Linear Algebraic Computation of Propositional Horn Abduction

Tuan Nguyen Quoc[1,2] and **Katsumi Inoue**[2,1] and **Chiaki Sakama**[3,2]
[1]*The Graduate University for Advanced Studies*, SOKENDAI, Tokyo, Japan
[2]*National Institute of Informatics*, Tokyo, Japan
[3]*Wakayama University*, Wakayama, Japan
Email: {tuannq, inoue}@nii.ac.jp, sakama@wakayama-u.ac.jp

***Abstract*—Linear algebraic characterization of logic programs has been investigated to perform logical inference in large-scale knowledge bases and has gained encouraging results. In this paper, we further extend the linear algebraic characterization in abductive reasoning by exploiting the transpose of the program matrix. Then we propose an efficient exhaustive search strategy, which combines the flexibility and robustness of numerical computation with the compactness and efficiency of set operations, in order to compute solutions of abductive Horn propositional tasks. Experimental results demonstrate that our method is competitive with conflict-driven techniques and has the potential to speed up on parallel computing platforms.**

***Index Terms*—Horn Abduction, Linear Algebra, Sparse Representation**

I. Introduction

Abduction is a form of explanatory reasoning that has been used for Artificial Intelligence (AI) in diagnosis and perception [1] as well as belief revision and automated planning. 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 [2]. Recently, abductive reasoning has gained interests in connecting neural and symbolic reasoning [3] together with explainable AI [4]. Abductive reasoning has been studied intensively in diagnosis and automated reasoning, and several procedures have been proposed in the literature. In the context of consistency-based diagnosis, the Assumption-based Truth Maintenance System (ATMS) has been used extensively [5]. Based on the background theory, ATMS constructs a directed graph in which propositions are represented as nodes, and in each node, ATMS stores all hypotheses allowing to infer this node. Further in [6], de Kleer developes an algorithm that ensures soundness, completeness, minimality, and consistency of every node label. In [7], Reiter has developed an approach via conflicts arising from the manifestation. Reiter exploits the hitting set relation between conflicts and consistency-based diagnoses to operate on a tree structure. In [8], Greiner *et al.* have extended Reiter's idea by utilizing a directed acyclic graph instead of a tree, then they have proposed Hitting Set Directed Acyclic Graph (HS-DAG).

In automated reasoning, Inoue proposed abduction as the search for logical consequences [9], in which explanations are derived deductively, via Skipping Ordered Linear (SOL) resolution. SOLAR (SOL for Advanced Reasoning) is the state-of-the-art implementation of SOL resolution based on the tableaux method [10]. 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 [11]. In [12], 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.* define relations between LP and tensor then propose algorithms for computation of LP models [13]. In [14], Nguyen *et al.* have analyzed the sparsity level of program matrices and proposed to employ sparse representation for scalable computation.

In this work, we consider the possibility to employ such linear algebraic computation for abductive reasoning. Our intention is to see if linear algebraic methods can contribute to the scalability of abduction. If we can see the light in this direction, we can utilize parallel computing based on GPU as well as neural-symbolic computation for robust abductive reasoning. In this regard, Aspis *et al.* have proposed a linear algebraic transformation for abduction by exploiting Sakama *et al.*'s algebraic transformation [15]. They have defined an explanatory operator based on third-order tensor for computing abduction in Horn propositional programs that simulates deduction through Clark completion for abductive programs [16]. The dimension explosion would arise, unfortunately, Aspis *et al.* have not yet reported an empirical work. In this paper, we explore different approaches for linear algebraic abduction.

Contribution & outline: This paper aims at exploring the potentials of linear algebraic computation for the Propositional Horn Clause Abduction Problem (PHCAP) in vector spaces. To this end, we firstly propose the use of the *transpose* of a program matrix that has been defined for deduction in [13; 14] to represent an *abductive matrix* for 1-step abduction in vector spaces. Secondly, we solve the Minimal Hitting Set (MHS) problem to deal with a number of alternative explanations in an efficient way. Thirdly, we employ the sparse representation of abductive matrices for efficient computation. Then, we formally prove the correctness of our method and compare it with other state-of-the-art abductive procedures in large abductive datasets of diagnosis. The rest of this paper is organized as follows: Section II reviews basic notions ; Section III illustrates the linear algebraic computation of abduction; Section IV demonstrates experimental results in Failure Modes and Effects Analysis (FMEA)-base benchmarks; Section V gives final remarks and discusses potential future works.

II. Preliminaries

We consider a language $\mathbb{P}$ 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 $\mathbb{P}$. 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 $(h \leftarrow B_1)$, ..., $(h \leftarrow B_n)$ $(n > 1)$, replace them with a set of new rules:

$$\begin{aligned} & h \leftarrow b_1 \vee \cdots \vee b_n \\ & b_1 \leftarrow B_1 \quad \cdots \quad b_n \leftarrow B_n \end{aligned} \tag{2}$$

where $b_1, \ldots, b_n$ are new atoms such that $b_i \notin B_P$ $(1 \leq i \leq n)$ and $b_i \neq b_j$ if $i \neq j$. 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 rules of the form (2). The resulting Π is called a *standardized program*. Note that the rule (2) is a 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 $head(r) = \emptyset$. A constraint $\leftarrow \ b_1 \wedge \cdots \wedge b_m$ is replaced with

$$\perp \leftarrow \ b_1 \wedge \cdots \wedge b_m$$

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

$$\perp \leftarrow \perp_1 \vee \cdots \vee \perp_n \quad \text{and} \quad \perp_i \leftarrow B_i \ \ (i = 1, \ldots, n)$$

where $\perp_i \notin B_P$ is a new symbol. Given a program P, the set of all propositional variables appearing in P is the *Herbrand base* of P (written B_P). An *interpretation* I $(\subseteq B_P)$ 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 atoms, we write $P \models S$ if $P \models a_1 \wedge \cdots \wedge a_n$. A program P is *consistent* if $P \not\models \perp$.

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

In this paper, we assume a program $\mathbb{T}$ is *acyclic*[1] [17] 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 rule in $\mathbb{T}$. If there exists $h \in \mathbb{H}$ and a rule $r : h \leftarrow body(r) \in \mathbb{T}$, we can replace r with $r' : h \leftarrow body(r) \vee h'$ in $\mathbb{T}$ 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 $\mathbb{T}$ and then replace $body(r)$ by b_r in r'.

Definition 2. Explanation of PHCAP: A set $E \subseteq \mathbb{H}$ is a *solution* of a PHCAP $\langle \mathbb{P}, \mathbb{H}, \mathbb{O}, \mathbb{T} \rangle$ if $\mathbb{T} \cup E \models \mathbb{O}$ and $\mathbb{T} \cup E$ 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 $E' \subset E$.

Deciding if there is a solution of a PHCAP is NP-complete [18; 2]. In this paper, we want to find the set $\mathbb{E}$ of minimal explanations E for a PHCAP $\langle \mathbb{P}, \mathbb{H}, \mathbb{O}, \mathbb{T} \rangle$.

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

III. Linear Algebraic Computation of Abduction

A. Linear algebraic encoding

We slightly modify the definition by Sakama *et al.* to define a matrix program of $\mathbb{T}$ in a vector space.

Definition 3. Matrix representation of standardized programs [13]: Let $\mathbb{T}$ be a standardized program with $\mathbb{P} = \{p_1$, ..., $p_n\}$. Then $\mathbb{T}$ is represented by a *program matrix* $M_P \in \mathbb{R}^{n \times n}$ $(n = |\mathbb{P}|)$ 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 $\mathbb{T}_{And}$ 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 $\mathbb{T}_{Or}$;
3) $a_{ii} = 1$ if $p_i \leftarrow$ is in $\mathbb{T}_{And}$ or $p_i \in \mathbb{H}$.;
4) $a_{ij} = 0$, otherwise.

In Definition 3, we have an update in the condition 3 that we set 1 for all abducible atoms $p_i \in \mathbb{H}$. We further extend Definition 3 to define the abductive matrix of a theory $\mathbb{T}$.

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

Example 1. Consider a PHCAP such that:
$\mathbb{P} = \{p,\ q,\ r,\ s,\ h_1,\ h_2,\ h_3\}$, $\mathbb{O} = \{p\}$, $\mathbb{H} = \{h_1,\ h_2,\ h_3\}$,

[1]A program $\mathbb{T}$ is acyclic if the *dependency graph* of $\mathbb{T}$ is acyclic. The *dependency graph* of a logic program $\mathbb{T}$ is a graph (V, E), where the nodes V are the atoms of $\mathbb{T}$ and, for each rule from $\mathbb{T}$, there are edges in E from the atoms appearing in the body to the atom in the head.

$\mathbb{T} = \{p \leftarrow q \wedge r,\ q \leftarrow h_1 \vee s,\ r \leftarrow s \vee h_2,\ s \leftarrow h_3\}$.
The program matrix and the abductive matrix of $\mathbb{T}$ are [2]:

$$M_P = \begin{array}{cc} & \begin{matrix} p & q & r & s & h_1 & h_2 & h_3 \end{matrix} \\ \begin{matrix} p \\ q \\ r \\ s \\ h_1 \\ h_2 \\ h_3 \end{matrix} & \begin{pmatrix} & 1/2 & 1/2 & & & & \\ & & & 1 & 1 & & \\ & & & 1 & & 1 & \\ & & & & & & 1 \\ & & & & 1 & & \\ & & & & & 1 & \\ & & & & & & 1 \end{pmatrix} \end{array}, \quad {M_P}^T = \begin{array}{cc} & \begin{matrix} p & q & r & s & h_1 & h_2 & h_3 \end{matrix} \\ \begin{matrix} p \\ q \\ r \\ s \\ h_1 \\ h_2 \\ h_3 \end{matrix} & \begin{pmatrix} & & & & & & \\ 1/2 & & & & & & \\ 1/2 & & & & & & \\ & 1 & 1 & & & & \\ & 1 & & & 1 & & \\ & & 1 & & & 1 & \\ & & & 1 & & & 1 \end{pmatrix} \end{array}$$

Definition 5. Correspondent vector of PHCAP: Any subset $s \subseteq \mathbb{P}$ can be represented by a *corresponding vector* v of the length $|\mathbb{P}|$ such that the i-th value $v[i] = 1$ $(1 \leq i \leq |\mathbb{P}|)$ iff the i-th atom p_i of $\mathbb{P}$ is in s; otherwise $v[i] = 0$.

Without ambiguity, we will identify the set representation s with the vector representation v, so we denote them all as v from now on. Henceforth, v_i is the i-th atom of $\mathbb{P}$ that constitutes s, while $v[i]$ is the value of the vector at index i.

In some specific cases, we also use v as a special function that outputs a corresponding vector of a subset in vector spaces: $v(\mathbb{O})$ the observation vector, $v(\mathbb{H})$ the hypotheses vector, $v(\bot)$ the integrity vector (shorthand of $v(\{\bot\})$ where $\{\bot\} \subset \mathbb{P}$), $v(head(\mathbb{T}_{And}))$ the vector of all head atoms of And-rules in $\mathbb{T}_{And}$, $v(head(\mathbb{T}_{Or}))$ the vector of all head atoms of Or-rules in $\mathbb{T}_{Or}$. We use this notation for better indexing each element and a vector value in the set/vector. If there is no need to indicate each individual item, we can omit the function notation $v()$.

In order to utilize the use of correspondent vectors, we define a thresholding method to perform needed set operations in vector spaces.

Definition 6. θ-thresholding:

1) Given a value $x \in \mathbb{R}$, define $\theta(x) = x'$ such that $x' = 1$ if $x > 0$; otherwise, $x' = 0$
2) Given a vector $v \in \mathbb{R}^n$, define $\theta(v) = v'$ such that $v'[i] = 1$ if $v[i] > 0$; otherwise $v'[i] = 0$
3) Given a matrix $M \in \mathbb{R}^{n \times m}$, define $\theta(M) = M'$ such that $M'[i][j] = 1$ if $M[i][j] > 0$; otherwise $M'[i][j] = 0$

where $1 \leq i \leq n,\ 1 \leq j \leq m$.

Proposition 1. The following equivalence relations hold :

$$u \cap v = \emptyset \Leftrightarrow u \cdot v = 0$$
$$u \cap v \neq \emptyset \Leftrightarrow u \cdot v > 0$$
$$u \subseteq v \Leftrightarrow \theta(u + v) \leq \theta(v)$$

where $\cdot$ is the inner product.

B. Linear algebraic computation

The goal of PHCAP is to find the set of minimal explanations $\mathbb{E}$ according to Definition 2. Using Definition 5, we can represent any $E \in \mathbb{E}$ by a column vector $E \in \mathbb{R}^{|\mathbb{P}| \times 1}$. To compute E, we define an *interpretation vector* $v \in \mathbb{R}^{|\mathbb{P}| \times 1}$. We use the interpretation 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 interpretation vector v as: in order to explain $\mathbb{O}$, we have to explain all atoms v_i such that $v[i] > 0$.

Definition 7. Explanation vector: The interpretation vector v *reaches* an explanation 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 short hand of $v(\mathbb{H})$ which is the hypotheses set/vector mentioned above.

Proposition 2. An interpretation vector v is *consistent* with $\mathbb{T}$ if *lfp*$(M_P, v) \cap \{\bot\} \neq \emptyset$. This condition can be written in linear algebra as follows:

$$v(\bot) \cdot \mathit{lfp}(M_P, v) = 0 \tag{4}$$

where M_P is the program matrix of $\mathbb{T}$ and *lfp*(M_P, v) is the vector representation of the least fixpoint of the T_P-operator [19] starting from v.

Proof. The *lfp* can be computed in the vector space by applying matrix multiplication $M_P \cdot v$ continuously until the fixpoint is reached. The resulting vector corresponds to the least model of $\mathbb{T} \cup v$ [13]. If this model contains $\bot$, $\mathbb{T} \cup v$ is inconsistent; otherwise v is consistent with $\mathbb{T}$. We can perform this test using Definition 1. $\square$

An efficient method to compute *lfp* of a definite program has been developed in [14].

We now define 1-step abduction in PHCAP step by step. We use the superscript (t) to denote the interpretation vector v at a step t.

Definition 8. 1-step abduction for $\mathbb{T}_{And}$ of a vector: We can obtain a *reduct abductive matrix* $M_P(\mathbb{T}_{And})^T$ from ${M_P}^T$ by removing all columns *w.r.t.* Or-rules in $\mathbb{T}_{Or}$. Then we define the *1-step abduction for* $\mathbb{T}_{And}$ as:

$$v^{(t+1)} = M_P(\mathbb{T}_{And})^T \cdot v^{(t)} \tag{5}$$

The 1-step abduction (5) is a reverse version of the T_P-operator on a single vector. By transposing the program matrix to an abductive matrix, we represent the abductive step in a vector space that computes the explanation $v^{(t+1)}$ for $v^{(t)}$. This step corresponds to a deductive step through Clark completion in an SD program [16]. Suppose that there is an index i such that $v_i \in v^{(t)} \cap head(\mathbb{T}_{And})$, according to Definition 3 and Definition 4 there is a column *w.r.t.* v_i in $M_P(\mathbb{T}_{And})^T$, denoted by $col(v_i)$. By applying (5), $v^{(t+1)}[j] = \dfrac{v^{(t)}[i]}{|col(v_i)|} > 0$, for any j such that $v_j^{(t+1)} \in col(v_i)$. Then vector $v^{(t+1)}$ represents the set of atoms required to explain $v^{(t)}$.

Example 2 (cont. Example 1)**.** $\mathbb{T}_{And} = \{p \leftarrow q \wedge r,\ s \leftarrow h_3\}$. We can obtain a reduct abductive matrix $M_P(\mathbb{T}_{And})^T$ by removing columns *w.r.t.* rules $\{q \leftarrow h_1 \vee s,\ r \leftarrow s \vee h_2\}$ in the

[2] We omit all zero elements in matrices for better readability.

original abductive matrix. Consider applying 1-step abduction for $\mathbb{T}_{And}$ with $v^{(t)} = \mathbb{O}$:

$$v^{(t)} = (1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0)^T\ (= \mathbb{O})$$
$$v^{(t+1)} = M_P(\mathbb{T}_{And})^T \cdot v^{(t)}$$
$$= \begin{array}{c} \\ p \\ q \\ r \\ s \\ h_1 \\ h_2 \\ h_3 \end{array} \!\!\begin{array}{c} \begin{array}{ccccccc} p & q & r & s & h_1 & h_2 & h_3 \end{array} \\ \begin{pmatrix} & & & & & & \\ 1/2 & & & & & & \\ 1/2 & & & & & & \\ & & & & & & \\ & & & & 1 & & \\ & & & & & 1 & \\ & & & 1 & & & 1 \end{pmatrix} \end{array} \cdot \begin{array}{c} p \\ q \\ r \\ s \\ h_1 \\ h_2 \\ h_3 \end{array} \!\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{array}{c} p \\ q \\ r \\ s \\ h_1 \\ h_2 \\ h_3 \end{array} \!\begin{pmatrix} 0 \\ 1/2 \\ 1/2 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

The vector $v^{(t+1)}$ can be interpreted as: in order to explain p, both q and r are to be explained.

Definition 8 illustrates that we can apply continuously the 1-step abduction (5) with $v^{(0)} = \mathbb{O}$ until it reaches an explanation by the condition in Definition 7 and satisfies consistency in Prop. 2. In fact, Definition 7 may not hold in case where there is an atom in the interpretation vector that we have no rule in $\mathbb{T}_{And}$ to apply to find its explanation.

Proposition 3. The summation of $v^{(t)}$ is bounded.

$$sum(v^{(t+1)}) \leq sum(v^{(t)}) \leq \cdots \leq sum(v^{(0)}) \tag{6}$$

where $sum(v) = \Sigma_{v_i \in v} v[i]$.

This proposition is trivial to prove using Definition 3 and Definition 4. For simplicity, we can initialize the starting point $v^{(0)}$ that statisfies $sum(v^{(0)}) = 1$. If there are multiple observations $o_1, o_2, \ldots, o_k \in \mathbb{O}$, a new atom o is introduced to replace the current observation set. Then a new conjunctive rule $o \leftarrow o_1 \wedge o_2 \wedge \cdots \wedge o_k$ is introduced to the theory $\mathbb{T}$. Then we can initialize the starting point $\mathbb{O} = \{o\}$ such that summation of the corresponding vector is 1. From now on, we assume $sum(v^{(0)}) = 1$ without loss of generality.

Proposition 4. If $sum(v^{(t)}) < 1$, then $v^{(t)} \cup \mathbb{T}_{And} \not\models \mathbb{O}$.

Proof. According to Prop. 3, for any interpretation $v^{(t)}$, we have $sum(v^{(t)}) \leq sum(v^{(t-1)}) \leq \cdots \leq sum(v^{(0)}) = 1$. Assume the equality holds until the step $t-1$ of the 1-step abduction (5). If there is any index i such that $v_i \in v^{(t-1)} \setminus head(\mathbb{T}_{And})$, the column *w.r.t.* v_i in the reduct abductive matrix is encoded as a zero column. Thus, when applying matrix multiplication in Definition 8, at the index i, $v^{(t)}[i] = 0$ while $v^{(t-1)}[i] > 0$. That is: $sum(v^{(t-1)}) - sum(v^{(t)}) \geq v^{(t-1)}[i] > 0 \Leftrightarrow sum(v^{(t)}) < 1$. This behavior is equivalent to considering an explanation of v_i but there is no rule in $\mathbb{T}_{And}$ that can explain v_i. □

According to Definition 5, an interpretation v can be represented by a column vector $v \in \mathbb{R}^{|\mathbb{P}| \times 1}$. We can stack multiple vectors v to form an interpretation matrix $M \in \mathbb{R}^{|\mathbb{P}| \times |M|}$ while all definitions and propositions with the 1-step abduction for $\mathbb{T}_{And}$ of a vector still work. Therefore, we can rewrite Definition 8 as follows:

Definition 9. 1-step abduction for $\mathbb{T}_{And}$:

$$M^{(t+1)} = M_P(\mathbb{T}_{And})^T \cdot M^{(t)} \tag{7}$$

We now introduce a notation M as a matrix that is equivalent to a vector of vectors or a set of sets. Note that we denote $|M|$ as the number of vectors or sets in M. We also use the same notation we mentioned above that M_i is the i-th set of M, while $M[i]$ is the vector at an index i.

Let v be an interpretation vector in $\langle \mathbb{P}, \mathbb{H}, \mathbb{O}, \mathbb{T} \rangle$ such that $v \cap head(\mathbb{T}_{Or}) = \{head(r_1), head(r_2), \ldots, head(r_k)\}$ with $r_1, r_2, \ldots, r_k \in \mathbb{T}_{Or}$. In order to compute explanations of v we have to explore all combinations c extracted from $\{body(r_1), body(r_2), \ldots, body(r_k)\}$ such that $\forall j \in \{1, 2, \ldots, k\}$, $c \cap body(r_j) \neq \emptyset$. It turns out that this is equivalent to enumerate the Minimal Hitting Sets (MHS) with the input set is $\{body(r_1), body(r_2), \ldots, body(r_k)\}$ [20].

We denote **MHS**$(\mathbb{S})$ as all MHS of a family of sets to be hit $\mathbb{S}$. Now we can define 1-step abduction for $\mathbb{T}_{Or}$.

Definition 10. 1-step abduction for $\mathbb{T}_{Or}$:

$$M^{(t+1)} = \bigcup_{\forall v \in M^{(t)}} \ \bigcup_{\forall s \in \mathbf{MHS}(\mathbb{S}_{(v,\ \mathbb{T}_{Or})})} \Big(\big(v \setminus head(\mathbb{T}_{Or}) \big) \cup s \Big) \tag{8}$$

where: $\mathbb{S}_{(v,\ \mathbb{T}_{Or})} = \{body(r_1), body(r_2), \ldots, body(r_k)\}$ is a family of sets to be hit such that $v \cap head(\mathbb{T}_{Or}) = \{head(r_1), head(r_2), \ldots, head(r_k)\}$.

Note that all new vectors $v \in M^{(t+1)}$ will be reallocated values such that $sum(v) = 1$ to maintain the condition in Prop. 4 of the 1-step abduction (7) for $\mathbb{T}_{And}$.

Example 3 (cont. Example 2)**.** $\mathbb{T}_{or} = \{q \leftarrow h_1 \vee s,\ r \leftarrow s \vee h_2\}$. We use the output of Example 2 as the input of the 1-step abduction for $\mathbb{T}_{Or}$, but now we treat it as a matrix instead:

$$M^{(t)^T} = \begin{array}{c} \\ 0 \end{array} \!\begin{array}{c} \begin{array}{ccccccc} p & q & r & s & h_1 & h_2 & h_3 \end{array} \\ \begin{pmatrix} 0 & 1/2 & 1/2 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$
$$M^{(t)} = \{\{q, r\}\}$$
$$\mathbb{S}_{(M_0^{(t)},\ \mathbb{T}_{Or})} = \{\{h_1,\ s\},\ \{s,\ h_2\}\}$$
$$\mathbf{MHS}(\mathbb{S}_{(M_0^{(t)},\ \mathbb{T}_{Or})}) = \{\{s\},\ \{h_1,\ h_2\}\}$$
$$M^{(t+1)} = \{\{s\},\ \{h_1,\ h_2\}\}$$
$$M^{(t+1)^T} = \begin{array}{c} \\ 0 \\ 1 \end{array} \!\begin{array}{c} \begin{array}{ccccccc} p & q & r & s & h_1 & h_2 & h_3 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 & 0 \end{pmatrix} \end{array}$$

To the best of our knowledge, it is not trivial to implement an efficient method in a vector space that enumerates exactly all MHS as we defined in Definition 10. Hence, to implement (8) at this time, we have no choice but to treat all interpretations as sets instead of vectors. Fortunately, we can perform the vector-set conversion with minimal cost using the sparse representation we are going to discuss later.

Up to now, we have defined 1-step abduction for $\mathbb{T}_{And}$ and $\mathbb{T}_{Or}$. Although each method itself is not sufficient to solve the PHCAP $\langle \mathbb{P}, \mathbb{H}, \mathbb{O}, \mathbb{T} \rangle$, their characteristics are important for us to define a general approach.

Definition 11. Or**-computable and** And**-computable:**

1) A vector v is Or-computable iff $v \cap head(\mathbb{T}_{Or}) \neq \emptyset$.
2) A matrix M is Or-computable iff $\exists v \in M$, v is Or-computable.
3) A vector v is And-computable iff v is not Or-computable.
4) A matrix M is And-computable iff $\forall v \in M$, v is not Or-computable.

Proposition 5. For any matrix M which is Or-computable in $\langle \mathbb{P}, \mathbb{H}, \mathbb{O}, \mathbb{T} \rangle$, there exists a fixpoint t of (8), such that $M^{(t+k)} = M^{(t)}$, $\forall k > 0,\ k \in \mathbb{N}$.

Proof. For each Or-computable vector $v \in M$, the 1-step abduction (8) replaces all atoms in the intersection of v and $head(\mathbb{T}_{Or})$ by the corresponding MHS. In addition, $\mathbb{T}$ is finite and acyclic so there is a fixpoint such that there is no Or-rule that can be used to abduce v or we can say that v is And-computable. That means $v \cap head(\mathbb{T}_{Or}) = \emptyset$, so the corresponding MHS is an empty set then $\forall k > 0$, $v^{(t+k)} = v^{(t)}$ ($k \in \mathbb{N}$). Extend this to other interpretations in M we have that M is And-computable and $\forall k > 0$, $M^{(t+k)} = M^{(t)}$ ($k \in \mathbb{N}$). ☐

Corollary 1. For any matrix M which is Or-computable in $\langle \mathbb{P}, \mathbb{H}, \mathbb{O}, \mathbb{T} \rangle$, if t is the fixpoint of (8) then $M^{(t)}$ is And-computable in $\langle \mathbb{P}, \mathbb{H}, \mathbb{O}, \mathbb{T} \rangle$.

Proposition 6. For any matrix M which is And-computable in $\langle \mathbb{P}, \mathbb{H}, \mathbb{O}, \mathbb{T} \rangle$, $M_P(\mathbb{T}_{And})^T \cdot M = {M_P}^T \cdot M$.

Proof. As in Definition 8, $M_P(\mathbb{T}_{And})^T$ is a reduct abductive matrix from ${M_P}^T$ by removing all columns *w.r.t.* Or-rules in $\mathbb{T}_{Or}$. So $M_P(\mathbb{T}_{And})^T \cdot M$ has no effect on Or-computable vector $v \in M$. Moreover, M is And-computable in $\langle \mathbb{P}, \mathbb{H}, \mathbb{O}, \mathbb{T} \rangle$ by definition, therefore $M_P(\mathbb{T}_{And})^T \cdot M = {M_P}^T \cdot M$. ☐

Based on the two 1-step abduction (7) and (8), we propose an exhaustive search strategy to solve the PHCAP $\langle \mathbb{P}, \mathbb{H}, \mathbb{O}, \mathbb{T} \rangle$ in a vector space as illustrated in Algorithm 1.

Some explanations are in order:

- Step 7: $sum_{col}(M')$ means applying summation on each vector $v \in M'$ to return a vector. Then we compare each element of this vector with $1 - \epsilon$ following the Prop. 4 to return a corresponding Boolean vector. Due to the numerical issue with floating-point numbers in computer *e.g.* $\frac{1}{3} + \frac{1}{3} + \frac{1}{3} = 0.999\ldots$, a small fraction ϵ is introduced to relax the condition in Prop. 4. Choosing the best ϵ depends on actual $\langle \mathbb{P}, \mathbb{H}, \mathbb{O}, \mathbb{T} \rangle$. If we set ϵ too small, we may filt out good interpretations and the algorithm might not gives expected output. While setting ϵ too large, we may waste of computation in unexplainable paths.
- Step 8: We use the Boolean vector in Step 7 to eliminate unexplainable interpretations. We keep only vectors that their Boolean value is **False**. [] is the projection method that extracts from M' only vectors that satisfy the condition inside []. Similarly, we also use the projection method in Steps 15-18.

Algorithm 1 Explanations finding in a vector space

Input: PHCAP consists of a tuple $\langle \mathbb{P}, \mathbb{H}, \mathbb{O}, \mathbb{T} \rangle$
Output: Set of explanations $\mathbb{E}$

1: Create an abductive matrix ${M_P}^T$ from $\mathbb{T}$
2: Initialize the observation matrix M from $\mathbb{O}$
3: $\mathbb{E} = \emptyset$
4: **while True do**
5:     $M' = {M_P}^T \cdot M$
6:     $M' =$ **consistent**(M') ▷ Prop. 2
7:     $v\_sum = sum_{col}(M') < 1 - \epsilon$ ▷ Prop. 4
8:     $M' = M'[v\_sum =$ **False**$]$
9:     **if** $M' = M$ **or** $M' = \emptyset$ **then**
10:         $v\_ans = \theta(M + \mathbb{H}) \leq \theta(\mathbb{H})$ ▷ Definition 7
11:         $\mathbb{E} = \mathbb{E} \cup M[v\_ans =$ **True**$]$
12:         **return minimal**$(\mathbb{E})$ ▷ Minimality check
13:     **do**
14:         $v\_ans = \theta(M' + \mathbb{H}) \leq \theta(\mathbb{H})$ ▷ Definition 7
15:         $\mathbb{E} = \mathbb{E} \cup M'[v\_ans =$ **True**$]$
16:         $M' = M'[v\_ans =$ **False**$]$
17:         $M = M \cup M'[$**not** Or-computable$]$
18:         $M' = M'[Or$-computable$]$
19:         $M' = \bigcup_{\forall v \in M'} \bigcup_{\forall s \in \mathbf{MHS}(\mathbb{S}_{(v,\,\mathbb{T}_{Or})})} \Big(\big(v \setminus head(\mathbb{T}_{Or})\big) \cup s \Big)$
20:         $M' =$ **consistent**(M') ▷ Prop. 2
21:     **while** $M' \neq \emptyset$

- Step 12: Applying the minimality check on the set $\mathbb{E}$ to eliminate redundant explanations according to Definition 1. We implement this method by sorting all $E \in \mathbb{E}$ by their cardinality, then applying a simple set iteration loop.
- Steps 16,18-19: Construct a matrix M' which is Or-computable then perform the 1-step abduction (8). Here we have to solve the MHS problem many times. We implement a naive approach in which we enumerate all combinations then apply the minimality check similar to Step 12. However, this implementation can deal with up to 500,000 combinations, therefore, we exploit PySAT[3] to solve large-size MHS problems [21].

Theorem 1. The output of Algorithm 1 is the set of all minimal explanations of the PHCAP $\langle \mathbb{P}, \mathbb{H}, \mathbb{O}, \mathbb{T} \rangle$.

Proof. Definition 5 defines 1-1 correspondence between subsets of $\mathbb{P}$ and vectors. Algorithm 1 employs both the 1-step abduction (7) and (8) in a vector space, which are equivalent to abductive steps in $\mathbb{T}_{And}$ and $\mathbb{T}_{Or}$ respectively, exploring all possibilities that satisfy both Definition 7 and Prop. 2. Therefore, $\forall E \in \mathbb{E}$, $E \subseteq \mathbb{H}$ we have $E \cup \mathbb{T} \vDash \mathbb{O}$ and $E \cup \mathbb{T} \nvDash \bot$. Futher, Algorithm 1 employs minimality check on $\mathbb{E}$, therefore $\forall E_1, E_2 \in \mathbb{E}$, $E_1 \nsubseteq E_2$. ☐

Example 4. Let us demonstrate how to solve the PHCAP in Example 1 using Algorithm 1. Actually, we have done the first iteration of Algorithm 1 as illustrated in Example 2

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

and Example 3. We continue the next iteration with the interpretation matrix $M = M^{(t+1)}$ obtained in Example 3.

$$M^T = \begin{array}{c c} & \begin{array}{ccccccc} p & q & r & s & h_1 & h_2 & h_3 \end{array} \\ \begin{array}{c} 0 \\ 1 \end{array} & \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 & 0 \end{pmatrix} \end{array}$$

$$M'^T = \left(M_P{}^T \cdot M\right)^T = \begin{array}{c c} & \begin{array}{ccccccc} p & q & r & s & h_1 & h_2 & h_3 \end{array} \\ \begin{array}{c} 0 \\ 1 \end{array} & \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 & 0 \end{pmatrix} \end{array}$$

Here Algorithm 1 stops because all interpretations reach explanations of Definition 7, satisfying the condition of Prop. 2, and $M' = \emptyset$ after that. Finally, the algorithm applies minimal checking and gives the output set of minimal explanations $\mathbb{E} = \{\{h_3\}, \{h_1, h_2\}\}$.

C. Matrix representation

In our previous work, we have analyzed the sparsity of logic programs in vector spaces and have a conclusion that program matrices are sparse in general [14]. The paper indicates that implementing the T_P-operator using a sparse format outperforms that using the dense format in large-scale logic programs. Similarly, the sparse representation will be promising in abductive reasoning.

The sparsity of a matrix equals the number of zero-valued elements divided by the total number of elements [22]. By definition, there is no doubt that in a PHCAP $\langle \mathbb{P}, \mathbb{H}, \mathbb{O}, \mathbb{T} \rangle$, the sparsity of the abductive matrix and that of the program matrix are equal and can be computed by the following equation [14]:

$$sparsity(\mathbb{T}) = 1 - \frac{\sum_{r \in \mathbb{T}} |body(r)|}{|\mathbb{P}|^2} \tag{9}$$

Extend the definition of sparsity to an interpretation matrix M, we have the following equation:

$$sparsity(M) = 1 - \frac{\sum_{v \in M} |v|}{|\mathbb{P}| \times |M|} \tag{10}$$

Because M is growing while we explore different possible explanations, there is no warranty that M always has a high level of sparsity. In case M is not sparse ($sparsity(M) \leq 0.9$), although the sparse representation may not help much in terms of performance, it provides faster vector-set conversion. In Section IV, we will analyze more detail about the sparsity level of interpretation matrices.

Regarding memory usage among general-purpose sparse representations, the Compressed Sparse Row (CSR) and the Compressed Sparse Column (CSC) formats are similar in case the matrix is square and they are usually better than the Coordinate (COO) format. The CSR format enables faster lookup by row while the CSC format provides faster lookup by column. In our previous work, we suggest using the CSR for the program matrix then when transpose it to obtain an abductive matrix, it will become a CSC matrix naturally. Therefore, we suggest using the CSC format for both the abductive matrix and the interpretation matrix.

IV. Experimental Results

A. Experimental setup

To demonstrate the linear algebraic computation of abduction, we conduct experiments on the benchmark datasets used in [23; 24][4]. The purpose of this paper is to compare the effectiveness of our method with that of other general-purpose solvers. Thus, we implement our method as two versions including a dense matrix method (*Dense matrix* for short) and a sparse matrix method (*Sparse matrix* for short). For both the abductive matrix and the interpretation matrix, we use dense format in *Dense matrix* while in *Sparse matrix*, CSC format is used. Our code is implemented in Python 3.7 using Numpy and Scipy for matrices representation and computation. As stated in Algorithm 1, we implement a naive approach for solving MHS that we only use built-in Python set operations. For large-size MHS problems, which have more than 50,000 combinations, we use MHS enumerator provided by PySAT. Importantly, we force to execute our code in a single core, in order to make a fair comparison with other methods. The computer we perform this experiment has the following configurations: CPU: Intel(R) Xeon(R) Bronze 3106 CPU @1.70GHz; RAM: 64GB DDR3 @1333MHz; Operating system: Ubuntu 18.04 LTS 64bit.

As stated in Section II, we need an extra step to transform the program into equivalent standardized format. Accordingly, we denote the input PHCAP as $\langle \mathbb{P}', \mathbb{H}, \mathbb{O}, \mathbb{T}' \rangle$ while the standardized PHCAP is $\langle \mathbb{P}, \mathbb{H}, \mathbb{O}, \mathbb{T} \rangle$. The detailed statistical information of all problem sets are represented in the first 4 rows $|\mathbb{H}|$, $|\mathbb{P}' \setminus \mathbb{H}|$, $|\mathbb{T}'|$, and $|\mathbb{O}|$ of Table I. The next 4 rows of Table I represent the transformed $\mathbb{T}$ and $\mathbb{P}$, while $|\mathbb{P}|$ is also the dimension of a corresponding abductive matrix.

Table I also records the sparsity analysis data on all benchmark datasets. $\eta_z(M_P{}^T)$ and $\eta_z(M)$ are the number of non-zero elements in the abductive matrix and the interpretation matrix respectively. Similarly, $sparsity(M_P{}^T)$ and $sparsity(M)$ are the sparsity of the abductive matrix and the interpretation matrix, respectively. Because interpretation matrices are not fixed, we record only the maximum number of interpretations ($max(|M|)$), the maximum η_z ($max(\eta_z(M))$), and the minimum sparsity $min(sparsity(M))$ for each interpretation matrix. Finally, max_iter is the number of iterations of the main loop in the Algorithm 1 and $|\mathbb{E}|$ is the number of correct minimal explanations.

B. Results

1) Artificial benchmarks: Figure 1 and Figure 2 illustrate the comparison on the Artificial samples I and II, while Table II and Table III give more detail information. As witnessed in Figure 1 and Figure 2, runtime trends of all algorithms grow exponentially by the number of solved samples (**#solved**).

In the Artificial samples I, together with $ATMS$ and $HS\text{-}DAG_{QX}$, our linear algebraic approaches are able to solve all problems. Surprisingly, in terms of total runtime, *Dense matrix* is even faster than $HS\text{-}DAG_{QX}$ while *Sparse matrix* is just

[4] Consult the paper for more detail about other algorithms.

Benchmark dataset	Artificial samples I (166 problems)				Artificial samples II (118 problems)				FMEA samples (213 problems)			
Parameters	mean	std	min	max	mean	std	min	max	mean	std	min	max
$\lvert\mathbb{H}\rvert$	275.07	167.12	10.00	504.00	120.42	74.35	12.00	235.00	26.16	20.81	3.00	90.00
$\lvert\mathbb{P}' \setminus \mathbb{H}\rvert$	1903.23	1504.90	6.00	6466.00	252.74	220.50	13.00	1055.00	27.58	19.32	6.00	84.00
$\lvert\mathbb{T}'\rvert$	2951.10	2131.57	11.00	7187.00	417.70	320.56	21.00	1147.00	71.59	75.88	13.00	299.00
$\lvert\mathbb{O}\rvert$	2.86	1.38	1.00	5.00	2.72	1.71	1.00	13.00	10.79	6.94	1.00	29.00
$\lvert\mathbb{T}\rvert$	2088.32	1584.48	11.00	6601.00	321.86	252.64	18.00	1110.00	27.58	19.32	6.00	84.00
$\lvert\mathbb{T}_{And}\rvert$	1188.63	1349.59	8.00	6375.00	201.86	186.64	9.00	1007.00	16.10	9.23	1.00	43.00
$\lvert\mathbb{T}_{Or}\rvert$	899.69	839.58	3.00	3345.00	119.99	107.40	4.00	437.00	11.48	11.01	1.00	41.00
$\lvert\mathbb{P}\rvert$	2372.36	1730.91	24.00	7148.00	450.89	318.33	38.00	1397.00	53.74	39.59	9.00	174.00
$\eta_z(M_P{}^T)$	6354.90	4902.87	50.00	22,307.00	1180.36	861.83	83.00	4117.00	107.54	98.57	18.00	413.00
$sparsity(M_P{}^T)$	0.99	0.02	0.90	1.00	0.99	0.01	0.90	1.00	0.95	0.04	0.73	0.99
$max(\lvert M\rvert)$	250.34	1729.52	1.00	16,866.00	16,494.04	149,787.13	1.00	1,618,050.00	2126.49	15,512.54	1.00	154,440.00
$max(\eta_z(M))$	5138.28	37,776.87	1.00	428,754.00	390,900.36	3,240,888.43	1.00	34,882,765.00	43,738.87	334,393.40	1.00	3,459,456.00
$min(sparsity(M))$	0.98	0.05	0.68	1.00	0.94	0.08	0.59	1.00	0.79	0.13	0.46	0.99
max_iter	4.63	5.36	2.00	65.00	6.56	8.56	2.00	58.00	1.94	0.24	1.00	2.00
$\lvert\mathbb{E}\rvert$	2.77	5.06	1.00	50.00	499.60	5386.87	1.00	58,520.00	68.89	272.54	1.00	2288.00

TABLE I: Statistics and sparsity analysis on benchmark datasets

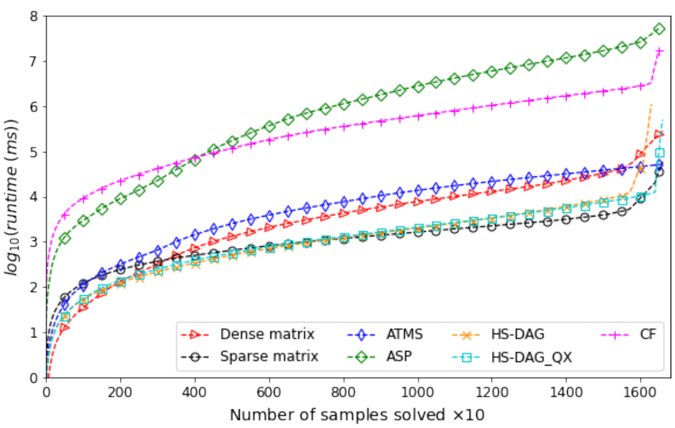

Fig. 1: Experimental results for the Artificial samples I.

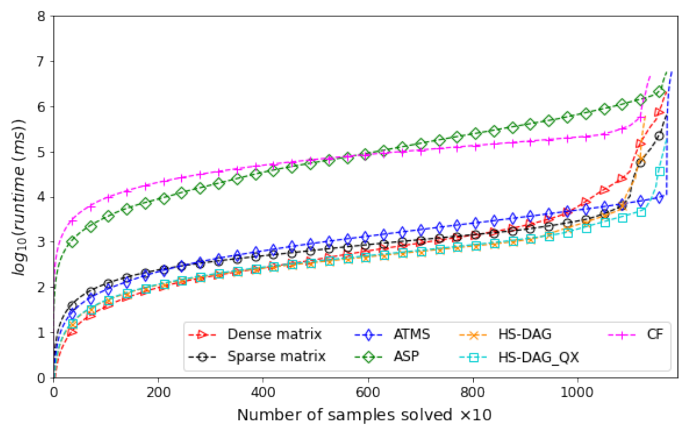

Fig. 2: Experimental results for the Artificial samples II.

Algorithms	#solved	#fastest	mean(t) (ms)	std(t) (ms)	mean($t+t_p$) (ms)	std($t+t_p$) (ms)
Dense matrix	1660.00	110.00	27,902.34	334.72	27,902.34	1743.57
Sparse matrix	1660.00	930.00	5899.76	45.19	5899.76	95.03
ATMS	1660.00	68.00	5170.98	153.63	5170.98	1088.59
ASP	1650.00	0.00	5,323,586.50	103,733.75	7,723,586.50	317,719.00
HS-DAG	1630.00	344.00	110,639.62	30,280.36	7,310,639.62	33,034.62
HS-DAG_QX	1660.00	208.00	50,229.78	9765.79	50,229.78	12,389.13
CF	1650.00	0.00	1,673,516.00	59,781.49	4,073,516.00	91,042.34

TABLE II: Detail runtime results on the Artificial samples I.

Algorithms	#solved	#fastest	mean(t) (ms)	std(t) (ms)	mean($t+t_p$) (ms)	std($t+t_p$) (ms)
Dense matrix	1170.00	248.00	207,014.78	3572.90	2,607,014.78	6905.92
Sparse matrix	1170.00	120.00	63,251.15	234.91	2,463,251.15	595.52
ATMS	1180.00	119.00	598,145.22	63,316.83	598,145.22	67,145.68
ASP	1170.00	0.00	568,407.50	2868.16	2,968,407.50	12,195.99
HS-DAG	1130.00	436.00	67,567.05	16,942.54	12,067,567.05	18,572.30
HS-DAG_QX	1170.00	257.00	18,198.16	4106.36	2,418,198.16	6744.99
CF	1140.00	0.00	508,309.00	7849.55	10,108,309.00	13,188.27

TABLE III: Detail runtime results on the Artificial samples II.

a few seconds behind the fastest - $ATMS$. Other methods fall behind by a large margin because they are penalized on unresolved samples. Table II further reveals the efficiency of linear algebraic methods that *Dense matrix* is the fastest in 110 runs while *Sparse matrix* is the fastest in 930 runs. In this dataset, the sparsity of abductive matrices and interpretation matrices maintains at a good level of **mean** (Table I).

In the Artificial samples II, only $ATMS$ is able to handle all problems although it is not the fastest algorithm. ASP, $HS\text{-}DAG_{QX}$ and linear algebraic methods are equal in terms of **#solved** that is 117/118. Table III gives a more detailed comparison in the Artificial samples II that *Dense matrix* and *Sparse matrix* are competitive as being the fastest algorithm in 248 and 120 runs, respectively. From Table I we also can see that $|\mathbb{E}|$ and $max(|M|)$ surge to very large figures, $58,520$ and $1,618,050$, respectively. This happens in the only one problem instance that our methods are failed to solve in time.

Notably in both the benchmarks, *Dense matrix* takes the lead over *Sparse matrix* in the beginning. This is understandable because the sparsity level of interpretation matrices, for example in the data for Artificial samples II in Table I, drops to **min** 0.59 and **mean** 0.94. In this situation, sparse representation cannot take much benefit. Due to that fact, *Sparse matrix* still takes over *Dense matrix* in the end with much better total execution time as can be seen in Figure 1. Further, *Sparse matrix* is the most stable algorithm with the best **std**.

2) Real-world samples: Figure 3 illustrates the comparison on FMEA samples benchmark while Table IV gives more detail information about each algorithm. In this benchmark, $ATMS$, CF and linear algebraic methods are able to solve all instances without penalty. Surprisingly, *Dense matrix* outperforms others and takes the lead by a remarkable margin (Figure 3) and ends up even more than 2 times faster than the 3rd place algorithm - $ATMS$ in terms of total execution

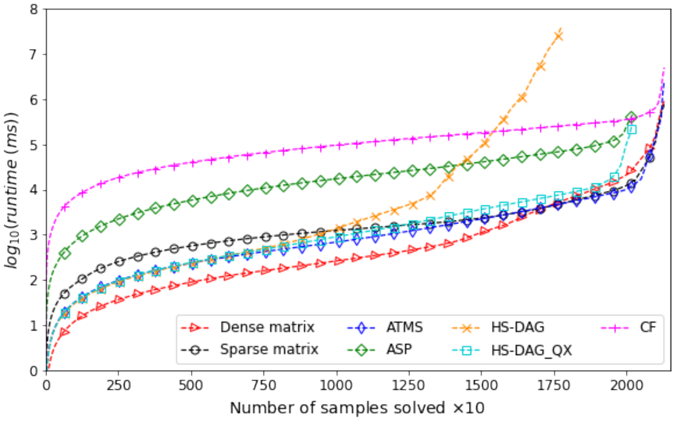

Fig. 3: Experimental results for the FMEA diagnosis problems.

Algorithms	**#solved**	**#fastest**	**mean**(t) (ms)	**std**(t) (ms)	**mean**$(t + t_p)$ (ms)	**std**$(t + t_p)$ (ms)
Dense matrix	2130.00	1166.00	92,984.70	1548.40	92,984.70	3558.29
Sparse matrix	2130.00	160.00	77,685.41	886.43	77,685.41	1424.03
$ATMS$	2130.00	579.00	250,000.13	16,202.02	250,000.13	23,799.18
ASP	2020.00	0.00	45,051.30	762.81	26,445,051.30	2798.81
$HS\text{-}DAG$	1775.00	31.00	3,883,275.76	650,599.24	89,083,275.76	950,627.02
$HS\text{-}DAG\_QX$	2020.00	184.00	27,926.05	350.21	26,427,926.05	1491.80
CF	2130.00	10.00	498,885.00	16,324.68	498,885.00	25,601.16

TABLE IV: Detail runtime results on the FMEA diagnosis problems..

time (Figure 3). *Sparse matrix* starts with a humble beginning but performs very well after that and finishes at the first place with the lowest execution time in total.

From Table I, we can see that $sparsity(M_P{}^T)$ and $sparsity(M)$ drop to **mean** 0.95, **min** 0.73 and **mean** 0.79, **min** 0.46, respectively. That is the reason for the good performance of *Dense matrix* in many runs. Despite of that fact, *Sparse matrix* is still better in overall because of faster lookup by column as explained in Section III. Moreover, *Sparse matrix* still is the best stable algorithm with the lowest **std** among those with highest **#solved**.

V. Conclusion

We have proposed a linear algebraic approach for solving PHCAP using the abductive matrix in either dense or sparse formats. Experimental results demonstrate that Algorithm 1 is competitive with other existing methods. The merit of solving PHCAP in vector space is not only the scalability but also the capability of integrating with other AI techniques *e.g.* Artificial Neural Network (ANN).

In addition, taking the MHS problem into account in vector space is a potential research topic. If we can handle the MHS problem efficiently in the vector space, we can unlock the capability of GPU computing in solving large-size PHCAPs. Future work includes developing an efficient method for abduction with normal logic programs in vector spaces.

Acknowledgement

This work has been supported in part by the JSPS KAKENHI grants JP17H00763 and 18H03288. Tuan Nguyen Quoc has also been supported by Japan International Cooperative Agency "Innovative Asia".

References

[1] Josephson, J. R. and Josephson, S. G. *Abductive inference: Computation, philosophy, technology*. Cambridge University Press, 1996.

[2] Eiter, T. and Gottlob, G. The complexity of logic-based abduction. *Journal of the ACM (JACM)*, 42(1):3–42, 1995.

[3] Dai, W.-Z., Xu, Q., Yu, Y., and Zhou, Z.-H. Bridging machine learning and logical reasoning by abductive learning. In *Neural Information Processing Systems 2019*, volume 32. Curran Associates, Inc., 2019.

[4] Ignatiev, A., Narodytska, N., and Marques-Silva, J. Abduction-based explanations for machine learning models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1511–1519, 2019.

[5] de Kleer, J. An assumption-based TMS. *Artif. Intell.*, 28(2):127–162, 1986.

[6] de Kleer, J. Problem solving with the ATMS. *Artif. Intell.*, 28(2): 197–224, 1986.

[7] Reiter, R. A theory of diagnosis from first principles. *Artif. Intell.*, 32 (1):57–95, 1987.

[8] Greiner, R., Smith, B. A., and Wilkerson, R. W. A correction to the algorithm in reiter's theory of diagnosis. *Artif. Intell.*, 41(1):79–88, 1989.

[9] Inoue, K. Linear resolution for consequence finding. *Artif. Intell.*, 56 (2-3):301–353, 1992.

[10] Nabeshima, H., Iwanuma, K., Inoue, K., and Ray, O. Solar: An automated deduction system for consequence finding. *AI communications*, 23(2-3):183–203, 2010.

[11] Rocktäschel, T. and Riedel, S. End-to-end differentiable proving. In *Neural Information Processing Systems 2017*, pages 3788–3800, 2017.

[12] Sato, T. Embedding tarskian semantics in vector spaces. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[13] Sakama, C., Inoue, K., and Sato, T. Linear algebraic characterization of logic programs. In *International Conference on Knowledge Science, Engineering and Management*, pages 520–533. Springer, 2017.

[14] Nguyen, T. Q., Inoue, K., and Sakama, C. Enhancing linear algebraic computation of logic programs using sparse representation. volume 325 of *EPTCS Online Proceedings of ICLP (2020)*, pages 192–205, 2020.

[15] Aspis, Y., Broda, K., and Russo, A. Tensor-based abduction in horn propositional programs. In *ILP 2018*, volume 2206 of *CEUR Workshop Proceedings*, pages 68–75, 2018.

[16] Console, L., Dupré, D. T., and Torasso, P. On the relationship between abduction and deduction. *Journal of Logic and Computation*, 1(5): 661–690, 1991.

[17] Apt, K. R. and Bezem, M. Acyclic programs. *New Generation Computing*, 9:335–364, 1991.

[18] Selman, B. and Levesque, H. J. Abductive and default reasoning: A computational core. In *AAAI*, pages 343–348, 1990.

[19] van Emden, M. H. and Kowalski, R. A. The semantics of predicate logic as a programming language. *J. ACM*, 23(4):733–742, 1976.

[20] Gainer-Dewar, A. and Vera-Licona, P. The minimal hitting set generation problem: algorithms and computation. *SIAM Journal on Discrete Mathematics*, 31(1):63–100, 2017.

[21] Ignatiev, A., Morgado, A., and Marques-Silva, J. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pages 428–437, 2018.

[22] Bunch, J. R. and Rose, D. J. *Sparse matrix computations*. Academic Press, 2014.

[23] Koitz-Hristov, R. and Wotawa, F. Applying algorithm selection to abductive diagnostic reasoning. *Applied Intelligence*, 48(11):3976–3994, 2018.

[24] Koitz-Hristov, R. and Wotawa, F. Faster horn diagnosis-a performance comparison of abductive reasoning algorithms. *Applied Intelligence*, 50(5):1558–1572, 2020.