7]. Recently, abductive reasoning has gained interests in connecting neural and symbolic reasoning [6] together with explainable AI [14,34].

Recently, several studies have been done to recognize the ability to use efficient parallel algorithms in linear algebra for computing logic programming (LP). For example, high-order tensors have been employed to support both deductive and inductive inferences for a limited class of logic programs [24]. In [29], Sato presented the use of first-order logic in vector spaces for Tarskian semantics, which demonstrates how tensorization realizes efficient computation of Datalog. Using a linear algebraic method, Sakama et al. explore relations between LP and tensor then propose algorithms for computation of LP models [27,28]. In [23], Nguyen et al. have analyzed the sparsity level of program matrices and proposed to employ sparse representation for scalable computation. Following this direction, Nguyen et al. have also exploited the sparse matrix

M. Hanus and D. Inclezan (Eds.): PADL 2023, LNCS 13880, pp. 197–215, 2023.
https://doi.org/10.1007/978-3-031-24841-2_13

representation to propose an efficient linear algebraic approach to abduction that incorporates solving Minimal Hitting Sets (MHS) problems [22].

Partial evaluation was introduced to generate a compiler from an interpreter based on the relationship between a formal description of the semantics of a programming language and an actual compiler [9]. The idea was also studied intensively in [3]. Then, Tamaki and Sato incorporated unfold and fold transformations in LP [33] as partial evaluation techniques, and Lloyd and Shepherdson have developed theoretical foundations for partial evaluation in LP [19]. According to Lloyd and Shepherdson, partial evaluation can be described as producing an equivalent logic program such that it should run more efficiently than the original one for reasoning steps. Following this direction, the idea of partial evaluation has been successfully employed to compute the least models of definite programs using linear algebraic computation [21]. Nguyen et al. have reported a significant improvement in terms of reducing runtime on both artificial data and real data (based on transitive closures of large network datasets) [21].

This paper aims at exploring the use of partial evaluation in abductive reasoning with linear algebraic approaches. We first propose an improvement to the linear algebraic algorithm for solving Propositional Horn Clause Abduction Problem (PHCAP). Then we present the efficiency of the method for solving PHCAP using the benchmarks based on FMEA. The rest of this paper is organized as follows: Sect. 2 reviews the background and some basic notions used in this paper; Sect. 3 presents the idea of partial evaluation using the linear algebraic approach with a theoretical foundation for correctness; Sect. 4 demonstrates experimental results using FMEA-based benchmarks; Sect. 5 discusses related work; Sect. 6 gives final remarks and discusses potential future works.

2 Preliminaries

We consider the language of propositional logic $\mathscr{L}$ that contains a finite set of propositional variables.

A *Horn logic program* is a finite set of *rules* of the form:

$$h \leftarrow b_1 \wedge \cdots \wedge b_m \quad (m \geq 0) \tag{1}$$

where h and b_i are propositional variables in $\mathscr{L}$. Given a program P, the set of all propositional variables appearing in P is the *Herbrand base* of P (written $\mathscr{B}_P$).
In (1) the left-hand side of $\leftarrow$ is called the *head* and the right-hand side is called the *body*. A Horn logic program P is called *singly defined* (*SD program*, for short) if $h_1 \neq h_2$ for any two different rules $h_1 \leftarrow B_1$ and $h_2 \leftarrow B_2$ in P where B_1 and B_2 are conjunctions of atoms. That is, no two rules have the same head in an SD program. When P contains more than one rule with the same head $(h \leftarrow B_1), \ldots, (h \leftarrow B_n)$ $(n > 1)$, replace them with a set of new rules:

$$\begin{aligned} &h \leftarrow b_1 \vee \cdots \vee b_n \quad (n > 1) \\ &b_1 \leftarrow B_1, \ \cdots, \ b_n \leftarrow B_n \end{aligned} \tag{2}$$

where $b_1, \ldots, b_n$ are new atoms such that $b_i \notin \mathscr{B}_P$ $(1 \leq i \leq n)$ and $b_i \neq b_j$ if $i \neq j$.
For convenience, we refer to (1) as an *And*-rule and (2) as an *Or*-rule.

Every Horn logic program P is transformed to $\Pi = Q \cup D$ such that Q is an SD program and D is a set of *Or*-rules. The resulting Π is called a *standardized program*. Therefore, a *standardized program* is a definite program such that there is no duplicate head atom in it and every rule is in the form of either *And*-rule or *Or*-rule. Note that the rule (2) is shorthand of n rules: $h \leftarrow b_1$, ..., $h \leftarrow b_n$, so a standardized program is considered a Horn logic program. Throughout the paper, a program means a standardized program unless stated otherwise. For each rule r of the form (1) or (2), define $head(r) = h$ and $body(r) = \{b_1, \ldots, b_m\}$ (or $body(r) = \{b_1, \ldots, b_n\}$). A rule is called a *fact* if $body(r) = \emptyset$. A rule is called a *constraint* if $head(r)$ is empty. A constraint $\leftarrow b_1 \wedge \cdots \wedge b_m$ is replaced with

$$\bot \leftarrow b_1 \wedge \cdots \wedge b_m$$

where $\bot$ is a symbol representing **False**. When there are multiple constraints, say $(\bot \leftarrow B_1), \ldots, (\bot \leftarrow B_n)$, they are transformed to

$$\bot \leftarrow \bot_1 \vee \cdots \vee \bot_n \quad \text{and} \quad \bot_i \leftarrow B_i \;\; (i = 1, \ldots, n)$$

where $\bot_i \notin \mathscr{B}_P$ is a new symbol. An *interpretation* $I \,(\subseteq \mathscr{B}_P)$ is a *model* of a program P if $\{b_1, \ldots, b_m\} \subseteq I$ implies $h \in I$ for every rule (1) in P, and $\{b_1, \ldots, b_n\} \cap I \neq \emptyset$ implies $h \in I$ for every rule (2) in P. A model I is the *least model* of P (written LM_P) if $I \subseteq J$ for any model J of P. We write $P \models a$ when $a \in LM_P$. For a set $S = \{a_1, \ldots, a_n\}$ of ground atoms, we write $P \models S$ if $P \models a_1 \wedge \cdots \wedge a_n$. A program P is *consistent* if $P \not\models \bot$.

Definition 1 Horn clause abduction: A *Propositional Horn Clause Abduction Problem (PHCAP)* consists of a tuple $\langle \mathscr{L}, \mathbb{H}, \mathbb{O}, P \rangle$, where $\mathbb{H} \subseteq \mathscr{L}$ (called *hypotheses* or *abducibles*), $\mathbb{O} \subseteq \mathscr{L}$ (called *observations*), and P is a consistent Horn logic program.

A logic program P is associated with a *dependency graph* (V, E), where the nodes V are the atoms of P and, for each rule from P, there are edges in E from the atoms appearing in the body to the atom in the head. We refer to the node of an *And*-rule and the node of an *Or*-rule as *And*-node and *Or*-node respectively. In this paper, we assume a program P is *acyclic* [1] and in its standardized form. Without loss of generality, we assume that any abducible atom $h \in \mathbb{H}$ does not appear in any head of the rule in P. If there exist $h \in \mathbb{H}$ and a rule $r : h \leftarrow body(r) \in P$, we can replace r with $r' : h \leftarrow body(r) \vee h'$ in P and then replace h by h' in $\mathbb{H}$. If r is in the form (2), then r' is an *Or*-rule, and no need to further update r'. On the other hand, if r is in the form (1), then we can update r' to become an *Or*-rule by introducing an *And*-rule $b_r \leftarrow body(r)$ in P and then replace $body(r)$ by b_r in r'.

Definition 2 Explanation of PHCAP: A set $E \subseteq \mathbb{H}$ is called a *solution* of a PHCAP $\langle \mathscr{L}, \mathbb{H}, \mathbb{O}, P \rangle$ if $P \cup E \models \mathbb{O}$ and $P \cup E$ is consistent. E is also called an *explanation* of $\mathbb{O}$. An explanation E of $\mathbb{O}$ is *minimal* if there is no explanation E' of $\mathbb{O}$ such that $E' \subset E$.

In this paper, the goal is to propose an algorithm finding the set $\mathbb{E}$ of all minimal explanations E for a PHCAP $\langle \mathscr{L}, \mathbb{H}, \mathbb{O}, P \rangle$. Deciding if there is a solution of a PHCAP (or $\mathbb{E} \neq \emptyset$) is *NP*-complete [7,32]. That is proved by a transformation from a satisfiability problem [10].

In PHCAP, P is partitioned into P_{And} - a set of And-rules of the form (1), and P_{Or} - a set of Or-rules of the form (2). Given P, define $head(P) = \{head(r) \mid r \in P\}$, $head(P_{And}) = \{head(r) \mid r \in P_{And}\}$, and $head(P_{Or}) = \{head(r) \mid r \in P_{Or}\}$.

3 Linear Algebraic Abduction with Partial Evaluation

3.1 Linear Algebraic Computation of Abduction

We first review the method of encoding and computing explanation in vector spaces that has been proposed in [22].

Definition 3. Matrix representation of standardized programs in PHCAP[22]**:** Let $\langle \mathscr{L}, \mathbb{H}, \mathbb{O}, P \rangle$ be a PHCAP such that P is a standardized program with $\mathscr{L} = \{p_1, \ldots, p_n\}$. Then P is represented by a *program matrix* $M_P \in \mathbb{R}^{n\times n}$ $(n = |\mathscr{L}|)$ such that for each element a_{ij} $(1 \leq i, j \leq n)$ in M_P:

1. $a_{ij_k} = \frac{1}{m}$ $(1 \leq k \leq m; 1 \leq i, j_k \leq n)$ if $p_i \leftarrow p_{j_1} \wedge \cdots \wedge p_{j_m}$ is in P_{And} and $m > 0$;
2. $a_{ij_k} = 1$ $(1 \leq k \leq l; 1 \leq i, j_k \leq n)$ if $p_i \leftarrow p_{j_1} \vee \cdots \vee p_{j_l}$ is in P_{Or};
3. $a_{ii} = 1$ if $p_i \leftarrow$ is in P_{And} or $p_i \in \mathbb{H}$;
4. $a_{ij} = 0$, otherwise.

Compared with the program matrix definition in [28], Definition 3 has an update in the condition 3 that we set 1 for all abducible atoms $p_i \in \mathbb{H}$. The program matrix is used to compute deduction, while in abductive reasoning, we do it in reverse. We then exploit the matrix for deduction to define a matrix that we can use for abductive reasoning.

Definition 4. Abductive matrix of PHCAP [22]**:** Suppose that a PHCAP has P with its *program matrix* M_P. The *abductive matrix* of P is the transpose of M_P represented as ${M_P}^T$.

In our method, we distinguish And-rules and Or-rules and handle them separately. Thus, it is crucial to have a simpler version of the abductive matrix for efficient computation.

Definition 5. Reduct abductive matrix of PHCAP: We can obtain a *reduct abductive matrix* $M_P(P^r_{And})^T$ from the abductive matrix ${M_P}^T$ by:

1. Removing all columns *w.r.t.* Or-rules in P_{Or}.
2. Setting 1 at the diagonal corresponding to all atoms which are heads of Or-rules.

We should note that this is a proper version of the previous definition in [22] that we will explain in detail later in this section. The reduct abductive matrix is the key component to define the partial evaluation method.

The goal of PHCAP is to find the set of minimal explanations $\mathbb{E}$ according to Definition 2. Therefore, we need to define a representation of explanations in vector spaces.

Definition 6. Vector representation of subsets in PHCAP[22]**:** Any subset $s \subseteq \mathscr{L}$ can be represented by a vector v of the length $|\mathscr{L}|$ such that the i-th value $v[i] = 1$ $(1 \leq i \leq |\mathscr{L}|)$ iff the i-th atom p_i of $\mathscr{L}$ is in s; otherwise $v[i] = 0$.

Using Definition 6, we can represent any $E \in \mathbb{E}$ by a column vector $E \in \mathbb{R}^{|\mathscr{L}|\times 1}$. To compute E, we define an *explanation vector* $v \in \mathbb{R}^{|\mathscr{L}|\times 1}$. We use the explanation vector v to demonstrate linear algebraic computation of abduction to reach an explanation E starting from an initial vector $v = v(\mathbb{O})$ which is the observation vector (note that we can use the notation $\mathbb{O}$ as a vector without the function notation $v()$ as stated before). At each computation step, we can interpret the meaning of the explanation vector v as: in order to explain $\mathbb{O}$, we have to explain all atoms v_i such that $v[i] > 0$.
An answer of PHCAP is a vector satisfying the following condition:

Definition 7. Answer of a PHCAP[22]: The explanation vector v *reaches* an answer E if $v \subseteq \mathbb{H}$. This condition can be written in linear algebra as follows:

$$\theta(v + \mathbb{H}) \leq \theta(\mathbb{H}) \tag{3}$$

where $\mathbb{H}$ is the shorthand of $v(\mathbb{H})$ which is the hypotheses set/vector. θ is a thresholding function mapping an element x of a vector/matrix to 0 if $x < 1$; otherwise map x to 1.

We here mention again Algorithm 1 in [22]. The main idea is built upon the two 1-step abduction for P_{And} (line 5) and P_{Or} (line 19) based on *And*-computable and *Or*-computable conditions. Each 1-step abduction applies on an *explanation vector* starting from the *observation vector* $\mathbb{O}$ until we reach an answer. During the abduction process, the *explanation vector* may "grow" to an *explanation matrix*, denoted by M, as *Or*-rules create new possible branches. Thus, we can abduce explanations by computing matrix multiplication (for *And*-computable matrices), and solving a corresponding MHS problem (for *Or*-computable matrices). Further detailed definitions and proofs of the method are presented in [22].

Algorithm 1. Explanations finding in a vector space

```
Input: PHCAP consists of a tuple $\langle \mathscr{L}, \mathbb{H}, \mathbb{O}, P \rangle$
Output: Set of explanations $\mathbb{E}$
1: Create an abductive matrix $M_P{}^T$ from $P$
2: Initialize the observation matrix $M$ from $\mathbb{O}$ (obtained
   directly from the observation vector $\mathbb{O}$)
3: $\mathbb{E} = \emptyset$
4: while True do
5:     $M' = M_P{}^T \cdot M$
6:     $M' = \textbf{consistent}(M')$
7:     $v\_sum = sum_{col}(M') < 1 - \varepsilon$
8:     $M' = M'[v\_sum = \textbf{False}]$
9:     if $M' = M$ or $M' = \emptyset$ then
10:        $v\_ans = \theta(M + \mathbb{H}) \leq \theta(\mathbb{H})$
11:        $\mathbb{E} = \mathbb{E} \cup M[v\_ans = \textbf{True}]$
12:        return minimal($\mathbb{E}$)
13:    do
14:        $v\_ans = \theta(M' + \mathbb{H}) \leq \theta(\mathbb{H})$
15:        $\mathbb{E} = \mathbb{E} \cup M'[v\_ans = \textbf{True}]$
16:        $M' = M'[v\_ans = \textbf{False}]$
17:        $M = M \cup M'[\textbf{not}\ Or\text{-computable}]$
18:        $M' = M'[Or\text{-computable}]$
19:        $M' = \bigcup_{\forall v \in M'} \bigcup_{\forall s \in \textbf{MHS}(\mathbb{S}_{(v, P_{Or})})} \left( (v \setminus head(P_{Or})) \cup s \right)$
20:        $M' = \textbf{consistent}(M')$
21:    while $M' \neq \emptyset$
```

Example 1. Consider a PHCAP $\langle \mathscr{L}, \mathbb{H}, \mathbb{O}, P \rangle$ such that:
$\mathscr{L} = \{obs,\ e_1,\ e_2,\ e_3,\ e_4,\ e_5,\ e_6,\ H_1,\ H_2,\ H_3\}$, $\mathbb{O} = \{obs\}$, $\mathbb{H} = \{H_1,\ H_2,\ H_3\}$, and $P = \{obs \leftarrow e_1,\ e_1 \leftarrow e_2 \wedge e_3,\ e_2 \leftarrow e_4 \wedge e_5,\ e_2 \leftarrow e_5 \wedge e_6,\ e_3 \leftarrow e_5,\ e_4 \leftarrow H_1,\ e_5 \leftarrow H_2,\ e_6 \leftarrow H_3\ \}$.

1. The standardized program $P' = \{obs \leftarrow e_1,\ e_1 \leftarrow e_2 \wedge e_3,\ e_2 \leftarrow x_1 \vee x_2,\ x_1 \leftarrow e_4 \wedge e_5,\ x_2 \leftarrow e_5 \wedge e_6,\ e_3 \leftarrow e_5,\ e_4 \leftarrow H_1,\ e_5 \leftarrow H_2,\ e_6 \leftarrow H_3\ \}$ is represented by the abductive matrix: $M_P^T =$

$$\begin{array}{c|cccccccccccc}
 & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & H_1 & H_2 & H_3 & obs & x_1 & x_2 \\ \hline
e_1 & & & & & & & & & & 1.00 & & \\
e_2 & 0.50 & & & & & & & & & & & \\
e_3 & 0.50 & & & & & & & & & & & \\
e_4 & & & & & & & & & & & 0.50 & \\
e_5 & & & 1.00 & & & & & & & & 0.50 & 0.50 \\
e_6 & & & & & & & & & & & & 0.50 \\
H_1 & & & & 1.00 & & & 1.00 & & & & & \\
H_2 & & & & & 1.00 & & & 1.00 & & & & \\
H_3 & & & & & & 1.00 & & & 1.00 & & & \\
obs & & & & & & & & & & & & \\
x_1 & & 1.00 & & & & & & & & & & \\
x_2 & & 1.00 & & & & & & & & & &
\end{array}$$

2. Iteration 1:
- $M^{(1)} = \theta(M_P^T \cdot M^{(0)})$, where $M^{(0)} = \mathbb{O}$:

$$\begin{array}{c|c}
 & 0 \\ \hline
e_1 & 1.00 \\ e_2 & \\ e_3 & \\ e_4 & \\ e_5 & \\ e_6 & \\ H_1 & \\ H_2 & \\ H_3 & \\ obs & \\ x_1 & \\ x_2 &
\end{array}
=
\begin{array}{c|cccccccccccc}
 & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & H_1 & H_2 & H_3 & obs & x_1 & x_2 \\ \hline
e_1 & & & & & & & & & & 1.00 & & \\
e_2 & 0.50 & & & & & & & & & & & \\
e_3 & 0.50 & & & & & & & & & & & \\
e_4 & & & & & & & & & & & 0.50 & \\
e_5 & & & 1.00 & & & & & & & & 0.50 & 0.50 \\
e_6 & & & & & & & & & & & & 0.50 \\
H_1 & & & & 1.00 & & & 1.00 & & & & & \\
H_2 & & & & & 1.00 & & & 1.00 & & & & \\
H_3 & & & & & & 1.00 & & & 1.00 & & & \\
obs & & & & & & & & & & & & \\
x_1 & & 1.00 & & & & & & & & & & \\
x_2 & & 1.00 & & & & & & & & & &
\end{array}
\cdot
\begin{array}{c|c}
 & 0 \\ \hline
e_1 & \\ e_2 & \\ e_3 & \\ e_4 & \\ e_5 & \\ e_6 & \\ H_1 & \\ H_2 & \\ H_3 & \\ obs & 1.00 \\ x_1 & \\ x_2 &
\end{array}$$

3. Iteration 2:
- $M^{(2)} = \theta(M_P^T \cdot M^{(1)})$

$$\begin{array}{c|c}
 & 0 \\ \hline
e_1 & \\ e_2 & 0.50 \\ e_3 & 0.50 \\ e_4 & \\ e_5 & \\ e_6 & \\ H_1 & \\ H_2 & \\ H_3 & \\ obs & \\ x_1 & \\ x_2 &
\end{array}
=
\begin{array}{c|cccccccccccc}
 & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & H_1 & H_2 & H_3 & obs & x_1 & x_2 \\ \hline
e_1 & & & & & & & & & & 1.00 & & \\
e_2 & 0.50 & & & & & & & & & & & \\
e_3 & 0.50 & & & & & & & & & & & \\
e_4 & & & & & & & & & & & 0.50 & \\
e_5 & & & 1.00 & & & & & & & & 0.50 & 0.50 \\
e_6 & & & & & & & & & & & & 0.50 \\
H_1 & & & & 1.00 & & & 1.00 & & & & & \\
H_2 & & & & & 1.00 & & & 1.00 & & & & \\
H_3 & & & & & & 1.00 & & & 1.00 & & & \\
obs & & & & & & & & & & & & \\
x_1 & & 1.00 & & & & & & & & & & \\
x_2 & & 1.00 & & & & & & & & & &
\end{array}
\cdot
\begin{array}{c|c}
 & 0 \\ \hline
e_1 & 1.00 \\ e_2 & \\ e_3 & \\ e_4 & \\ e_5 & \\ e_6 & \\ H_1 & \\ H_2 & \\ H_3 & \\ obs & \\ x_1 & \\ x_2 &
\end{array}$$

- Solving MHS: $\{\ \{x_1,\ x_2\},\ \{e_3\}\ \}$. MHS solutions: $\{\ \{e_3,\ x_1\},\ \{e_3,\ x_2\}\ \} = M^{(3)}$.

4. Iteration 3:
- $M^{(4)} = \theta(M_P^T \cdot M^{(3)})$

$$\begin{array}{c|cc}
 & 0 & 1 \\ \hline
e_1 & & \\ e_2 & & \\ e_3 & & \\ e_4 & 0.25 & \\ e_5 & 0.75 & 0.75 \\ e_6 & & 0.25 \\ H_1 & & \\ H_2 & & \\ H_3 & & \\ obs & & \\ x_1 & & \\ x_2 & &
\end{array}
=
\begin{array}{c|cccccccccccc}
 & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & H_1 & H_2 & H_3 & obs & x_1 & x_2 \\ \hline
e_1 & & & & & & & & & & 1.00 & & \\
e_2 & 0.50 & & & & & & & & & & & \\
e_3 & 0.50 & & & & & & & & & & & \\
e_4 & & & & & & & & & & & 0.50 & \\
e_5 & & & 1.00 & & & & & & & & 0.50 & 0.50 \\
e_6 & & & & & & & & & & & & 0.50 \\
H_1 & & & & 1.00 & & & 1.00 & & & & & \\
H_2 & & & & & 1.00 & & & 1.00 & & & & \\
H_3 & & & & & & 1.00 & & & 1.00 & & & \\
obs & & & & & & & & & & & & \\
x_1 & & 1.00 & & & & & & & & & & \\
x_2 & & 1.00 & & & & & & & & & &
\end{array}
\cdot
\begin{array}{c|cc}
 & 0 & 1 \\ \hline
e_1 & & \\ e_2 & & \\ e_3 & 0.50 & 0.50 \\ e_4 & & \\ e_5 & & \\ e_6 & & \\ H_1 & & \\ H_2 & & \\ H_3 & & \\ obs & & \\ x_1 & 0.50 & \\ x_2 & & 0.50
\end{array}$$

5. Iteration 4:
   - $M^{(5)} = \theta(M_P^T \cdot M^{(4)})$

$$
\begin{array}{c|cc}
 & 0 & 1 \\
e_1 & & \\
e_2 & & \\
e_3 & & \\
e_4 & & \\
e_5 & & \\
e_6 & & \\
H_1 & 0.50 & \\
H_2 & 0.50 & 0.50 \\
H_3 & & 0.50 \\
obs & & \\
x_1 & & \\
x_2 & & \\
\end{array}
=
\begin{array}{c|cccccccccccc}
 & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & H_1 & H_2 & H_3 & obs & x_1 & x_2 \\
e_1 & & & & & & & & & & 1.00 & & \\
e_2 & 0.50 & & & & & & & & & & & \\
e_3 & 0.50 & & & & & & & & & & & \\
e_4 & & & & & & & & & & & 0.50 & \\
e_5 & & & 1.00 & & & & & & & & 0.50 & 0.50 \\
e_6 & & & & & & & & & & & & 0.50 \\
H_1 & & & & 1.00 & & & 1.00 & & & & & \\
H_2 & & & & & 1.00 & & & 1.00 & & & & \\
H_3 & & & & & & 1.00 & & & 1.00 & & & \\
obs & & & & & & & & & & & & \\
x_1 & & 1.00 & & & & & & & & & & \\
x_2 & & 1.00 & & & & & & & & & & \\
\end{array}
\cdot
\begin{array}{c|cc}
 & 0 & 1 \\
e_1 & & \\
e_2 & & \\
e_3 & & \\
e_4 & 0.50 & \\
e_5 & 0.50 & 0.50 \\
e_6 & & 0.50 \\
H_1 & & \\
H_2 & & \\
H_3 & & \\
obs & & \\
x_1 & & \\
x_2 & & \\
\end{array}
$$

6. The algorithm stops. Found minimal explanations: $\{\ \{H_1,\ H_2\},\ \{H_2,\ H_3\}\ \}$.

In solving this problem, Algorithm 1 takes four iterations and a call to the MHS solver. One can notice in Iteration 3 that e_3, appearing in both explanation vectors of $M^{(3)}$, is computed twice. Imagine if an e_3-like node is repeated multiple times, then the computation spending from the second time on is duplicated. To deal with this issue we employ the idea of partial evaluation which is going to be discussed in the next section.

3.2 Partial Evaluation

Now, we define the formal method of partial evaluation in solving PHCAP by adapting the definition of partial evaluation of definite programs in vector spaces in [21].

Definition 8 Partial evaluation in abduction: Let a PHCAP $\langle\, \mathscr{L}, \mathbb{H}, \mathbb{O}, P\,\rangle$ where P is a standardized program. For any *And*-rule $r = (h \leftarrow b_1 \wedge \cdots \wedge b_m)$ in P,

- if $body(r)$ contains an atom b_i $(1 \leq i \leq m)$ which is not the head of any rule in P, then remove r.
- otherwise, for each atom $b_i \in body(r)$ $(i = 1, \ldots, m)$, if there is an *And*-rule $b_i \leftarrow B_i$ in P (where B_i is a conjunction of atoms), then replace each b_i in $body(r)$ by the conjunction B_i.

The resulting rule is denoted by $\mathsf{unfold}(r)$. Define

$$\mathsf{peval}(P) = \bigcup_{r \in P_{And}} \mathsf{unfold}(r)\,.$$

$\mathsf{peval}(P)$ is called *partial evaluation* of P.

Example 2 (continue Example 1) .

- Let $P' = \{r_1, ..., r_9\}$ where:	- Unfolding rules of P' becomes:	- Then $\mathsf{peval}(P')$ consists of:
$r_1 = (obs \leftarrow e_1)$,	$\mathsf{unfold}(r_1) = (obs \leftarrow e_2 \wedge e_3)$,	$obs \leftarrow e_2 \wedge e_3$,
$r_2 = (e_1 \leftarrow e_2 \wedge e_3)$,	$\mathsf{unfold}(r_2) = (e_1 \leftarrow e_2 \wedge e_5)$,	$e_1 \leftarrow e_2 \wedge e_5$,
$r_3 = (e_2 \leftarrow x_1 \vee x_2)$,	$\mathsf{unfold}(r_3) = r_3$,	$e_2 \leftarrow x_1 \vee x_2$,
$r_4 = (x_1 \leftarrow e_4 \wedge e_5)$,	$\mathsf{unfold}(r_4) = (x_1 \leftarrow H_1 \wedge H_2)$,	$x_1 \leftarrow H_1 \wedge H_2$,
$r_5 = (x_2 \leftarrow e_5 \wedge e_6)$,	$\mathsf{unfold}(r_5) = (x_2 \leftarrow H_2 \wedge H_3)$,	$x_2 \leftarrow H_2 \wedge H_3$,
$r_6 = (e_3 \leftarrow e_5)$,	$\mathsf{unfold}(r_6) = (e_3 \leftarrow H_2)$,	$e_3 \leftarrow H_2$,
$r_7 = (e_4 \leftarrow H_1)$,	$\mathsf{unfold}(r_7) = r_7$,	$e_4 \leftarrow H_1$,
$r_8 = (e_5 \leftarrow H_2)$,	$\mathsf{unfold}(r_8) = r_8$,	$e_5 \leftarrow H_2$,
$r_9 = (e_6 \leftarrow H_3)$.	$\mathsf{unfold}(r_9) = r_9$.	$e_6 \leftarrow H_3$.

We do not consider unfolding rules by *Or*-rules and unfolding *Or*-rules, as in the deduction case considered in [21]. Obviously, $\mathsf{peval}(P) = \mathsf{peval}(P_{And})$ and $\mathsf{peval}(P)$ is a standardized program. The *reduct abductive matrix* $M_P(P^r_{And})^T$ is the representation of P_{And} as presented in Definition 5, therefore, we can base on $M_P(P^r_{And})^T$ to build up the matrix representation of $\mathsf{peval}(P)$.

Example 3 (continue Example 2) .
According to Definition 5 we have the *reduct abductive matrix*:

$$M_P(P'^{r}_{And})^T = \begin{array}{c|cccccccccccc}
 & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & H_1 & H_2 & H_3 & obs & x_1 & x_2 \\
e_1 & & & & & & & & & & 1.00 & & \\
e_2 & 0.50 & & & & & & & & & & & \\
e_3 & 0.50 & & & & & & & a & & & & \\
e_4 & & & & & & & & & & & 0.50 & \\
e_5 & & & 1.00 & & & & & & & & 0.50 & 0.50 \\
e_6 & & & & & & & & & & & & 0.50 \\
H_1 & & & & 1.00 & & & 1.00 & & & & & \\
H_2 & & & & & 1.00 & & & 1.00 & & & & \\
H_3 & & & & & & 1.00 & & & 1.00 & & & \\
obs & & & & & & & & & & & & \\
x_1 & & 1.00 & & & & & & & & & & \\
x_2 & & 1.00 & & & & & & & & & &
\end{array}$$

1. $\mathsf{peval}(P')$ can be obtained by computing the power of the *reduct abductive matrix*: $\left(M_P(P'^{r}_{And})^T\right)^2$, $\left(M_P(P'^{r}_{And})^T\right)^4$, ... $\left(M_P(P'^{r}_{And})^T\right)^{2^k}$ where k is the number of peval steps. Here, we reach a fixpoint at $k = 2$.

$$\left(M_P(P'^{r}_{And})^T\right)^4 = \begin{array}{c|cccccccccccc}
 & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & H_1 & H_2 & H_3 & obs & x_1 & x_2 \\
e_1 & & & & & & & & & & & & \\
e_2 & 0.50 & 1.00 & & & & & & & & 0.50 & & \\
e_3 & & & & & & & & & & & & \\
e_4 & & & & & & & & & & & & \\
e_5 & & & & & & & & & & & & \\
e_6 & & & & & & & & & & & & \\
H_1 & & & & 1.00 & & & 1.00 & & & & 0.50 & \\
H_2 & 0.50 & & 1.00 & & 1.00 & & & 1.00 & & 0.50 & 0.50 & 0.50 \\
H_3 & & & & & & 1.00 & & & 1.00 & & & 0.50 \\
obs & & & & & & & & & & & & \\
x_1 & & & & & & & & & & & & \\
x_2 & & & & & & & & & & & &
\end{array}$$

We refer to this “stable” matrix as $\mathsf{peval}(P)$ and take it to solve the PHCAP.

2. Iteration 1:
- $M^{(1)} = \theta(\mathsf{peval}(P) \cdot M^{(0)})$, where $M^{(0)} = \mathbb{O}$

$$\begin{array}{c|c}
 & 0 \\
e_1 & \\
e_2 & 0.50 \\
e_3 & \\
e_4 & \\
e_5 & \\
e_6 & \\
H_1 & \\
H_2 & 0.50 \\
H_3 & \\
obs & \\
x_1 & \\
x_2 &
\end{array}
=
\begin{array}{c|cccccccccccc}
 & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & H_1 & H_2 & H_3 & obs & x_1 & x_2 \\
e_1 & & & & & & & & & & & & \\
e_2 & 0.50 & 1.00 & & & & & & & & 0.50 & & \\
e_3 & & & & & & & & & & & & \\
e_4 & & & & & & & & & & & & \\
e_5 & & & & & & & & & & & & \\
e_6 & & & & & & & & & & & & \\
H_1 & & & & 1.00 & & & 1.00 & & & & 0.50 & \\
H_2 & 0.50 & & 1.00 & & 1.00 & & & 1.00 & & 0.50 & 0.50 & 0.50 \\
H_3 & & & & & & 1.00 & & & 1.00 & & & 0.50 \\
obs & & & & & & & & & & & & \\
x_1 & & & & & & & & & & & & \\
x_2 & & & & & & & & & & & &
\end{array}
\begin{array}{c|c}
 & 0 \\
e_1 & \\
e_2 & \\
e_3 & \\
e_4 & \\
e_5 & \\
e_6 & \\
H_1 & \\
H_2 & \\
H_3 & \\
obs & 1.00 \\
x_1 & \\
x_2 &
\end{array}$$

- Solving MHS: $\{\, \{x_1, x_2\}, \{H_2\} \,\}$. MHS solutions: $\{\, \{H_2, x_2\}, \{H_2, x_1\} \,\} = M^{(2)}$.

3. Iteration 2:
- $M^{(3)} = \theta(\mathsf{peval}(P) \cdot M^{(2)})$

$$\begin{array}{c|cc}
 & 0 & 1 \\
e_1 & & \\
e_2 & & \\
e_3 & & \\
e_4 & & \\
e_5 & & \\
e_6 & & \\
H_1 & & 0.25 \\
H_2 & 0.75 & 0.75 \\
H_3 & 0.25 & \\
obs & & \\
x_1 & & \\
x_2 & &
\end{array}
=
\begin{array}{c|cccccccccccc}
 & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & H_1 & H_2 & H_3 & obs & x_1 & x_2 \\
e_1 & & & & & & & & & & & & \\
e_2 & 0.50 & 1.00 & & & & & & & & 0.50 & & \\
e_3 & & & & & & & & & & & & \\
e_4 & & & & & & & & & & & & \\
e_5 & & & & & & & & & & & & \\
e_6 & & & & & & & & & & & & \\
H_1 & & & & 1.00 & & & 1.00 & & & & 0.50 & \\
H_2 & 0.50 & & 1.00 & & 1.00 & & & 1.00 & & 0.50 & 0.50 & 0.50 \\
H_3 & & & & & & 1.00 & & & 1.00 & & & 0.50 \\
obs & & & & & & & & & & & & \\
x_1 & & & & & & & & & & & & \\
x_2 & & & & & & & & & & & &
\end{array}
\begin{array}{c|cc}
 & 0 & 1 \\
e_1 & & \\
e_2 & & \\
e_3 & & \\
e_4 & & \\
e_5 & & \\
e_6 & & \\
H_1 & & \\
H_2 & 0.50 & 0.50 \\
H_3 & & \\
obs & & \\
x_1 & & 0.50 \\
x_2 & 0.50 &
\end{array}$$

4. The algorithm stops. Found minimal explanations: $\{\ \{H_1, H_2\}, \{H_2, H_3\}\ \}$.

As we can see in Example 3, partial evaluation precomputes all *And*-nodes and we can just reuse their explanation vectors immediately. The number of iterations is reduced from 4 in Example 1 to 2. Moreover, e_3 is already precomputed to H_2, so we do not need to recompute it twice. Thus, the effect of partial evaluation remarkably boosts the overall performance of Algorithm 1 by reducing the number of needed iterations and also cutting down the cost of redundant computation.

Now let us formalize the partial evaluation step.

Proposition 1. *Let $\langle \mathscr{L}, \mathbb{H}, \mathbb{O}, P \rangle$ be a PHCAP such that P is a standardized program. Let $M_P(P^r_{And})^T$ be the reduct abductive matrix of P, also let v_0 be the vector representing observation of the PHCAP.*
Then $\theta\left(\left(M_P(P^r_{And})^T\right)^2 \cdot v_0\right) = \theta\left(M_P(P^r_{And})^T \cdot \theta\left(M_P(P^r_{And})^T \cdot v_0\right)\right)$

Proof. There are two different cases that we need to consider:

- In case v_0 is an *Or*-computable vector, the matrix multiplication maintains all values of atoms appearing in the heads of *Or*-rules by $M_P(P^r_{And})^T \cdot v_0 = v_0$. This is because we set 1 at the diagonal of the reduct abductive matrix as in Definition 5.[1]
- Suppose that v_0 is an *And*-computable vector. An atom is defined by a single rule since P is a standardized program. Suppose that an atom p_i $(1 \leq i \leq n)$ is defined by $\frac{1}{m}$ of q_j and q_j is defined by $\frac{1}{l}$ of r_k in $M_P(P^r_{And})^T$. Then p_i is defined by $(\frac{1}{m} \times \frac{1}{l})$ of r_k via q_j in $M_P(P^r_{And})^T$, which is computed by the matrix product $\left(M_P(P^r_{And})^T\right)^2$. This corresponds to the result of abductively unfolding a rule $p_i \leftarrow q_1 \wedge \cdots \wedge q_m$ by a rule $q_j \leftarrow r_1 \wedge \cdots \wedge r_l$ $(1 \leq j \leq m)$ in P. $\theta(\left(M_P(P^r_{And})^T\right)^2 \boldsymbol{v}_0)$ then represents the results of two consecutive steps of 1-step abduction in P_{And}. And $\left(\left(M_P(P^r_{And})^T\right)^2 \cdot v_0\right)[i] \geq 1$ iff $M_P(P^r_{And})^T \cdot \theta\left(M_P(P^r_{And})^T \cdot v_0\right)[i] \geq 1$ for any $1 \leq i \leq n$.

Hence, the result holds. □

Partial evaluation has the effect of reducing deduction steps by unfolding rules in advance. Proposition 1 realizes this effect by computing matrix products. Partial evaluation is repeatedly performed as:

$$\mathsf{peval}^0(P) = P \quad \text{and} \quad \mathsf{peval}^k(P) = \mathsf{peval}(\mathsf{peval}^{k-1}(P)) \ \ (k \geq 1). \tag{4}$$

The k-step partial evaluation has the effect of realizing 2^k steps of deduction at once. Multiplying an explanation vector and the peval matrix thus realizes an exponential speed-up that has been demonstrated in Example 3.

Proposition 2. *Partial evaluation realized in Proposition 1 has a fixpoint.*

[1] This behavior is unlike the behavior of the previous definition in [22] that we set 0 at the diagonal that will eliminate all values of *Or*-rule head atoms in v_0.

Proof. Note that we assume a program is acyclic. As Algorithm 1 causes no change to atoms in the head of *Or*-rules, one can create a corresponding standardized program containing only *And*-rules. The resulting program, with only *And*-rules, is monotonic so it has a fixpoint for every initial vector. Thus, partial evaluation has a fixpoint. □

Accordingly, incorporating peval to Algorithm 1 is made easy by first finding the reduct abductive matrix and then computing the power of that matrix until we reach a fixpoint. Then we use the output vector to replace the abductive matrix in the Algorithm 1 for computing explanations. The motivation behind this idea is to take advantage of the recent advance in efficient linear algebra routines.

Intuitively speaking, non-zero elements in the reduct abductive matrix represent conjuncts appearing in each rule. By computing the power of this matrix, we assume all *And*-nodes are needed to explain the observation. Then we precompute the explanations for all these nodes. However, the good effect of partial evaluation depends on the graph structure of the PHCAP. If there are many *And*-nodes that just lead to "nothing" or somehow these subgraphs of *And*-rules are not repeated at a certain number of times. Then partial evaluation just does the same job as the normal approach but at a higher cost with computing the power of a matrix. We will evaluate the benefit of partial evaluation in the next section.

4 Experimental Results

In this section, we evaluate the efficiency of partial evaluation based on benchmark datasets that are used in [16,17,22]. The characteristics of the benchmark datasets are summarized below. Both dense and sparse formats are considered as the representation of program matrices and abductive matrices in the partial evaluation method.

- **Artificial samples I** (166 problems): deeper but narrower graph structure.
- **Artificial samples II** (117 problems)[2]: deeper and wider graph structure, some problems involve solving a large number of medium-size MHS problems.
- **FMEA samples** (213 problems): shallower but wider graph structure, usually involving a few (but) large-size MHS problems.

For further detailed statistics data, readers should follow the experimental setup in [22].

Additionally, to demonstrate the efficiency of partial evaluation, we do enhancing the benchmark dataset based on the transitive closure problem: $P = \{path(X, Y) \leftarrow edge(X, Y),\ path(X, Y) \leftarrow edge(X, Z) \wedge path(Z, Y)\}$. First, we generate a PHCAP problem based on the transitive closure of the following single line graph: $edge(1, 2)$, $edge(2, 3)$, $edge(3, 4)$, $edge(4, 5)$, $edge(5, 6)$, $edge(6, 7)$, $edge(7, 8)$, $edge(8, 9)$, $edge(9, 10)$. Then we consider the observation to be $path(1, 10)$, and look for the explanation of it. Obviously, we have to include all the edges of this graph in the explanation and the depth of the corresponding graph or *And*-rules is 10. Next, for each problem instance of the original benchmark, we enumerate rules of the form $e \leftarrow h$, where h is a hypothesis, and append the atom of the observation of the new PHCAP into this rule with a probability of 20%. The resulting problem is expected to have the subgraph of

[2] We excluded the unresolved problem `phcap_140_5_5_5.atms`.

And-rules occur more frequently.

Similar to the experiment setup in [17,22], each method is conducted 10 times with a limited runtime on each PHCAP problem to record the execution time and correctness of the output. The time limit for each run is 20 min, that is, if a solver cannot output the correct output within this limit, 40 min will be penalized to its execution time following PAR-2[3] as used in SAT competitions [12]. Accordingly, for each problem instance, we denote t as the effective solving time, t_{peval} as the time for the partial evaluation step, and t_p as the penalty time. Thus, $t+t_p$ is the total running time. Partial evaluation time t_{peval} and also the extra time for transforming to the standardized format are included in t. We also report t_{peval} separately to give a better insight. All the execution times are reported in Table 1 and Table 2.

The two parts of Table 3 and Table 4 compare the two methods in: the maximum number of explanation vectors ($max(|M|)$), the maximum η_z ($max(\eta_z(M))$), and the minimum sparsity $min(sparsity(M))$ for each explanation matrix. Finally, max_iter is the number of iterations of the main loop of each method, mhs_calls is the number of MHS problems, and $|\mathbb{E}|$ is the number of correct minimal explanations. For the methods with partial evaluation, we report $peval\_steps$ as the number of partial evaluation steps.

We refer to each method as *Sparse matrix - peval*, *Sparse matrix*, *Dense matrix - peval*, and *Dense matrix* for the linear algebraic method in Algorithm 1 with a sparse representation with partial evaluation, sparse representation without partial evaluation, dense representation with partial evaluation, and dense representation without partial evaluation respectively. Our code is implemented in Python 3.7 using Numpy and Scipy for matrices representation and computation. We also exploit the MHS enumerator provided by PySAT[4] for large-size MHS problems. All the source code and benchmark datasets in this paper will be available on GitHub: https://github.com/nqtuan0192/LinearAlgebraicComputationofAbduction. The computer we perform experiments has the following configurations: CPU: Intel® Xeon® Bronze 3106 @1.70GHz; RAM: 64GB DDR3 @1333MHz; OS: Ubuntu 18.04 LTS 64bit.

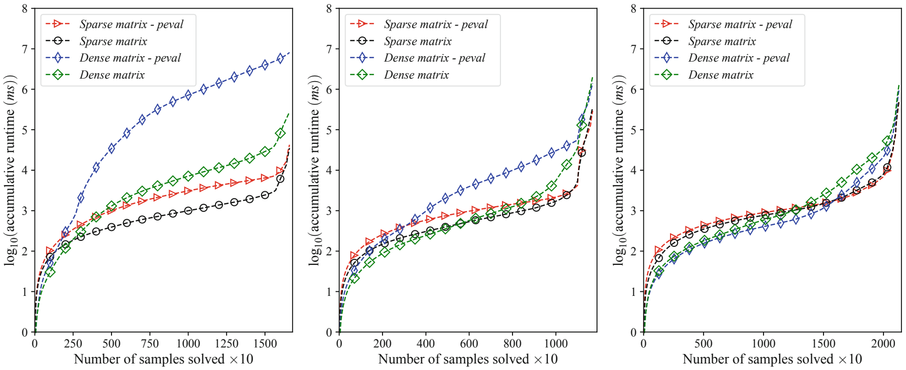
4.1 Original Benchmark

Table 1. Detailed execution results for the original benchmark.

Datasets	Artificial samples I (166 problems)			Artificial samples II (117 problems)			FMEA samples (213 problems)		
Algorithms	#solved/ #fastest	$t+t_p$ mean/std	t_{peval} mean/std	#solved/ #fastest	$t+t_p$ mean/std	t_{peval} mean/std	#solved/ #fastest	$t+t_p$ mean/std	t_{peval} mean/std
Sparse matrix - peval	**1,660** 89	4,243 93	514 19	**1,170** 246	**29,438** 112	124 48	**2,130** 726	**49,481** 1,214	84 4
Sparse matrix	**1,660** 1,401	**3,527** 29	– –	**1,170** 513	35,844 62	– –	2,130 150	53,553 1,254	– –
Dense matrix - peval	**1,660** 13	811,841 2,227	728,086 31,628	**1,170** 90	140,589 1,293	3,599 910	**2,130** 1,0007	98,614 2,950	25 3
Dense matrix	**1,660** 157	27,569 183	– –	**1,170** 321	205,279 1,866	– –	**2,130** 247	131,734 3,629	– –

[3] A PAR-2 score of a solver is defined as the sum of all runtimes for solved instances plus 2 times timeout for each unsolved instance.

[4] https://github.com/pysathq/pysat.



(a) Artificial samples I

(b) Artificial samples II

(c) FMEA samples

Fig. 1. Effective runtime by the number of solved samples for the original benchmark.

Figure 1 and Table 1 demonstrate the runtime trend and execution time comparison on the original benchmark, while Table 3 gives more detailed information about the sparsity analysis of the dataset in the benchmark. Overall, all algorithms can solve the entire benchmark without any problems. The experiment setup is similar to [22] so readers can compare the data reported in this section with other methods which were reported in [22].

In Artificial samples I, *Sparse matrix* is the fastest algorithm with 1,401 **#fastest** and it finishes the first with $3,527\,ms$ on average for each run. *Sparse matrix - peval*, which stands at second place, is slightly slower with average $4,243\,ms$ for each run, however, it only is the fastest algorithm in 89 problem instances. A similar trend that the algorithm with partial evaluation is not faster than the original version can be seen with the dense matrix format. In fact, *Dense matrix - peval* is considerably slow in this sample with $811,841\,ms$ for each average run, multiple times slower than *Dense matrix*. This could be explained by pointing out that the program matrix size in this dataset is relatively large with **mean** is $2,372.36$ as can be seen in the first part of Table 3. In this case, matrix multiplication with the dense format is costly and is not preferable.

In Artificial samples II, *Sparse matrix - peval* is the fastest algorithm with only $29,438\,ms$, while *Sparse matrix* takes $35,844\,ms$ for each run on average. However, *Sparse matrix* has higher **#fastest** than *Sparse matrix - peval* that is because many problems in the samples are relatively small. In this dataset, the execution time of *Dense matrix - peval* is significantly improved compared to that of *Dense matrix* with about 25%. In this dataset, the average abductive matrix size is not too large with **mean** is 451.90 while there are multiple branches being created as we see many *mhs_calls*. This condition is favorable for partial evaluation in precomputing multiple branches in advance.

In FMEA samples, a similar trend that partial evaluation significantly improves the original version can be seen in both *Sparse matrix - peval* and *Dense matrix - peval*. *Sparse matrix - peval* again is the fastest algorithm with no doubt, it finishes each run in only about $49,481\,ms$. In spite of that fact, *Dense matrix* is the algorithm with the highest **#fastest** - $1,007$. This is because the graph structure of this dataset is shallower, so it produces less complicated matrices that we can consider it is more preferable for dense computation.

4.2 Enhanced Benchmark

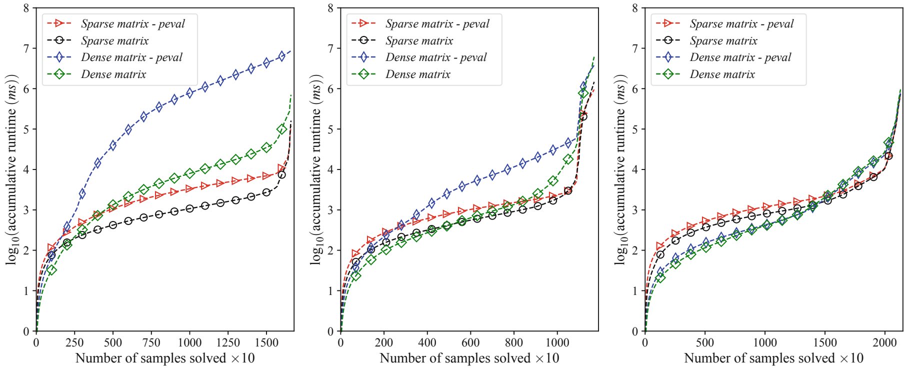
Figure 2 and Table 2 demonstrate the runtime trend and execution time comparison on the original benchmark, while Table 4 gives more detailed information about sparsity analysis of the dataset in the benchmark. Overall, the enhanced problems are more difficult than those in the original benchmarks as we see apparently all figures reported in Fig. 2 are higher than that in Fig. 1. However, similar to the original benchmark, all algorithms can solve the entire enhanced benchmark without any problems.

In the enhanced Artificial samples I, with enriched more subgraphs of *And*-rules, now the fastest algorithm is *Sparse matrix - peval* with $12,140\,ms$ for each run on average. Interestingly, *Sparse matrix* is the one with the highest **#fastest** $1,389$, although it is not the algorithm that finishes first. *Dense matrix - peval* still cannot catch up with *Dense matrix* even though the execution time of *Dense matrix* now is double what we can see in the previous benchmark. That is because the matrix size is relatively large so we need to increase the depth of embedded subgraphs to see a better effect of partial evaluation with the dense matrix implementation.

In the enhanced Artificial samples II, *Sparse matrix - peval* again takes the first position that it solves in 95,079 ms only for the whole problem samples and being fastest in 254 problem instances. *Sparse matrix* again has the highest **#fastest** 516 but it slower than *Sparse matrix - peval* more than 50% in solving the whole dataset. *Dense matrix - peval* is also faster than *Dense matrix* by more than 50% although there are 323 problems in which *Dense matrix* is the fastest.

Table 2. Detailed execution results for the enhanced benchmark datasets.

Datasets	Artificial samples I (166 problems)			Artificial samples II (117 problems)			FMEA samples (213 problems)		
Algorithms	#solved/ #fastest	$t+t_p$ mean/std	t_{peval} mean/std	#solved/ #fastest	$t+t_p$ mean/std	t_{peval} mean/std	#solved/ #fastest	$t+t_p$ mean/std	t_{peval} mean/**std**
Sparse matrix - peval	**1,660** 116	**12,140** 124	545 15	**1,170** 254	**95,079** 616	138 4	**2,130** 384	**72,776** 1,103	157 5
Sparse matrix	**1,660** 1.389	16,163 209	– –	**1,170** 516	147,444 1,508	– –	**2,130** 553	74,861 526	– –
Dense matrix - peval	**1,660** 5	869,922 2,434	799,965 58,500	**1,170** 77	380,033 2,228	4,483 688	**2,130** 436	81,837 1,005	103 10
Dense matrix	**1,660** 150	70,365 681	– –	**1,170** 323	613,422 3,651	– –	**2,130** 757	95,996 1,021	– –



(a) Artificial samples I

(b) Artificial samples II

(c) FMEA samples

Fig. 2. Effective runtime by the number of solved samples for the enhanced benchmark.

In the enhanced FMEA samples, *Sparse matrix - peval* once again outruns all other algorithms with $72,776\,ms$ on average for solving the entire dataset. Surprisingly, *Dense matrix* and *Dense matrix - peval* catch up closely with sparse versions that *Dense matrix* has highest **#fastest** with 757. In fact, the shape of the abductive matrices in this dataset is relatively small, and computing these matrices of this size is usually well-optimized. This also can benefit the partial evaluation as we can see *Dense matrix - peval* surpasses *Dense matrix* by more than 12% which is a remarkable improvement.

Discussion: In summary, partial evaluation remarkably improves the linear algebraic approach for abduction. The merit of partial evaluation is that it can be precomputed before abduction steps. Further, once it is computed, we can reuse it repeatedly for different abduction problems. The positive effect can be seen more clearly in case there are multiple subgraphs of *And*-rules exist in the corresponding graph of the PHCAP. In addition, partial evaluation especially boosts the method with sparse representation at a more steady level than with the dense matrix format as reported data for t_{peval} in Table 1 and Table 2.

Table 3. Statistics and sparsity analysis on original benchmark datasets.

Benchmark dataset	Artificial samples I (166 problems)				Artificial samples II (117 problems)				FMEA samples (213 problems)			
Algorithm 1 (without partial evaluation)	**mean**	**std**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**	**min**	**max**
$max(\|M\|)$	920.73	10,233.54	1.00	131,418.00	5,245.37	25,961.91	1.00	188,921.00	2,126.49	15,512.54	1.00	154,440.00
$max(\eta_z(M))$	32,162.44	386,905.76	1.00	4,983,288.00	235,884.85	1,138,981.38	1.00	7,302,298.00	43,738.87	334,393.40	1.00	3,459,456.00
$min(sparsity(M))$	0.98	0.05	0.67	1.00	0.94	0.10	0.52	1.00	0.79	0.13	0.46	0.99
max_iter	4.65	5.37	2.00	65.00	7.32	11.46	2.00	91.00	1.94	0.24	1.00	2.00
mhs_calls	5.74	28.22	0.00	349.00	23.15	118.52	0.00	1,208.00	0.93	0.26	0.00	1.00
$\|\mathbb{E}\|$	2.77	5.06	1.00	50.00	3.70	9.62	1.00	63.00	68.89	272.54	1.00	2,288.00
Algorithm 1 (without partial evaluation)	**mean**	**std**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**	**min**	**max**
$max(\|M\|)$	627.03	6,479.97	1.00	82,728.00	5,991.66	28,924.93	1.00	188,921.00	1.00	0.00	1.00	1.00
$max(\eta_z(M))$	21,210.33	246,116.37	1.00	3,167,154.00	267,195.89	1,301,308.57	1.00	9,648,741.00	1.00	0.00	1.00	1.00
$min(sparsity(M))$	0.98	0.05	0.67	1.00	0.94	0.09	0.54	1.00	0.97	0.02	0.89	0.99
max_iter	2.87	3.54	1.00	33.00	4.41	6.84	1.00	48.00	1.00	0.00	1.00	1.00
mhs_calls	5.67	27.49	0.00	339.00	30.59	195.41	0.00	2,067.00	0.93	0.26	0.00	1.00
$\|\mathbb{E}\|$	2.77	5.06	1.00	50.00	3.70	9.62	1.00	63.00	68.89	272.54	1.00	2,288.00
$peval\_steps$	3.78	0.95	2.00	5.00	3.71	0.81	2.00	6.00	2.00	0.00	2.00	2.00

Table 4. Statistics and sparsity analysis on benchmark datasets enhanced with transitive closure problem.

Benchmark dataset	Artificial samples I (166 problems)				Artificial samples II (117 problems)				FMEA samples (213 problems)			
Algorithm 1 (without partial evaluation)	**mean**	**std**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**	**min**	**max**
$max(\|M\|)$	1,094.92	12,382.96	1.00	159,138.00	7,889.15	34,097.99	1.00	193,943.00	2,918.32	19,190.31	1.00	183,960.00
$max(\eta\_z(M))$	38,796.85	470,699.48	1.00	6,063,138.00	357,363.52	1,587,936.23	1.00	10,607,545.00	59,601.64	410,555.07	1.00	3,886,020.00
$min(sparsity(M))$	0.99	0.03	0.83	1.00	0.96	0.06	0.61	1.00	0.92	0.05	0.8	1.00
max_iter	4.70	5.41	2.00	65.00	7.19	11.34	2.00	91.00	1.97	0.18	1.00	2.00
mhs_calls	5.75	28.22	0.00	349.00	24.83	120.31	0.00	1,208.00	0.93	0.26	0.00	1.00
$\|\mathbb{E}\|$	2.49	5.21	0.00	50.00	4.03	16.72	0.00	168.00	41.46	299.51	0.00	4,000.00
Algorithm 1 (without partial evaluation)	**mean**	**std**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**	**min**	**max**
$max(\|M\|)$	727.49	7,677.78	1.00	98,181.00	6,671.67	29,457.12	1.00	193,943.00	1,324.90	8,591.52	1.00	82,440.00
$max(\eta\_z(M))$	25,065.43	294,060.12	1.00	3,785,073.00	277,984.67	1,176,915.48	1.00	6,222,569.00	25,302.40	172,962.09	1.00	1,716,972.00
$min(sparsity(M))$	0.99	0.03	0.83	1.00	0.96	0.06	0.61	1.00	0.94	0.04	0.86	1.00
max_iter	2.91	3.56	1	33.00	4.44	6.93	1.00	48.00	1.66	0.47	1.00	2.00
mhs_calls	5.68	27.49	0.00	339.00	26.59	137.00	0.00	1,391.00	0.93	0.26	0.00	1.00
$\|\mathbb{E}\|$	2.49	5.21	0.00	50.00	4.03	16.72	0.00	168.00	41.46	299.51	0.00	4,000.00
$peval\_steps$	4.20	0.40	4.00	5.00	4.13	0.36	4.00	6.00	4.00	0.00	4.00	4.00

5 Related Work

Propositional abduction has been solved using propositional satisfiability (SAT) techniques in [13], in which a quantified MaxSAT is employed and implicit hitting sets are computed. Another approach to abduction is based on the search for stable models of a logic program [11]. In [25], Saikko et al. [25] have developed a technique to encode the propositional abduction problem as disjunctive logic programming under answer set semantics. Answer set programming has also been employed for first-order Horn abduction in [31], in which all atoms are abduced and weighted abduction is employed.

In terms of linear algebraic computation, Sato et al. [30] developed an approximate computation to abduce relations in Datalog [30], which is a new form of predicate invention in Inductive Logic Programming [20]. They did empirical experiments on linear and recursive cases and indicated that the approach can successfully abduce base relations, but their method cannot compute explanations consisting of possible abducibles.

In this regard, Aspis et al. [2] [2] have proposed a linear algebraic transformation for abduction by exploiting Sakama et al. [27]'s algebraic transformation. Aspis et al. [2] have defined an explanatory operator based on a third-order tensor for computing abduction in Horn propositional programs that simulates deduction through Clark completion for the abductive program [5]. The dimension explosion would arise, unfortunately, and Aspis et al. [2] have not yet reported an empirical work. Aspis et al. [2] propose encoding every single rule as a slice in a third-order tensor then they achieve the growth naturally. Then, they only consider removing columns that are duplicated or inconsistent with the program. According to our analysis, their current method has some points that can be improved to avoid redundant computation. First, they can consider merging all slices of *And*-rules into a single slice to limit the growth of the output matrix. Second, they have to consider incorporating MHS-based elimination strategy, otherwise, their method will waste a lot of computation and resources on explanations that are not minimal.

Nguyen et al. has proposed partial evaluation for computing least models of definite programs [21]. Their method realizes exponential speed-up of fixpoint computation using a program matrix in computing a long chain of *And*-rules. However, computing the least fixpoint of a definite program is very fast with Sparse Matrix-Vector Multiplication (SpMV) [23]. Therefore, the cost of computing the power of the program matrix may only show benefit in a limited number of specific cases. Further, the possibility of applying partial evaluation for model computation in normal logic programs is remaining unanswered in Nguyen et al. 's work [21].

In terms of partial evaluation, Lamma andMello has demonstrated that Assumption based Truth Maintenance System (ATMS) can be considered as the unfolded version of the logic program following bottom-up reasoning mechanism [18]. Our work, on the other hand, could be considered as a linear algebraic version of top-down partial evaluation for abductive programs. In [26], Sakama and Inoue have proposed *abductive partial deduction* with the purpose to preserve the meanings of abductive logic programs [26]. The main idea of this method is that it retains the original clauses together with the unfolded clauses to reserve intermediate atoms which could be used as assumptions [26]. This idea is incorporated in our method already because the matrix representation simply stores every possible clause by nature.

6 Conclusion

We have proposed to improve the linear algebraic approach for abduction by employing partial evaluation. Partial evaluation steps can be realized as the power of the reduct abductive matrix in the language of linear algebra. Its significant enhancement in terms of execution time has been demonstrated using artificial benchmarks and real FMEA-based datasets with both dense and sparse representation, especially more with the sparse format. The performance gain can be more impressive if there are multiple repeated subgraphs of *And*-rules and even more significant if these subgraphs are deeper and deeper. In this case, the benefit of precomputing these subgraphs outweighs the cost of computing the power of the reduct abductive matrix which is considerably expensive.

However, there are many other issues that need to be resolved in future research to realize the full potential of partial evaluation in abduction. If there is a loop in the program, the current method cannot reach a fixpoint. Handling loops and extending the method to work on non-Horn clausal forms is our ongoing work. As we discussed, it may depend on the possibility to derive consequences of clausal theories in a linear algebraic way. Another challenging problem is knowing when to apply partial evaluation and how deep we do unfolding before solving the problem. Even though repeated partial evaluation finishes in finite steps, it is not necessary to perform until an end concerning the cost of the matrix multiplication. An effective prediction of where to stop without sacrificing too much time can significantly improve the overall performance of the linear algebraic method. Moreover, incorporating some efficient pruning techniques or knowing where to zero out in the abductive matrix is also a potential future topic.

Acknowledgements. This work has been supported by JSPS, KAKENHI Grant Numbers JP18H03288 and JP21H04905, and by JST, CREST Grant Number JPMJCR22D3, Japan.

References

1. Apt, K.R., Bezem, M.: Acyclic programs. New Gener. Comput. **9**, 335–364 (1991). https://doi.org/10.1007/BF03037168
2. Aspis, Y., Broda, K., Russo, A.: Tensor-based abduction in Horn propositional programs. In: ILP 2018, CEUR Workshop Proceedings, vol. 2206, pp. 68–75 (2018)
3. Beckman, L., Haraldson, A., Oskarsson, Ö., Sandewall, E.: A partial evaluator, and its use as a programming tool. Artif. Intell. **7**(4), 319–357 (1976). https://doi.org/10.1016/0004-3702(76)90011-4
4. Boutilier, C., Beche, V.: Abduction as belief revision. Artif. Intell. **77**(1), 43–94 (1995). https://doi.org/10.1016/0004-3702(94)00025-V
5. Console, L., Dupré, D.T., Torasso, P.: On the relationship between abduction and deduction. J. Logic Comput. **1**(5), 661–690 (1991). https://doi.org/10.1093/logcom/1.5.661
6. Dai, W.Z., Xu, Q., Yu, Y., Zhou, Z.H.: Bridging machine learning and logical reasoning by abductive learning. In: Proceedings of the 33rd International Conference on Neural Information Processing Systems, Curran Associates Inc., Red Hook, NY, USA (2019)
7. Eiter, T., Gottlob, G.: The complexity of logic-based abduction. J. ACM (JACM) **42**(1), 3–42 (1995). https://doi.org/10.1145/200836.200838
8. Eshghi, K.: Abductive planning with event calculus. In: ICLP/SLP, pp. 562–579 (1988)

9. Futamura, Y.: Partial evaluation of computation process-an approach to a compiler-compiler. High.-Order Symbolic Comput. **12**(4), 381–391 (1999). https://doi.org/10.1023/A:1010095604496.This is an updated and revised version of the previous publication in "Systems, Computers, Control", Volume 25, 1971, pages 45-50
10. Garey, M.R., Johnson, D.S.: Computers and Intractability: a guide to the theory of NP-completeness. Freeman, W.H. (1979). ISBN 0-7167-1044-7
11. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: ICLP/SLP, vol. 88, pp. 1070–1080 (1988)
12. Heule, M.J., Järvisalo, M., Suda, M.: Sat competition 2018. J. Satisfiability Boolean Model. Comput. **11**(1), 133–154 (2019)
13. Ignatiev, A., Morgado, A., Marques-Silva, J.: Propositional abduction with implicit hitting sets. In: ECAI 2016, Frontiers in Artificial Intelligence and Applications, vol. 285, pp. 1327–1335. IOS Press (2016). https://doi.org/10.3233/978-1-61499-672-9-1327
14. Ignatiev, A., Narodytska, N., Marques-Silva, J.: Abduction-based explanations for machine learning models. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 1511–1519 (2019). https://doi.org/10.1609/aaai.v33i01.33011511
15. Josephson, J.R., Josephson, S.G.: Abductive Inference: Computation, Philosophy, Technology. Cambridge University Press, Cambridge (1996)
16. Koitz-Hristov, R., Wotawa, F.: Applying algorithm selection to abductive diagnostic reasoning. Appl. Intell. **48**(11), 3976–3994 (2018). https://doi.org/10.1007/s10489-018-1171-9
17. Koitz-Hristov, R., Wotawa, F.: Faster horn diagnosis - a performance comparison of abductive reasoning algorithms. Appl. Intell. **50**(5), 1558–1572 (2020). https://doi.org/10.1007/s10489-019-01575-5
18. Lamma, E., Mello, P.: A rationalisation of the ATMS in terms of partial evaluation. In: Lau, K.K., Clement, T.P., (eds) Logic Program Synthesis and Transformation, pp. 118–131. Springer, Cham (1993). https://doi.org/10.1007/978-1-4471-3560-9_9
19. Lloyd, J.W., Shepherdson, J.C.: Partial evaluation in logic programming. J. Logic Program. **11**(3–4), 217–242 (1991). https://doi.org/10.1016/0743-1066(91)90027-M
20. Muggleton, S.: Inductive logic programming. New Gener. Comput. **8**(4), 295–318 (1991). https://doi.org/10.1007/BF03037089
21. Nguyen, H.D., Sakama, C., Sato, T., Inoue, K.: An efficient reasoning method on logic programming using partial evaluation in vector spaces. J. Logic Comput. **31**(5), 1298–1316 (2021). https://doi.org/10.1093/logcom/exab010
22. Nguyen, T.Q., Inoue, K., Sakama, C.: Linear algebraic computation of propositional Horn abduction. In: 2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI), pp. 240–247. IEEE (2021). https://doi.org/10.1109/ICTAI52525.2021.00040
23. Nguyen, T.Q., Inoue, K., Sakama, C.: Enhancing linear algebraic computation of logic programs using sparse representation. New Gener. Comput. **40**(5), 1–30 (2021). https://doi.org/10.1007/s00354-021-00142-2
24. Rocktäschel, T., Riedel, S.: End-to-end differentiable proving. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, pp. 3791–3803, NIPS 2017, Curran Associates Inc., Red Hook, NY, USA (2017). ISBN 9781510860964
25. Saikko, P., Wallner, J.P., Järvisalo, M.: Implicit hitting set algorithms for reasoning beyond NP. In: KR, pp. 104–113 (2016)
26. Sakama, C., Inoue, K.: The effect of partial deduction in abductive reasoning. In: ICLP, pp. 383–397 (1995)
27. Sakama, C., Inoue, K., Sato, T.: Linear algebraic characterization of logic programs. In: Li, G., Ge, Y., Zhang, Z., Jin, Z., Blumenstein, M. (eds.) KSEM 2017. LNCS (LNAI), vol. 10412, pp. 520–533. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63558-3_44
28. Sakama, C., Inoue, K., Sato, T.: Logic programming in tensor spaces. Ann. Math. Artif. Intell. **89**(12), 1133–1153 (2021). https://doi.org/10.1007/s10472-021-09767-x

29. Sato, T.: Embedding Tarskian semantics in vector spaces. In: Workshops at the Thirty-First AAAI Conference on Artificial Intelligence (2017)
30. Sato, T., Inoue, K., Sakama, C.: Abducing relations in continuous spaces. In: IJCAI, pp. 1956–1962 (2018). https://doi.org/10.24963/ijcai.2018/270
31. Schüller, P.: Modeling variations of first-order Horn abduction in answer set programming. Fundam. Informaticae **149**(1–2), 159–207 (2016). https://doi.org/10.3233/FI-2016-1446
32. Selman, B., Levesque, H.J.: Abductive and default reasoning: a computational core. In: AAAI, pp. 343–348 (1990)
33. Tamaki, H., Sato, T.: Unfold/fold transformation of logic programs. In: Proceedings of the Second International Conference on Logic Programming, pp. 127–138 (1984)
34. Vasileiou, S.L., Yeoh, W., Son, T.C., Kumar, A., Cashmore, M., Magazzeni, D.: A logic-based explanation generation framework for classical and hybrid planning problems. J. Artif. Intell. Res. **73**, 1473–1534 (2022). https://doi.org/10.1613/jair.1.13431