Efficient Abductive Learning of Microbial Interactions Using Meta Inverse Entailment

Dany Varghese[1](✉), Didac Barroso-Bergada[2], David A. Bohan[2], and Alireza Tamaddoni-Nezhad[1]

[1] Department of Computer Science, University of Surrey, Guildford, UK
{dany.varghese,a.tamaddoni-nezhad}@surrey.ac.uk
[2] Agroécologie, AgroSup Dijon, Dijon, France
{didac.barroso-bergada,david.bohan}@inrae.fr

Abstract. Abductive reasoning plays an essential part in day-to-day problem-solving. It has been considered a powerful mechanism for hypothetical reasoning in the presence of incomplete knowledge; a form of "common sense" reasoning. In machine learning, abduction is viewed as a conceptual method in which data and the bond that jointly brings the different types of inference. The traditional Mode-Directed Inverse Entailment (MDIE) based systems such as Progol and Aleph for the abduction were not data-efficient since their execution time with the large dataset was too long. We present a new abductive learning procedure using Meta Inverse Entailment (MIE). MIE is similar to Mode-Directed Inverse Entailment (MDIE) but does not require user-defined mode declarations. In this paper, we use an implementation of MIE in Python called PyGol. We evaluate and compare this approach to reveal the microbial interactions in the ecosystem with state-of-art-of methods for abduction, such as Progol and Aleph. Our results show that PyGol has comparable predictive accuracies but is significantly faster than Progol and Aleph.

Keywords: Abduction · Learning Microbial Interactions · Abductive ILP · Meta Inverse Entailment · PyGol · Bottom Clause of Relevant Literals

1 Introduction

Interactions between microbes are indispensable to successfully establishing and maintaining a population of microbes. For example, microbial communities in the soil significantly protect plants from diseases and abiotic stresses or increase nutrient uptake. This is one of the many ways in which the microbial community plays a vital role in preventing diseases caused by microbes that are themselves infectious. Microbial communities are defined by the interactions that take

ILP 2022, 31st International Conference on Inductive Logic Programming, Cumberland Lodge, Windsor, UK.

place between their members. The most recent mechanisms for meta-barcoding, such as DNA sequencing in conjunction with bio-informatics processes, are able to provide an estimate of the amount of information available on the various microbes present in a community. Machine learning models can infer an interaction network that can generalise the interaction between the microbes by using this information about the abundance and the rules of interactions as background knowledge (please refer to Fig. 1).

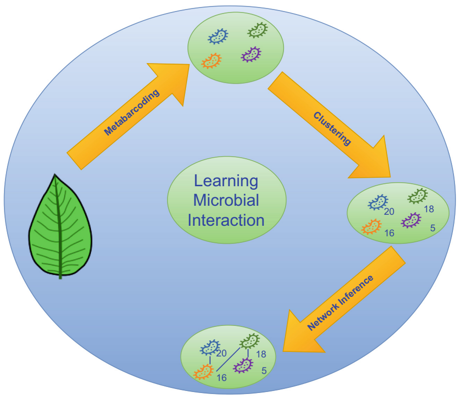


Fig. 1. Different steps involved in Microbial Interaction

Prior studies [2] introduced an Abductive/Inductive Logic Programming (A/ILP) framework to infer microbial interactions from abundance data, and Progol 5.0 [12] was used to infer the interactions in terms of logical rules and presented as a classification problem. This promising study proposes the idea of inferring ecological interaction information from diverse ecosystems, which is currently not possible to study using other methods. However, mode-directed inverse entailment systems such as Progol [12] and Aleph [20] are data-inefficient, emphasising the need for a more efficient approach to abduction.

Inductive logic programming [16] is a machine learning form that induces a hypothesis that generalises examples and can deal with very few amounts of data, even from one-shot data. It can also include background knowledge in the form of logic rules [23,24]. In contrast, most forms of ML use vectors or tensors to represent data. The ILP models are more data-efficient, explainable, and can incorporate human knowledge compared to other forms of machine learning.

Abduction is also counted as a synthetic form of reasoning along with induction. In abductive learning, logic generates new knowledge not directly included in the current theory. This sort of learning can be referred to as knowledge-intensive learning, where the new information generated drives to complete the current knowledge of the problem domain as described in the given theory. Early abduction works by Michalski [10], Ourston and Mooney [18], and Abe et al. [1] consider abductive learning a theory revision operator for specifying where the existing theory could be modified to accommodate the new learning data. Later it was realised that the role of abduction in learning could be strengthened by induction as in Progol 5.0 [12] and HAIL [19].

This paper presents abduction as an inductive approach using a novel method called Meta Inverse Entailment (MIE). In Meta Inverse Entailment (MIE), each hypothesis clause is derived from a bottom clause of relevant literals and meta theory. A bottom clause of relevant literals is a notion that efficiently collects all literals related to an example from the background knowledge and acts as a bound for the hypothesis search while abducing facts. Meta theory is a kind of language bias induced automatically from background knowledge. We implement the new learning framework for abduction using MIE in a system called PyGol[1] and presented as a Python package.

2 Background and Related Work

Abductive learning based on Inverse Entailment (IE) was first introduced in [15] and implemented in Progol 5.0, where the 'start-set' routine is regarded as a form of abduction. The system HAIL [19] also uses Bottom Set, an advanced concept from the bottom clause, and it is presented as a generalisation of Progol. The state-of-the-art system Aleph [20] can also perform abduction using Moyle's ALECTO [11] approach.

Progol 5.0 uses a standard covering algorithm where each example is generalised using a multi-predicate search [14]. This search will be done over all the predicates in the mode declarations. The two-stage process, which includes a "start-set", can be considered as a complex procedure. First, the algorithm generates the bottom clause for a seed example in the start-set and then does a covering test to find the rule covering the given example with maximum compression. Most ILP systems, such as Progol and Aleph, select a positive example and then explore its implied hypothesis space. In all cases, computing the cover set of a hypothesis is a costly process that usually involves sequentially considering all negative and relevant positive examples.

HAIL uses the 'bottom set' routine from Progol 5.0 to compute the body atoms of each Kernel clause, and M-SEARCH performs a recursive specific to general search through the collection subsumption lattices obtained from the given Kernel Set. The high-level operation of the HAIL includes abduce, deduce and search. Like its predecessor, Progol 5.0, HAIL also uses coverage testing. HAIL try to overcome some of the limitations of Progol and can find better-quality hypothesis than Progol. Also, one can consider HAIL as a greedy approach as Progol and Aleph, since HAIL begins by removing the examples covered in each stage.

The basic abductive procedure used by Aleph is a simplified variant of Moyle's ALECTO [11]. The basic workflow of ALECTO is as follows: For each positive example, an "abductive explanation" is generated. This explanation is

[1] Available from https://github.com/PyGol.

a set of ground atoms. The union of abductive explanations from all positive examples is formed. These are then generalized to give the final theory. The ground atoms in an abductive explanation are generated using Yamamoto's SOLD resolution or SOLDR [28]. Next, we introduce the fundamental definitions of induction and abduction. We assume the reader to be familiar with the basic concepts of logic programming and inductive logic programming [17]. The goal of an ILP system is to induce a logic program, H, that entails all positive examples and none of the negative examples while some prior knowledge or background knowledge is given. Formally we define the ILP as in Definition 1.

Definition 1 (An ILP learning approach). *Let E^+, E^- be the set of positive and negative examples, and B be the background knowledge. Then ILP system learn hypothesis H and has to satisfy:*

Prior Satisfiability : $B \wedge E^- \not\models \square$
Prior Necessity : $B \not\models E^+$
Posterior Satisfiability : $B \wedge E^- \wedge H \not\models \square$
Posterior Sufficiency : $B \wedge H \models E^+$

The role of abduction has been demonstrated in various applications [7,21,22]. A/ILP, a high-level knowledge-representation framework, solves problems declaratively based on abductive reasoning. It extends regular logic programming by allowing some predicates to be incompletely defined and declared as abducible predicates. In the context of formal logic, abduction is often defined as follows. Given a logical theory, T represents the expert knowledge, and a formula Q represents an observation on the problem domain, abductive inference searches for an explanation formula $\mathcal{E}$ such that:

– $\mathcal{E}$ is satisfiable w.r.t. T and
– it holds that $T \models \mathcal{E} \rightarrow Q$

In general, ε will be subjected to further restrictions, such as the aforementioned minimality criteria and the explanation formula's form (e.g. by restricting the predicates that may appear in it). This view defines an abductive explanation of observation as a rule which logically entails the observation itself. Formally, we can define abduction as in Definition 2. The definitions are taken from [8].

Definition 2. *Given an abductive logic theory (P, A, C), an abductive explanation for a query Q is a set $\Lambda \subseteq A$ of ground abducible atoms such that:*

– $P \vee \Lambda \models Q$
– $P \vee \Lambda \models C$
– $P \vee \Lambda$ *is consistent*

3 Abduction via Meta Inverse Entailment

Abductive learning is a machine learning approach which generates explanation H from a given observation E and background knowledge B. Abduction can be regarded as a form of induction in various ways. Michalski [10] described abduction in terms of induction as follows: "Given a background knowledge Th and observations O, induction hypothesises a premise H, consistent with Th, such as $H \vee Th \models O \ldots$ Induction is viewed as 'tracing backwards' this relationship". In this section, we introduce a new inductive learning approach called Meta Inverse Entailment (MIE) and present abduction as a form of MIE.

Many ILP methods use the reality that creating hypotheses incrementally as a series of short theories, each covering a few samples at a time, is usually more comfortable than making one large theory covering most of the examples. For this reason, several systems employ a so-called covering-loop [9] that uses one seed example at a time. In an ILP system, the computational cost of the search depends mainly on the cost of subsumption, which is used to evaluate the clause and then on the size of the search space. ILP systems use different kinds of biases to have a tractable search. For example, a language bias restricts the size of an acceptable hypothesis, and a search bias determines the way of the search.

Instead of selecting a random example, MIE generates the hypothesis space from all the examples using a bottom clause of relevant literals (BCRL) and meta theory (MT). MIE takes advantage of IE and MIL by introducing two novel concepts: the bottom clause of relevant literals and meta theory. BCRL bounds the hypothesis space search of MIE, and MT will guide the search. Unlike other metarule-based approaches [4,5,13] in ILP, MIE introduces a higher-order language bias, meta theory, synthesizing automatically from the background knowledge. The complete explanation of MIE is out of this report's scope, so we explain only the major concepts.

Definition 3 (Related literals). *Let $L_1 = P(s_1, s_2, \cdots, s_n)$ and $L_2 = Q(t_1, t_2, \cdots, t_m)$ be two ground literals and $\mathcal{K}$ be a set of constant terms then L_1 and L_2 are related literals if they have any common terms other than terms in $\mathcal{K}$.*

Example 1. Let $L_1 = has\_car(train_1, car_1)$, $L_2 = open(car_1)$, $L_3 = closed(car_2)$ and $\mathcal{K} = \{\}$ then according to Definition 3, L_1 and L_2 are related literals but neither L_1 and L_3 nor L_2 and L_3.

In the Example 1, L_1 and L_2 are connected since they have a common term car_1, but there is no common term between either L_1 and L_3 or L_2 and L_3.

Example 2. Let $L_1 = has\_load(car_1, circle, 2)$, $L_2 = has\_load(car_2, circle, 3)$ and $\mathcal{K} = \{circle\}$ then according to Definition 3, L_1 and L_2 are not related literals because the common term between L_1 and L_2 is present in $\mathcal{K}$.

Ordered clauses will be beneficial for collecting all the related literals of e from B. An ordered clause, denoted by $\overrightarrow{C}$, is a sequence of literals where the order and duplication of literal matter. In ILP, the usage of ordered clauses is not novel. For instance, it may be necessary to duplicate literals when using an upward refinement operator to invert an elementary substitution. Ordered clauses are used since reusing literals is forbidden in the standard encoding of clauses [17].

Definition 4 ($\overrightarrow{\mathcal{B}}$). *Let B be the background knowledge. $\overrightarrow{\mathcal{B}}$ is an ordered clause of ground literals constructed from B without any repetition of literals.*

Definition 5 (Relevant literals of an example ($\overrightarrow{\mathcal{R}_e}$)). *Let B be the background knowledge, $\overrightarrow{\mathcal{B}}$ be the ordered clause constructed from B as defined in Definition 4, and e be an example. Then the relevant literals of e is defined as $\overrightarrow{\mathcal{R}_e} = L_1 \vee L_2 \vee \cdots \vee L_n$ if and only if $L_i \in \overrightarrow{\mathcal{B}}$ and L_i be related literal of either L_j or e such that $1 \leq j < i$.*

Definition 6 (Bottom clause of relevant literals ($\bot_{e,B}$)). *Let B be the background knowledge, e be a definite clause representing an example, $\mathcal{K}$ as defined in Definition 3 and $\overrightarrow{\mathcal{R}_e}$ be the relevant literals of example e as defined in the Definition 5. Then bottom clause of relevant literals of e, denoted as $\bot_{e,B}$ is $\overrightarrow{\mathcal{R}_e}$ where each unique occurrence of term $t_i \notin \mathcal{K}$ replaced with a new variable.*

The bottom clause of relevant literals introduced in MIE can resemble the bottom clause concept in Progol, but it never uses language bias like mode declaration. The Algorithm 1 sketches a search-based algorithm to generate the bottom clause of relevant literals. Now, we define a language set for a bottom clause of relevant literals, denoted as $\overrightarrow{\mathcal{L}_\bot}$, as the set of definite ordered clauses which are sequential generalisations of $\bot_{e,B}$. This chapter focuses on languages such as $\overrightarrow{\mathcal{L}_\bot}$, comprised of clauses exhibiting the characteristics of generalisations derived from a flattened bottom clause. Consequently, all the clauses within the language $\overrightarrow{\mathcal{L}_\bot}$ can be effectively treated function-free.

Definition 7 ($\overrightarrow{\mathcal{L}_\bot}$). *Let $\bot_{e,B}$ be the bottom clause of relevant literals as defined in Definition 6 and $\overrightarrow{C}$ a definite ordered clause. $\overrightarrow{C}$ is in $\overrightarrow{\mathcal{L}_\bot}$ if and only if there exists a substitution θ such that $\overrightarrow{C}\theta$ is a subsequence of $\bot_{e,B}$.*

Definition 8 (Meta theory ($\mathcal{M}$)). *A meta theory is a higher-order well-formed-formula*

$$P(s_1, \cdots, s_m) \leftarrow Q_1(t_1, \cdots, t_n), Q_2(u_1, \cdots, u_p), \cdots, Q_N(\cdots) \qquad (1)$$

where P, Q_i are existentially quantified variables and $t_1, u_i, \cdots$ are universally quantified variables and $P \wedge Q_i \wedge \{t_i\} \wedge \{u_i\} \wedge \cdots = \emptyset$. m is the arity of arguments of the target literal, and N is the number of literals in the body of a meta theory. Two kinds of substitution will be involved using meta theory during the learning phase, such as substitution on existentially and universally quantified variables.

Definition 9 (Meta substitution). *A meta substitution Θ is a set $\{v_1/p_1, \cdots, v_n/p_n\}$ where each v_i is a distinct variable, and each p_i is a predicate. Here, t_i represents the value that is substituted for the predicate p_i.*

Algorithm 1. Algorithm to generate $\perp_{e,B}$

Input: Background knowledge (B), pass $= i$, example (e), constant set ($\mathcal{K}$)
$HashFn(t)$: A function which uniquely maps terms to variables
$GenerateLiteral(Pred, [Terms])$: A function generates a literal with $Pred$ as predicate and $[Terms]$ as its arguments.
Output: $\perp_{E,B}$

```
1: Let $Pred$ be the predicate related to the example
2: $BC = \emptyset$
3: $HeadTerms = \emptyset$
4: $TermSet = \emptyset$
5: $BodyLiterals = \emptyset$
6: for each term $e_i$ in $e$ do
7:    if $e_i \notin \mathcal{K}$ then
8:       push $HashFn(e_i)$ to $HeadTerms$
9:       push $e_i$ to $TermSet$
10:   else
11:      push $e_i$ to $HeadTerms$
12:   end if
13: end for
14: Let $h = GenerateLiteral(Pred, HeadTerms)$
15: k=1
16: while k $\leq$ pass do
17:    for each $b_i$ in B do
18:       Let $P_{b_i}$ be the predicate related to $b_i$
19:       if any term of $b_i$ in $TermSet$ then
20:          $TempTerms = \emptyset$
21:          for each term $t_i$ of $b_i$ do
22:             if $t_i \notin \mathcal{K}$ then
23:                push $HashFn(t_i)$ to $TempTerms$
24:                push $t_i$ to $TermSet$
25:             else
26:                push $t_i$ $TempTerms$
27:             end if
28:          end for
29:          if $generate\_literal(P_{b_i}, TempTerms) \notin BodyLiterals$ then
30:             push $generate\_literal(P_{b_i}, TempTerms)$ to $BodyLiterals$
31:          end if
32:       end if
33:    end for
34:    k = k+1
35: end while
36: return $BC = h \leftarrow BodyLiterals$
```

Definition 10 ($\overrightarrow{\mathcal{L}_{\mathcal{M}}}$). *Let $\perp_{e,B}$ be the bottom clause of relevant literals as defined in Definition 6 and $\overrightarrow{C}$ be a meta theory as defined in Definition 8. $\overrightarrow{C}$ is in $\overrightarrow{\mathcal{L}_{\mathcal{M}}}$ if and only if there exists a meta substituition Θ such that $\overrightarrow{C}\Theta$ is a subsequence of $\perp_{e,B}$.*

The definition of meta theory and meta-substitution is motivated by the concept of metarules [3]. The incompleteness of Progol was discussed by Yamamoto [26,27], and one of the incompleteness is due to its inability to generate multiple hypotheses from a single bottom clause. In MIE, we solved the incompleteness of Progol by introducing the concept of the double-bounded hypothesis set.

Definition 11 (Double-bounded hypothesis set ($HS(\perp_{e,B}, \mathcal{M}_e)$)). *Let e be an example, $\perp_{e,B}$ be a bottom clause of relevant literals of e as defined in Definition 6, $\mathcal{M}_e$ be a meta theory as defined in Definition 8. Then*

$$HS(\perp_{e,B}, \mathcal{M}_e) = \{h| \textit{ s.t: } (1)\ h \in \overrightarrow{\mathcal{L}_\perp}, \textit{ and } (2)\ h \in \overrightarrow{\mathcal{L}_\mathcal{M}}\}$$

From the Definition 11, it is clear that the double-bounded hypothesis set can be considered as a combination of top-down and bottom-up approaches, and each hypothesis will be generated as sequential subsumption of meta-theory relative to bottom clause of relevant literals. Instead of starting from a seed example, we generate the bottom clause of relevant literals of all the examples. This global-theory generating mechanism is not new to the ILP community, as a similar approach can be seen in the bottom clause propositionalisation [6].

Definition 12 (Abduction using MIE). *Let B be the background knowledge, $HS(\perp_{e,B}, \mathcal{M}_e)$ be the double-bounded hypothesis set as defined in the Definition 11, e be an example, A be the abducible predicate and R be the rule to explain the observable predicates such that $R = R^{'}A$. Let $\perp_{e,B}$ the bottom clause of relevant literals of e, and $\hat{H}$ be the set of inductive hypothesis; $\hat{H} = HS(\perp_{e,B}, R^{'})$. Then the MIE will generate a set of ground abductive hypotheses, $\Lambda = \{a\theta \colon h\theta \models \perp_{e,B},\ h \in \hat{H}$ and $a \in A\}$; $\hat{H} = \mathcal{HS}(\perp_{B,e}, R^{'})$. Then the MIE will generate a set of ground abductive hypotheses, $\Lambda = \{a\theta \colon h\theta \models \perp_{B,e},\ h \in \hat{H}$ and $a \in A\}$;*

- $R \wedge \Lambda \models E$
- $\Lambda \not\models B$

Example 3. This example illustrates the steps through which we can perform abduction using MIE [22] (please refer the Fig. 2). Let B, e, K, A as formulated in Definition 12. Then the abductive procedure starts by generating the bottom clause of relevant literals, $\perp_{B,e}$ for the example e and later generates the double-bounded hypothesis set $\hat{H} = HS(\perp_{B,e}, R^{'})$. The abductive procedure will keep track of θ for each $h \in \hat{H}$ and generate ground abducible facts.

4 Abduction of Microbial Interaction Using MIE

Microbial interactions refer to the various ways in which microbes interact with each other and with other organisms. Microbial interactions include cooperative and competitive interactions, such as symbiosis, commensalism, parasitism,

B
taxacode(a)., taxacode(b)., taxacode(c)., taxacode(d)., taxacode(e)., taxacode(f)., predator(a)., predator(b)., predator(c)., size_class(a, 4)., size_class(b, 3)., size_class(c, 2)., size_class(d, 1)., size_class(e, 1)., size_class(f, 1)., bigger(a,b)., bigger(a,c)., bigger(a,d)., bigger(a,e)., bigger(a,f)., bigger(b,c)., bigger(b,d)., bigger(b,e)., bigger(b,f)., bigger(c,d)., bigger(c,e)., bigger(c,f)., abundance1(a, s1, down)., abundance1(a, s2, down)., abundance1(a, s3, down)., abundance1(a, s4, down)., abundance1(b, s1, up)., abundance1(b, s2, up)., abundance1(b, s3, down)., abundance1(b, s4, down)., abundance1(c, s1, down)., abundance1(c, s2, down)., abundance1(c, s3, up)., abundance1(c, s4, up)., abundance1(d, s1, up)., abundance1(d, s2, up)., abundance1(d, s3, down)., abundance1(d, s4, down)., abundance1(e, s1, up)., abundance1(e, s2, up)., abundance1(e, s3, down)., abundance1(e, s4, down)., abundance1(f, s1, down)., abundance1(f, s2, down)., abundance1(f, s3, up)., abundance1(f, s4, up).

e	K	A
abundance(a, s1, down).	{a,b,c,d,e,f}	eats

R	R'
abundance(X,S,D) :- predator(X), bigger(X,Y), abundance1(Y,S,D), eats(X,Y).	abundance(X,S,D) :- predator(X), bigger(X,Y), abundance1(Y,S,D).

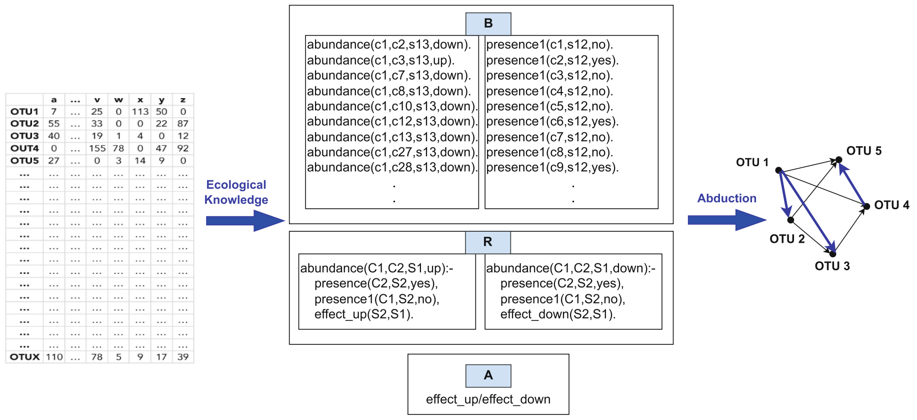
$\perp_{e,B}$
abundance(a, B, C):- taxacode(a), predator(a), size_class(a,4), bigger(a,b), bigger(a,c), bigger(a,d), bigger(a,e), bigger(a,f), abundance1(a,B,C), abundance1(a,D,C), abundance1(a,E,C), abundance1(a,F,C), abundance1(b,B,G), abundance1(b,D,G), abundance1(b,E,C), abundance1(b,F,C), abundance1(c,B,C), abundance1(c,D,C), abundance1(c,E,G), abundance1(c,F,G), abundance1(d,B,G), abundance1(d,D,G), abundance1(d,E,C), abundance1(d,F,C), abundance1(e,B,G), abundance1(e,D,G), abundance1(e,E,C), abundance1(e,F,C), abundance1(f,B,C), abundance1(f,D,C), abundance1(f,E,G), abundance1(f,F,G).

$\hat{H}$	θ	Λ
abundance(a,B,C):- predator(a), bigger(a,c), abundance1(c,B,C).	{X : a, Y : c}	eats(a,b)
abundance(a,B,C):- predator(a), bigger(a,f), abundance1(f,B,C).	{X : a, Y : f}	eats(a,f)

Fig. 2. (Example 3) Learning abducible explanations using MIE

predation, and competition. Microbes can also interact indirectly by releasing metabolites into their environment, which can affect the growth and development of other microbes. Microbial interactions are essential for the functioning of ecosystems since they contribute to the nutrient and energy cycles and can influence the structure and composition of microbial communities. In a more general way, A microbial interaction can be defined as a conserved effect on the abundance of one microbial species caused by the presence of another. Thus, the abductive procedure aims to infer interactions, following ecological theory to explain the observed changes in the abundance of the species. Barroso-Bergada et al. successfully encoded the microbial interactions into logical clauses and applied A/ILP using Progol in [2].

The abductive procedure aims to infer interactions following ecological theory to explain the observed changes in the abundance of the species. The steps involved in an abductive procedure are shown in Fig. 3.



	a	...	v	w	x	y	z
OTU1	7	...	25	0	113	50	0
OTU2	55	...	33	0	0	22	87
OTU3	40	...	19	1	4	0	12
OUT4	0	...	155	78	0	47	92
OTU5	27	...	0	3	14	9	0
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
OTUX	110	...	78	5	9	17	39

B

```
abundance(c1,c2,s13,down).
abundance(c1,c3,s13,up).
abundance(c1,c7,s13,down).
abundance(c1,c8,s13,down).
abundance(c1,c10,s13,down).
abundance(c1,c12,s13,down).
abundance(c1,c13,s13,down).
abundance(c1,c27,s13,down).
abundance(c1,c28,s13,down).
            .
            .
```

```
presence1(c1,s12,no).
presence1(c2,s12,yes).
presence1(c3,s12,no).
presence1(c4,s12,no).
presence1(c5,s12,no).
presence1(c6,s12,yes).
presence1(c7,s12,no).
presence1(c8,s12,no).
presence1(c9,s12,yes).
            .
            .
```

R

```
abundance(C1,C2,S1,up):-
    presence(C2,S2,yes),
    presence1(C1,S2,no),
    effect_up(S2,S1).
```

```
abundance(C1,C2,S1,down):-
    presence(C2,S2,yes),
    presence1(C1,S2,no),
    effect_down(S2,S1).
```

A

```
effect_up/effect_down
```

Fig. 3. Abductive learning of microbial interaction

The first step for abducing the inference is to reflect the abundance changes between communities of each species using logical statements. In [2], the 'abundance change' and 'presence' logical statements are used as observations in an abduction process. The observable predicate 'abundance' is defined as 'abundance (C1, C2, S1, Dir)'. Here, C1 and C2 symbolize two different community samples where species S1 is present, and 'Dir' is the change in the direction of abundance. The 'presence' of each species is also converted to a logical statement with the structure: presence(C1, S2, yes/no) where C1 refers to a sample community, S2 to a species and yes/no describes if S2 is present in C1 or not. The second step is to encode the interaction hypothesis using logical statements. The logical statements to define the observations are given in Eq. 2, in which $effect\_up$ and $effect\_down$ are the abducible predicates. These abducibles are used to learn whether two species interact positively (up) or negatively (down) in a community.

$$\begin{aligned}
\text{abundance(C1, C2, S1, up)} \leftarrow\ & \text{presence(C2, S2, yes)}, \\
& \text{presence(C1, S2, no)}, \\
& \text{effect\_up(S2, S1)} \\
\text{abundance(C1, C2, S1, down)} \leftarrow\ & \text{presence(C2, S2, yes)}, \\
& \text{presence(C1, S2, no)}, \\
& \text{effect\_down(S2, S1)}
\end{aligned} \tag{2}$$

5 Empirical Evaluation

The performance of PyGol was evaluated using artificially generated datasets[2] introduced in [2]. We have mainly considered two criteria for comparison with two state-of-the-art systems, Aleph and Progol: accuracy and execution time.

5.1 Materials and Methods

For the experiment setup, we have chosen nine different datasets of 50 species from 3 different strengths (2, 3 and 5). Table 1 will give you the statistical information of the datasets we are considering. Since the interactions that drive the abundance of the computer-generated tables are known, it is possible to treat interaction inference as a classification problem. Interactions can be classified between existing and non-existing, and the estimator values obtained using the different functions are the classification accuracy. Thus, the area under the curve (AUC) of the true positive rate against the false positive rate (ROC curve) can be used to measure performance.

In all the three systems, we have used the same experimental setup. The search for the best hypotheses is guided by an evaluation function called 'compression', which is defined as $f = p-(c+n)$, where p is the number of observations (training examples) correctly explained by the hypothesis (positive examples), n is the number incorrectly explained (negative examples) and c is the length of the hypothesis (in this study, it always One because the hypothesis is a single fact). The rules to define the observations are shown in the Eq. 2, and atoms $effect\_up$ and $effect\_down$ are the abducible.

Other than Progol, we have also considered Aleph [20], a state-of-the-art ILP algorithm, for the empirical evaluation. Like PyGol, Aleph also uses the advantage of inverse entailment but mode-directed inverse entailment. A novel user-friendly Python/Jupyter interface for Inductive Logic programming, PyILP [25], was used to run the experiments for Aleph and Progol 5.0 was used for progol-related experiments.

6 Results and Discussions

Figure 4, records the area under ROC curves, and Fig. 5 displays the total execution time from different experiments. In Fig. 4, each row represents datasets of different strengths, such as 2, 3 and 5, as in Table 1. PyGol outperforms all other approaches in most of the experiments. The average difference between PyGol and Aleph is 4.3% and 1.7% with Progol. Regarding the execution time, it is evident that PyGol is very fast compared to Progol and Aleph. While considering all the execution time, it is clear that PyGol is 45 to 65 times faster than Aleph and 40 to 60 times faster than Progol.

The primary reason for the speedy execution time for PyGol is its new efficient way of generating the hypothesis space using meta inverse entailment.

[2] Available from https://github.com/danyvarghese/IJCLR22-Abduction.

Table 1. Dataset statistics

Dataset	Strength	No. of observations				
		Species	abundance	presence	presence1	Total
S_1	2	40	17690	2000	2000	21690
S_2			25252			29252
S_3			23680			27680
S_4	3		20732			24732
S_5			21774			25774
S_6			28190			32190
S_7	5		20742			24742
S_8			25072			29072
S_9			24708			28708

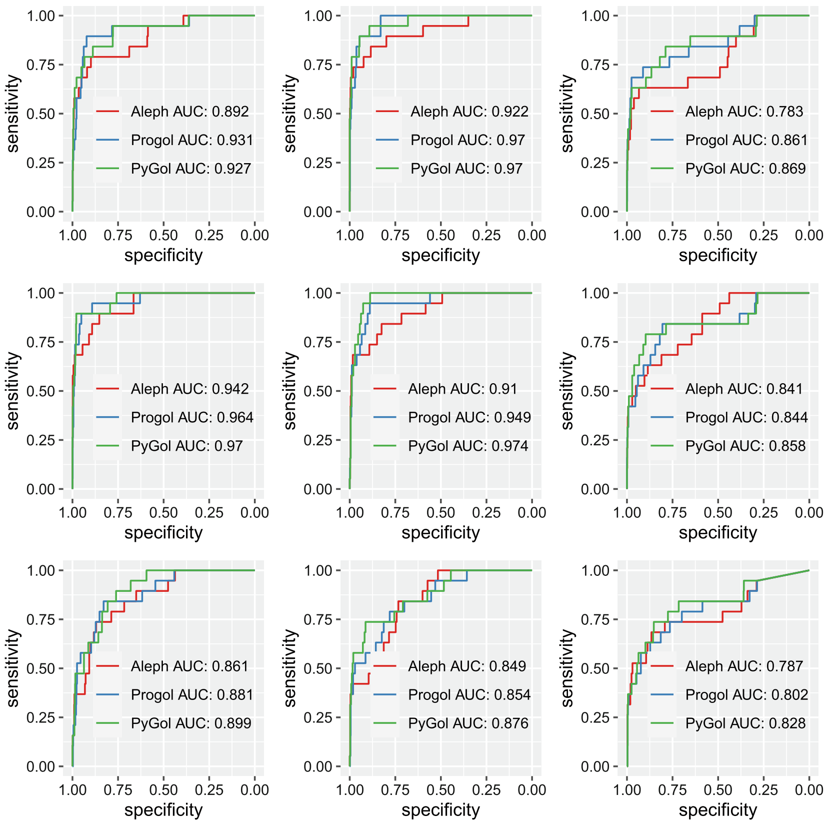


Fig. 4. Comparison of the area under the ROC curve

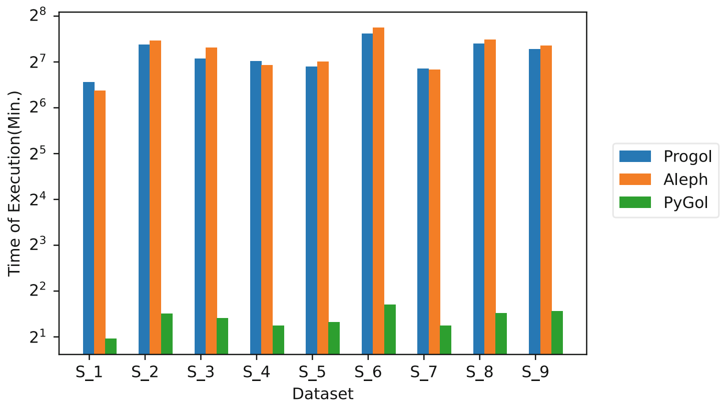


Fig. 5. Comparison on execution time

The theoretical framework for converting the general subsumption to atomic subsumption makes MIE more efficient. While considering the results, the new abductive learning approach using meta inverse entailment is more efficient than Progol and Aleph. MIE could learn all the abducted facts generated by both approaches quickly, showing the system's impact on hypothesis space generation. The abductive framework using MIE will not affect the order of examples as it follows a global-theory-making mechanism, but Progol and Aleph follow the greedy approach.

7 Conclusion

This paper presents an application that uses abductive learning to infer microbial interaction. We presented abduction as a variant of inductive learning in the first part of the paper, using a novel ILP approach called meta inverse entailment. The bottom clause of relevant literals and meta theory is used in meta inverse entailment, generated automatically from background knowledge without any declarative bias of user interaction. In the second part, we presented a novel abductive framework for learning microbial interaction, which we implemented as a Python package in PyGol.

According to empirical evaluations, the MIE is efficient enough for abductive learning regarding execution time and accuracy. The results show that PyGol outperforms other systems by 40 to 60 times. The new learning approach demonstrates the theoretical strength of the system introduced using sequential subsumption of meta-theory relative to the bottom clause of relevant literals, which can be reduced to atomic subsumption and performed without much complexity.

References

1. Adé, H., Malfait, B., De Raedt, L.: RUTH: an ILP theory revision system. In: Raś, Z.W., Zemankova, M. (eds.) ISMIS 1994. LNCS, vol. 869, pp. 336–345. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-58495-1_34
2. Barroso-Bergada, D., Tamaddoni-Nezhad, A., Muggleton, S.H., Vacher, C., Galic, N., Bohan, D.A.: Machine learning of microbial interactions using abductive ILP and hypothesis frequency/compression estimation. In: Katzouris, N., Artikis, A. (eds.) Inductive Logic Programming, ILP 2021. LNCS, vol. 13191, pp. 26–40. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-97454-1_3
3. Cropper, A.: Efficiently learning efficient programs. Ph.D. thesis. Imperial College London, UK (2017)
4. De Raedt, L., Bruynooghe, M.: Interactive concept-learning and constructive induction by analogy. Mach. Learn. **8**(2), 107–150 (1992)
5. Evans, R., Grefenstette, E.: Learning explanatory rules from noisy data. J. Artif. Int. Res. **61**(1), 1–64 (2018)
6. França, M.V.M., Zaverucha, G., Garcez, A.: Fast relational learning using bottom clause propositionalization with artificial neural networks. Mach. Learn. **94**(1), 81–104 (2014). https://doi.org/10.1007/s10994-013-5392-1. https://openaccess.city.ac.uk/id/eprint/3057/
7. Kakas, A., Tamaddoni, N.A., Muggleton, S., Chaleil, R.: Application of abductive ILP to learning metabolic network inhibition from temporal data. Mach. Learn. **64**, 209–230 (2006). https://doi.org/10.1007/s10994-006-8988-x
8. Kakas, A.C., Kowalski, R.A., Toni., F.: Abduction in logic programming. J. Log. Comput. **2**, 719–770 (1993)
9. Michalski, R.S.: A theory and methodology of inductive learning. Artif. Intell. **20**(2), 111–161 (1983)
10. Michalski, R.S.: Inferential theory of learning as a conceptual basis for multistrategy learning. Mach. Learn. **11**(2–3), 111–151 (1993)
11. Moyle, S.: Using theory completion to learn a robot navigation control program. In: Matwin, S., Sammut, C. (eds.) ILP 2002. LNCS (LNAI), vol. 2583, pp. 182–197. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36468-4_12
12. Muggleton, S.: Inverse entailment and Progol. N. Gener. Comput. **13**, 245–286 (1995)
13. Muggleton, S., Lin, D., Tamaddoni-Nezhad, A.: Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited. Mach. Learn. **100**(1), 49–73 (2015). https://doi.org/10.1007/s10994-014-5471-y
14. Muggleton, S.: Learning from positive data. In: Muggleton, S. (ed.) ILP 1996. LNCS, vol. 1314, pp. 358–376. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63494-0_65
15. Muggleton, S.H., Bryant, C.H.: Theory completion using inverse entailment. In: Cussens, J., Frisch, A. (eds.) ILP 2000. LNCS (LNAI), vol. 1866, pp. 130–146. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44960-4_8
16. Muggleton, S., de Raedt, L.: Inductive logic programming: Theory and methods. J. Log. Program. **19–20**, 629–679 (1994). Special Issue: Ten Years of Logic Programming
17. Nienhuys-Cheng, S.-H., de Wolf, R.: Foundations of Inductive Logic Programming. LNCS, vol. 1228. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-62927-0

18. Ourston, D., Mooney, R.J.: Theory refinement combining analytical and empirical methods. Artif. Intell. **66**(2), 273–309 (1994)
19. Ray, O., Broda, K., Russo, A.: Hybrid Abductive inductive learning: a generalisation of Progol. In: Horváth, T., Yamamoto, A. (eds.) ILP 2003. LNCS (LNAI), vol. 2835, pp. 311–328. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39917-9_21
20. Srinivasan, A.: A learning engine for proposing hypotheses (Aleph) (2001). https://www.cs.ox.ac.uk/activities/programinduction/Aleph/aleph.html
21. Tamaddoni-Nezhad, A., Lin, D., Watanabe, H., Chen, J., Muggleton, S.: Machine Learning of Biological Networks Using Abductive ILP, pp. 363–401. Wiley, Hoboken (2014). https://doi.org/10.1002/9781119005223.ch10
22. Tamaddoni-Nezhad, A., Bohan, D., Raybould, A., Muggleton, S.: Towards machine learning of predictive models from ecological data. In: Davis, J., Ramon, J. (eds.) ILP 2014. LNCS (LNAI), vol. 9046, pp. 154–167. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23708-4_11
23. Varghese, D., Bauer, R., Baxter-Beard, D., Muggleton, S., Tamaddoni-Nezhad, A.: Human-like rule learning from images using one-shot hypothesis derivation. In: Katzouris, N., Artikis, A. (eds.) Inductive Logic Programming, ILP 2021. LNCS, vol. 13191, pp. pp 234–250. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-97454-1_17
24. Varghese, D., Tamaddoni-Nezhad, A.: One-shot rule learning for challenging character recognition. In: Proceedings of the 14th International Rule Challenge, August 2020, Oslo, Norway, vol. 2644, pp. 10–27 (2020)
25. Varghese, D., Tamaddoni-Nezhad, A.: Pyilp (2022). https://github.com/danyvarghese/PyILP/
26. Yamamoto, A.: Revising the logical foundations of inductive logic programming systems with ground reduced programs. New Gener. Comput. **17**, 119–127 (1998). https://cir.nii.ac.jp/crid/1571417125491386240
27. Yamamoto, A.: Which hypotheses can be found with inverse entailment? In: Lavrač, N., Džeroski, S. (eds.) ILP 1997. LNCS, vol. 1297, pp. 296–308. Springer, Heidelberg (1997). https://doi.org/10.1007/3540635149_58
28. Yamamoto, A.: Using abduction for induction based on bottom generalization. In: Flach, P.A., Kakas, A.C. (eds.) Abduction and Induction. Applied Logic Series, vol. 18, pp. 267–280. Springer, Dordrecht (2000). https://doi.org/10.1007/978-94-017-0606-3_17