Inductive Learning in Shared Neural Multi-Spaces

Edjard de Souza Mota[1,2], Jacob M. Howe[1], and Artur S. d'Avila Garcez[1]

[1] City, University of London, London, EC1V 0HB, UK
{edjard.de-souza-mota,j.m.howe,a.garcez}@city.ac.uk,
[2] Universidade Federal do Amazonas,
Instituto de Computação, Campus Setor Norte
Coroado - Manaus - AM - Brasil CEP: 69080-900

Abstract. The learning of rules from examples is of continuing interest to machine learning since it allows generalization from fewer training examples. Inductive Logic Programming (ILP) generates hypothetical rules (clauses) from a knowledge base augmented with (positive and negative) examples. A successful hypothesis entails all positive examples and does not entail any negative example. The Shared Neural Multi-Space (Shared NeMuS) structure encodes first order expressions in a graph suitable for ILP-style learning. This paper explores the NeMuS structure and its relationship with the Herbrand Base of a knowledge-base to generate hypotheses inductively. It is demonstrated that inductive learning driven by the knowledge-base structure can be implementated successfully in the Amao cognitive agent framework, including the learning of recursive hypotheses.

1 Introduction

There is renewed interest in inductive logic programming as a framework for machine learning owing to its human like ability to infer new knowledge from background knowledge together with a small number of examples. It also comes with strong mathematical and logical underpinnings. This paper builds on [9], where a data structure (Shared NeMuS) for the representation of first-order logic was introduced. It revisits inductive logic programming and demonstrates that Shared NeMuS provides a structure that can be used to build an inductive logic programming system.

A part of the Shared NeMuS structure is weightings on individual elements (atom, predicate, function). The purpose of these weightings is to provide a guide to the use of these elements, for example in theorem proving. One of the key challenges in inductive logic programming is to find good heuristics to search the hypothesis space; the long-term goal of this work is to learn weights in a training phase that can in turn be used to guide the search of the hypothesis space and improve the efficiency and success of inductive learning.

The main purpose of this work is to present a new approach for Clausal Inductive Learning (CIL) using Shared NeMuS in which the search mechanism

uses the Herbrand Base (HB) to build up hypothesis candidates using inverse unification. This generalization of ground expressions to universally quantified ones is supported by the idea of *regions of concepts* that was explored in [9] to find refutation patterns. Here, weights are not explicitly used, but the intuitive use of them is explored to define *linkage patterns* between predicates of the HB and the occurrences of ground terms in positive examples. Meaningless hypotheses are pruned away as a result of *inductive momentum* between predicates connected to positive and negative examples. This paper makes the following contributions: it is demonstrated that the Shared NeMuS data structure can be used as a suitable structure for inductive logic programming; the Herbrand Base is used to build candidate hypotheses; chaining and abstraction for rules, including recursive rules, are given; these rules help keep the number of hypotheses small. NeMuS is designed for extension with machine learning tactics.

The remainder of this paper is structured as follows: section 2 gives some brief background on inductive logic programming and the Shared NeMuS data structure, sections 3 and 4 describe the implementation of inductive learning in Amao using the Shared NeMuS data structure, then section 5 describes some related work and section 6 discusses the work presented.

2 Background

2.1 Inductive Logic Programming (ILP)

The goal of inductive logic programming (introduced in [10]) is to learn logical formulae that describe a target concept, based on a set of examples. In a typical set up there is a knowledge base of predicates, called background knowledge (BK), along with a set of examples that the target concept should prove (positive examples, e^+) and a set of examples that the target concept should not prove (negative examples, e^-). The inductive logic programming problem is to search for a logical description (a hypothesis, H) of the target concept based on the knowledge and the examples so that the knowledge base plus the hypothesis entails the positive examples, whilst does not entail the negative examples.

Inductive logic programming systems implement search strategies over the space of possible hypotheses. A good heuristic is one that will a) arrive at a successful hypothesis, b) find this hypothesis quickly and c) find a succinct hypothesis. In order to achieve this, the efficiency of hypothesis searching mechanisms depend on partial order of θ-subsumption [13], or on a total ordering over the Herbrand Base to constrain deductive, abductive and inductive operations [12].

The approach presented in this paper for CIL takes a totally different approach. Search considers separated spaces for constant terms, predicates and clauses, and elements are indexed by their unique identification codes. The spaces are interconnected through weighted bindings pointing to the target space in which an occurrence of an element appears. This creates a network of shared spaces because the elements are shared via bindings. As the weights are not used here, the networks shall be referred to as multi-spaces.

2.2 Shared NeMuS

Shared Neural Multi-Space (Shared NeMuS) [9] takes inspiration from [2] to give a shared multi-space representation for a portion of first-order logic designed for use with machine learning and neural network methods. The structure incorporates a relative degree of importance for each element according to the element's attributes and uses an architecture that leads to a fast implementation.

Shared NeMuS uses a Smarandache multi-space [8], a union of n spaces $A_1, ..., A_n$ in which each A_i is the space of a distinct observed characteristic of the overall space. For each A_i there is a different metric to describe a different side of the "major" side. There will be a space for each component of a first-order language: atomic constants (of the Herbrand Universe, space 0), functions (space 1), predicates with literal instances (space 2), and clauses (space 3). Variables are used to refer to sets of atomic terms via quantification, and they belong to the same space of atoms. In this work, the function space is suppressed, since it is not being dealt with for relational learning. In what follows vectors are written $\boldsymbol{v}$, and $\boldsymbol{v}[i]$ or $\boldsymbol{v}_i$ is used to refer to an element of a vector at position i.

Each logical element is described by a T-Node, and in particular each element is described by a unique integer code within its space. In addition, a T-Node identifies the lexicographic occurrence of the element, and (when appropriate) an attribute position.

Definition 1 (T-Node and Binding) *Let $c, a, i \in \mathbb{Z}$ and $h \in \{0, 1, 2, 3\}$. A* T-Node (target node) *is a quadruple (h, c, i, a) that identifies an object at space h, with code c and occurrence i, at attribute position a (when it applies, otherwise 1). If p is a T-Node, then $n_h(p) = h$, $n_c(p) = c$, $n_a(p) = a$ and $n_i(p) = i$. A NeMuS* Binding *is a pair $(p, w)_k$, which represents the influence w of object k over occurrence $n_i(p)$ of object $n_c(p)$ at space $n_h(p)$ in position $n_a(p)$.*

The elements of the subject space represent all of the occurrences of atoms and variables in an expression.

Definition 2 (Subject Space) *Let $\boldsymbol{C} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m]$ and $\boldsymbol{V} = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n]$, where each $\boldsymbol{x}_i, \boldsymbol{y}_i$ is a vector of bindings.* Subject Space *refers to the pair $(\boldsymbol{C}, \boldsymbol{V})$ for constants and variables, respectively.*

The function β maps a constant i to the vector of its bindings $\boldsymbol{x_i}$, as above.

Higher spaces are made of structured elements, such as predicates and clauses. Such objects have attributes uniquely identified by T-Nodes referring to objects in the spaces below and their bindings of the objects on which they exert influence.

Definition 3 (Compound) *Let $\boldsymbol{x}_a^i = [c_1, \ldots, c_m]$ be a vector of T-Nodes for each attribute instance of compound i. Let $\boldsymbol{w}_i$ be a vector of NeMuS bindings. Then a NeMuS* Compound *is the pair $(\boldsymbol{x}_a^i, \boldsymbol{w}_i)$. A NeMuS* Compound Space (C-Space) *is a vector of NeMuS Compounds.*

For each literal there is a C-Space to represent it. Since a literal is an instance of a predicate the predicate space is a vector of C-Spaces. As predicates have positive and negative instances, there are two vector regions for a predicate space. Clauses' attributes are the literals that they are made of, and as they exert no influence upon spaces above for simplicity the bindings of a clause shall be empty.

Definition 4 (Predicate and Clause Spaces) *Let* $\boldsymbol{C}_p^+$ *and* $\boldsymbol{C}_p^-$ *be two vectors of C-spaces. The pair* $(\boldsymbol{C}_p^+, \boldsymbol{C}_p^-)$ *is called the NeMuS* Predicate Space*. A NeMuS* Clause Space *is a vector of C-spaces such that every pair in the vector shall be* $(\boldsymbol{x}_a^i, [])$*.*

A Shared NeMuS for a coded first-order expression is a triple $\langle \mathcal{S}, \mathcal{P}, \mathcal{C} \rangle$, in which $\mathcal{S}$ is the subject space, $\mathcal{P}$ is the predicate space and $\mathcal{C}$ is the clause space.

Example 1. Assume the following symbolic BK (adapted from [3]):

```
mother(pam,ann).
wife(ann,bob).
wife(eve,wallie).
brother(wallie, ann).
```

This translates into the Shared NeMuS below (with weights at default value of 0). The constant region of the subject space, the predicate space and the clause space are each given with some commentary. The labels and the numbers before the column are just to emphasise structure.

```
Subject Space:   Constant region: {
  1:[((2,1,1,1),0)]
  2:[(2,1,1,2),0),(2,2,1,1),0),((2,3,1,2),0)]
  3:[((2,2,1,2),0)]
  4:[((2,2,2,1),0)]
  5:[((2,2,2,2),0),((2,3,1,1),0)]  }
```

The subject space encodes the occurrences of the constants. For example, the first entry gives the binding for `pam`, with code 1, stating that this atom occurs in the first occurrence of the first predicate (`mother`) as the first attribute. The second entry gives the list of bindings for the three occurrences of `ann`.

```
Predicate Space:
    //      pam        ann
   {+1:[([(0,1,1,1),(0,2,1,2)],[((3,1,1,1),0)])],  -1: []}
    //      ann        bob
   {+2:[([(0,2,2,1),(0,3,1,2)],[((3,2,1,1),0)]),
        ([(0,4,1,1),(0,5,1,2)],[((3,3,1,1),0)])],  -2: []}
   {+3:[([(0,5,2,1),(0,2,3,2)],[((3,4,1,1),0)])],  -3: []}
```

The predicate space encodes each predicate. For example, the first entry links to the bindings of the two constants occurring in the only clause in which it occurs. The second entry details the two clauses for `wife`.

```
Clause Space:
   {([(2,1,2,1)],[()]), ([(2,2,2,1)],[()]),
    ([(2,2,3,1)],[()]), ([(2,3,2,1)],[()])}
```

Here, the clauses link back to the predicates that define them. For example, the first entry says that the first clause is built from the first predicate.

This simple example shows how easy it is to navigate across a shared NeMuS structure to induce new rules from relational data or sets of literals.

3 Inductive Learning with Shared NeMuS

This section describes, with the aid of running examples, how inductive learning is performed in Amao, using the Shared NeMuS structure. Amao[3] a cognitive artificial agent which originally performed just symbolic reasoning on structured clauses via Linear Resolution [16]. It was extended in [9] to generate a shared NeMuS representation, during the compilation of symbolic representation, for neural-symbolic reasoning purposes.

Inverse unification from HB is *plausible* since all ground expressions of a given concept p will be at the same region as far as weighted grounds are concerned. For instance, $p(a, b)$, $p(c, d)$ and $p(b, e)$ all belong to the region of p. So, $p(X, Y)$ is a sound generalization of such instances. However, if there are other concepts involving the elements of the Herbrand Universe, say $q(a, d)$ and $r(d)$ then $p(X, Y)$ is not a straight generalization without taking into account the combination of the regions for r and q since their ground atoms have occurrences of constants appearing in all three concepts.

The induction algorithm proposed is guided by kinds of *linkage patterns* (sections 3.1 and 3.2) and using only those in which there is an *inductive momentum* (section 3.3). The explanation is that positive examples bring the ground expressions "close" to the hypothesis to be generated, while the negative ones pull apart those which are likely to generate inconsistent hypothesis.

3.1 Linear Linkage Patterns

One form of the Amao operation for performing inductive learning with target predicate `p/n` is:

```
consider induction on p(X1,...,Xn) knowing p(t1,...,tn).
```

That is, `p/n` is not in the BK and Amao will attempt to find a hypothesis H such that the positive example(s) `p(t1,...,tn)` can be deduced from $BK \cup H$. In what follows, the symbol representation will be used to mean the Shared NeMuS code of each logical element and recall that β maps code symbols to their bindings.

[3] Amao is the name of a deity that taught people of Camanaos tribe, who lived on the margins of the Negro River in the Brazilian part of the Amazon rainforest, the process of making mandioca powder and beiju biscuit for their diet.

The BK is that used in Example 1. Suppose that the target predicate is `motherInLaw/2`, with positive knowledge `motherInLaw(pam,bob)` then when Amao is asked to generate hypotheses with

```
consider induction on motherInLaw(X,Y) knowing motherInLaw(pam,bob).
```

amongst the successful hypotheses should be:

$$motherInLaw(X,Y) \leftarrow mother(X,Z) \wedge wife(Z,Y)$$

Parsing the positive example against the Shared NeMuS representation of the BK gives that in the constant region `pam` has code 1 and `bob` has code 3. From the constant region of the subject space the vectors of bindings $\beta(1)$ and $\beta(3)$ are found:

$\beta(1) = [(1,1,1)]$: in the first predicate, its first instance, as first attribute
$\beta(3) = [(2,1,2)]$: in the second predicate, its first instance, as second attribute

For each element of the vector of bindings, $\beta(i)$, the vector of attributes of the predicate in which it occurs is found, written $x_a(\beta(i)_j)$, where j is an index. Here,

$$x_a(\beta(1)_1) = [(0,1,1,1),(0,2,1,2)] \text{ and } x_a(\beta(3)_1) = [(0,2,2,1),(0,3,1,2)]$$

that is, `mother(pam, ann)` and `wife(ann, bob)`.

The intersection between $x_a(\beta(pam)_1)$ and $x_a(\beta(bob)_1)$ is non-empty since `ann` (code 2) occurs in both. Hence predicate codes 1 (`mother`) and 2 (`wife`) are used in the hypothesis. These are ground atoms from the HB of Example 1, and from them a new clause is built with head (positive) literal as the target anti-unified, and the negative literals are those found above. Inverse unification will incrementally build anti-substitution θ^{-1} at each step by adding literals to the body with constants substituted by variables `X` and `Y` from the targeted head. As `X` appears as the first attribute of the first predicate (`mother`), and `Y` as the second attribute of the second predicate (`wife`), then the linkage term shall be `Z_0`. Call `Z_0` the *hook* between both literals and the final θ^{-1} is {`pam/X,bob/Y,ann/Z_0`}. The terms which are not the hook term are called *attribute-mates*. Thus the hypothesis is the following clause in Amao notation:

```
motherInLaw(X,Y); ~mother(X,Z_0); ~wife(Z_0,Y)
```

This rule is added to the KB and its Shared NeMuS is updated accordingly.

In general this hook chain can be longer and involve more *linkage predicates*. Depending on the position where the linkage is formed from one to another there may be different sorts of *linkage pattern*. Besides, there can be intermediate predicates that should not be part of the hypothesis since they may deduce negative examples. Consider the following example, and this time for the sake of readability the space information will be suppressed from the bindings and the attribute position from x_a.

Example 2. Consider the following BK.

1. `parent(pam,bob).`	6. `parent(pat,jim).`	1. `female(pam).`
2. `parent(tom,bob).`	7. `parent(ann,eve).`	2. `female(liz).`
3. `parent(tom,liz).`	1. `male(tom).`	3. `female(ann).`
4. `parent(bob,ann).`	2. `male(bob).`	4. `female(pat).`
5. `parent(bob,pat).`	3. `male(jim).`	5. `female(eve).`

Codes for the logical elements in the order they are read or scanned are:

parent	male	female	pam	bob	tom	liz	ann	pat	jim	eve
1	2	3	1	2	3	4	5	6	7	8

The induction Amao is requested to perform is

```
consider induction on hasDaughter(X)
                      knowing hasDaughter(ann) ~hasDaughter(pat).
```

First find the bindings associated with the positive and negative examples.

$$+\beta(ann) = [(1,4,2),(1,7,1),(3,3,1)] \text{ and } -\beta(pat) = [(1,5,2),(1,6,1),(3,4,1)]$$

$n_c(\beta(ann)_1) = n_c(\beta(pat)_1) = 1$ and their positions are the same in the predicate attributes, given by $n_a(\beta(ann)_1) = n_a(\beta(pat)_1) = 2$. As both appear along with the same constant *bob* (2).

$$x_a(\beta(ann)_1) = [(0,2,3),(0,5,1)] \text{ and } x_a(\beta(pat)_1) = [(0,2,4),(0,6,1)]$$

This path will give hypotheses which make `hasDaughter(pat)` deducible, which is not desirable since `hasDaughter(pat)`$\in e^-$. Call this an *inconsistent path*, and the instance is dropped and another selected.

$n_c(\beta(ann)_2) = n_c(\beta(pat)_2) = 1$ and $n_a(\beta(ann)_2) = n_a(\beta(pat)_2) = 1$. This time their attribute-mates are different, *eve* (8) for *ann* and *jim* (7) for *pat*

$$x_a(\beta(ann)_2) = [(0,5,2),(0,8,1)] \text{ and } x_a(\beta(pat)_2) = [(0,6,2),(0,7,1)].$$

This splits the path into two branches and it cannot be said, at this stage, if both will lead to atomic sentences belonging to the Herbrand Base, thus deducible. Call this a *plausible path*. From this point the first literal for the body of the hypothesis can be considered as a generalization of `parent(ann, eve)`, i.e. `parent(X, `Z_0`)` (where $\{ann/X, eve/Z_0\}$ is the inverse or anti-unification of terms) has to be confirmed by pruning away any possible predicate found in the bindings of *jim* onwards.

$$+\beta(eve) = [(1,7,2),(3,5,1)] \text{ and } -\beta(jim) = [(1,6,2),(2,3,1)]$$

Their first bindings fail in the same inconsistent path as in the case of *ann* and *pat*, and so they must be dropped. However, their occurrences happen at different predicates as shown by

$$n_c(\beta(eve)_2) = 3 \text{ (for female)} \quad n_c(\beta(jim)_2) = 2 \text{ (for male)}$$

This means that the path has reached a state of positive path only, meaning that predicate 2 ($male$) can be dropped and $female(eve)$ ends the search for this branch. Add to the body the general formula $female(Z_0)$. The search for a hypothesis might stop here and Amao would learn

$$hasDaughter(X) \leftarrow parent(X, Z_0) \wedge female(Z_0)$$

This hypothesis meets the desired definition. If search is continued, further hypotheses might be found such as

$$hasDaughter(X) \leftarrow parent(X, Z_0) \wedge female(Z_0) \wedge female(X)$$

which is also consistent with BK, e^+ and e^-.

Algorithm 1 LinkagePattern(p_k, p_{k1} are T-Nodes)

```
1: if n_c(p_k) = n_c(p_k1) then                          ▷ possible recursive pattern
2:     if x_a(p_k) ∩ x_a(p_k1) ≠ ∅ then
3:         if n_a(p_k) < n_a(p_k1) then      ▷ (position of a_k is less than position of a_k1)
4:             return linear_and_recursive
5:         else if n_a(p_k) = n_a(p_k1) then
6:             return sink_hook
7:         else
8:             return side_hook_pattern
9:     else
10:        if n_a(p_k) < n_a(p_k1) then      ▷ (position of a_k is less than position of a_k1)
11:            return linear_or_recursive
12:        else if n_a(p_k) = n_a(p_k1) then
13:            return deep_sink_hook
14:        else
15:            return long_side_hook
16: else
17:    if x_a(p_k) ∩ x_a(p_k1) = ∅ then
18:        return unknown_hook
19:    else
20:        return short_linear_hook
```

3.2 Recursive Linkage Pattern

Suppose Amao is asked to generate hypotheses with `ancestor(X,Y)` with positive background knowledge `ancestor(pam,jim)`.

```
consider induction on ancestor(X,Y) knowing ancestor(pam,jim).
```

Although there is a long linear linkage from `pam` until `jim`, the successful expected hypotheses should be recursive having $\sim$`parent(X,Y)` as base. Amao generates

these as learned hypotheses by following the same steps as for linear linkage, except that the first pair of p_k and p_{k1} from $\beta(a_k)$ and $\beta(a_{k1})$ is checked for equality. In other words, if $n_c(p_k) \neq n_c(p_{k1})$, then the linkage pattern is linear and proceed as above. Otherwise it is possible that there is a *recursive pattern*. Algorithm 1 diagnoses which linkage pattern to apply (including some not discussed here as they are intended to be used within a neural network learning extension of the current method for large background knowledge). In Algorithm 1 $n_a(p_k)$ and $n_a(p_{k1})$ are the attribute positions for a_k and a_{k1}, respectively.

3.3 Inductive Momentum

The definition of ILP and all implementations use negative examples e^- more as a testing case to check whether a generated hypothesis H plus BK will entail e^-. If so, then the cause of the undesired deduction is detected, fixed and a new hypothesis generated. The approach in this work is different: negative example can be used to prune away candidates to generalized body literals if they have been reached from the bindings of terms from e^- (call them negative terms).

The *inductive momentum* between two T-Nodes a^+ of $\boldsymbol{x}_a^+$ and a^- of $\boldsymbol{x}_a^-$ given partial θ^{-1} is given by Algorithm 2.

Algorithm 2 InductiveMomentum($\boldsymbol{x}_a^+$,a^+,$\boldsymbol{x}_a^-$,a^-)

```
1: if no attribute of x_a^+ has an anti-substitution in θ^-1 then
2:     return useless_path.
3: if n_c(a^+) = n_c(a^-) then
4:     if n_a(a^+) = n_a(a^-) then
5:         if both attribute-mates a^+ of x_a^+ and a^- of x_a^- are equal then
6:             return inconsistent_path
7:         else
8:             return plausible_path
9:     else
10:        plausible_path
11: return positive_path_only
```

4 The Inductive Learning Algorithm

The induction algorithm presented in Algorithm 3 works as a search guided by the linkage pattern (Algorithm 1) and inductive momentum (Algorithm 2). Algorithm 3 is given a shared NeMuS $\mathcal{N}$, a target P with code p/n, and coded positive and negative examples e^+ and e^- with their respective coded terms being a_k^+ (possibly a_{k1}^+), a_k^- (possibly a_{k1}^-) respectively.

Note that step 8 guarantees that ground atoms common to the e^+ and e^- derivation chain are left out of the partial clause hypothesis building process.

Algorithm 3 InductiveLearn($\mathcal{N}$,p_k, p_{k1} are T-Nodes)

1: $\theta_T \leftarrow \theta^{-1}(a_k, a_{k1})$
2: $\beta_k^+ \leftarrow \beta(a_k^+)$ (and $\beta_{k1}^+ \leftarrow \beta(a_{k1}^+)$ if exists a_{k1}^+) from $\mathcal{N}$
3: $\beta_k^- \leftarrow \beta(a_k^-)$ (and $\beta_{k1}^- \leftarrow \beta(a_{k1}^-)$ if exists a_{k1}^-) from $\mathcal{N}$
4: **while** $\beta_k \neq \varnothing$ **do**
5: $H_p \leftarrow \theta_T P$
6: select p_k^+ (p_{k1}^+) p_k^- (p_{k1}^-) from β_k^+ (β_{k1}^+) and β_k^- (β_{k1}^-)
7: **if** There exists a useful LinkagePattern(p_k^+, p_{k1}^+) **then**
8: **if** InductiveMomentum($x_a(n_c(p_k^+)), n_a(p_k^+)), x_a(n_c(p_k^-)), n_a(p_k^-)$) **then**
9: **if** linkage is `linear_and_recursive` **then**
10: $\theta_b \leftarrow \theta^{-1}(x_a(n_c(p_k^+))) \cup clone(\theta_T)$
11: $H_b \leftarrow \theta_b(clone(H_p) \cup \sim p_k)$
12: $\theta_r \leftarrow \theta^{-1}(x_a(n_c(p_{k1}^+))) \cup \theta_T$
13: $H_r \leftarrow \theta_r(clone(H_p) \cup \sim p_{k1})$
14: **else**
15: $\theta_1 \leftarrow \theta^{-1}(clone(\theta_T), x_a(n_c(p_k^+)))$
16: $\theta_2 \leftarrow \theta^{-1}(\theta_1, x_a(n_c(p_{k1}^+)))$
17: $H_p \leftarrow \theta_2(clone(H_p) \cup \sim p_k \cup \sim p_{k1})$
18: **if** $\theta_1 = \theta_2$ **then**
19: save hypotheses generated
20: update βs with bindings from attribute-mates(linkage terms).

This avoids inconsistency by not allowing the generation of hypotheses that would satisfy e^- along with BK.

Consider again the example with `ancestor/2` to give an intuitive idea of how negative examples are used as an *inductive momentum*. Everything else is left out of the hypothesis since it will not be part of the HB that satisfies the positive examples e^+ plus BK and hypothesis. The process builds the hypothesis by selecting from the intersection those that meet one of the linkage patterns, and are not eliminated as a result of an inductive momentum.

```
>> consider induction on ancestor(X,Y) knowing ancestor(pam,jim).
--> Consider using these hypotheses...
ancestor(X,Y); ~parent(X,Y).
ancestor(X,Y); ~parent(X,Z0); ~ancestor(Z0,Y).
```

5 Related Work

Inductive logic programming has a large literature, from its antecedents in inductive generalization [14], through work on search strategies, the logical framework that the learning sits in, as well as the building of systems and application of these to specific problems. Inductive learning continues to be of interest in a wide range of contexts and applications as recently surveyed in [5].

Of particular relevance is the work in [4] that investigates path based algorithms to generate relationships and [15] that uses inductive logic programming

concepts in an early instance of theory repair (that is, revising a theory that is incorrect so that the counter-examples are no longer such). Additionally [6], that investigates variations on the standard anti-unification algorithm and how these impact on the efficiency of hypothesis search, is of interest in the current context. More recently, in [1] boolean constraints are used to describe undesirable areas of the hypothesis search space and solving these to prune the search space achieves significant speed ups over older inductive logic programming systems on a range of examples, whilst retaining accuracy.

The higher-order approach taken in [11, 12] uses a meta-interpreter with iterative deepening to build Metagol. Metagol has had success on a range of examples, including learning a subclass of context-free grammars from examples and inferring relationships in sports data (whilst also uncovering an error in the data representation). This includes predicate invention, a topic that these papers suggest has not been paid due attention in inductive learning.

6 Discussion and Future Work

This paper has shown how the Amao Shared NeMuS data structure can be used to build an inductive logic programming system which has been successfully applied on some small trial examples.

The results on inductive learning in Amao show that using its shared structure leads to reliable hypothesis generation in the sense that the minimally correct ones are generated. However, it still generates additional hypothesis, logically sound and correct with respect to the Herbrand base derivation. Most important is the size of the set of hypotheses generated which is small in comparison with the literature, e.g. [3].

Future work will focus on two areas. First, the power of the shared structure to allow a fast implementation of inductive inference. Second, the weights incorporated in the Shared NeMuS structure (not used in the current paper) will be used to play an important role in providing heuristics. In [9] it is shown how these weights can be updated in a manner inspired by self-organising maps [7]. The propagation across the network of nodes in the structure allows the weights to capture patterns of refutation. It should be possible to capture negative examples in inductive logic programming in the network in this way, guiding search away from these unfruitful regions to fine tune to a small set of generated hypotheses. Alongside improved hypothesis search the use of the weighted structure to drive predicate invention – to add to the knowledge base additional inferred predicates contained in neither it nor the target predicate – will be investigated.

Acknowledgement

The authors would like to thank Gerson Zaverucha for fruitful discussion and guidance through emails about Inductive Logic Programming.

References

1. Ahlgren, J., Yuen, S.Y.: Efficient Program Synthesis Using Constraint Satisfaction in Inductive Logic Programming. Journal of Machine Learning Research 14, 3649–3681 (2013)
2. Boyer, R.S., Moore, J.S.: The Sharing of Structure in Theorem-Proving Programs. In: Machine Intelligence 7. pp. 101–116. Edinburgh University Press (1972)
3. Bratko, I.: Prolog Programming for Artificial Intelligence, 4th edition. Addison Wesley (2011)
4. Bunescu, R.C., Mooney, R.J.: A Shortest Path Dependency Kernel for Relation Extraction. In: Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing. pp. 724–731. Association for Computational Linguistics (2005)
5. Gulwani, S., Hernández-Orallo, J., Kitzelmann, E., Muggleton, S.H., Schmid, U., Zorn, B.G.: Inductive Programming Meets the Real World. Communications of the ACM 58(11), 90–99 (2015)
6. Idestam-Almquist, P.: Generalization under Implication by Recursive Anti-unification. In: International Conference on Machine Learning. pp. 151–158. Morgan-Kaufmann (1993)
7. Kohonen, T.: Self-Organizing Maps. Springer, 3rd edn. (2001)
8. Mao, L.: An introduction to Smarandache multi-spaces and mathematical combinatorics. Scientia Magna 3(1), 54–80 (2007)
9. Mota, E.d.S., Diniz, Y.B.: Shared Multi-Space Representation for Neural-Symbolic Reasoning. In: Besold, T.R., Lamb, L., Serafini, L., Tabor, W. (eds.) NeSy 2016. vol. 1768. CEUR Workshop Proceedings (July 2016)
10. Muggleton, S.H.: Inductive Logic Programming. New Generation Computing 8(4), 295–318 (1991)
11. Muggleton, S.H., Lin, D., Pahlavi, N., Tamaddoni-Nezhad, A.: Meta-interpretive learning: application to grammatical inference. Machine Learning 94(1), 25–49 (2014)
12. Muggleton, S.H., Lin, D., Tamaddoni-Nezhad, A.: Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited. Machine Learning 100(1), 49–73 (2015)
13. Nienhuys-Cheng, S.H., De Wolf, R.: Foundations of Inductive Logic Programming, Lecture Notes in Artificial Intelligence, vol. 1228. Springer (1997)
14. Plotkin, G.D.: A Note on Inductive Generalization. In: Machine Intelligence 5. pp. 153–164. Edinburgh University Press (1969)
15. Richards, B.L., Mooney, R.J.: Automated Refinement of First-Order Horn-Clause Domain Theories. Machine Learning 19(2), 95–131 (1995)
16. Robinson, A.: A machine-oriented logic based on the resolution principle. Journal of the ACM 12(1), 23–42 (1965)