AN ALGEBRAIC APPROACH TO INDUCTIVE LEARNING

ZDRAVKO MARKOV
Department of Computer Science, Central Connecticut State University
1615 Stanley Street, New Britain, CT 06050, USA
E-mail: markovz@ccsu.edu

Received 7/7/00
Revised 10/27/00

The paper presents a framework to induction of concept hierarchies based on consistent integration of metric and similarity-based approaches. The hierarchies used are subsumption lattices induced by the least general generalization operator (lgg) commonly used in inductive learning. Using some basic results from lattice theory the paper introduces a semantic distance measure between objects in concept hierarchies and discusses its applications for solving concept learning and conceptual clustering tasks. Experiments with well known ML datasets represented in three types of languages - propositional (attribute-value), atomic formulae and Horn clauses, are also presented.

Keywords: Metrics, Concept Learning, Conceptual clustering, Machine Learning.

1. Introduction

Inductive learning addresses mainly classification tasks where a series of training examples (instances) are supplied to the learning system and the latter builds an intensional or extensional representation of the examples (hypothesis). The approaches to inductive learning are based mainly on generalization/specialization or similarity-based techniques. Two types of systems are considered here – *concept learning* and *conceptual clustering*. They both generate inductive hypotheses made by abstractions (generalizations) from specific examples and differ in the way examples are presented to the system (whether or not they are pre-classified). The hypotheses generated by these systems usually form a partially ordered set under some generality ordering. The properties of partially ordered sets are well studied in lattice theory. One concept from this theory is mostly used in inductive learning – this is the *least general generalization (lgg)* which given two hypotheses builds their most specific common generalization. The existence of an *lgg* in a hypothesis space implies that this space is a semi-lattice (the lgg plays the role of infimum).

The idea behind the lgg is to make "cautious" (minimal) generalization. However this property of the lgg greatly depends on how similar are the hypotheses/examples used to build the lgg. For example there exist elements in the hypothesis space whose lgg is the top element (empty hypothesis).

An obvious solution of the latter problem is to use a distance (metric) over the hypothesis/example space in order to evaluate the similarity between the hypotheses/examples. The basic idea is when building an lgg to choose the pair of hypotheses/examples with the minimal distance between them in order to produce the *minimal lgg*. Thus the problem is *to find a distance measure which is both well coupled with the lgg and account for the background knowledge and for the coverage of positive/negative examples.* This is the main problem we address in the present paper.

Hereafter we propose a consistent way to integrate a syntactical lgg with semantic evaluation of the hypotheses. For this purpose we use two different relations on the hypothesis space – a constructive one, used to generate lgg's and a semantic one giving the coverage-based evaluation of the lgg. These two relations jointly implement a *semantic distance measure.* The formal background for this is a height-based definition of a semi-distance on join semi-lattices. We use some basic results from lattice theory and introduce a language independent coverage-based height function. We also define the necessary conditions for two relations to form a correct height function.

The paper is organized as follows. The next section outlines the basic notions from lattice theory used throughout the paper. Section 3 introduces the new height-based semi-distance and Section 4 discusses its applications in a series of languages commonly used in ML. Section 5 discusses related work and Section 6 outlines the directions for future work.

2. Preliminaries

In this section we introduce a height-based distance measure on a join semi-lattice. We use some basic notions from lattice theory with some modifications and extensions (for a survey of metrics on partially ordered sets see [1]).

Definition 1 (Semi-distance, Quasi-metric). A *semi-distance (quasi-metric)* is a mapping $d : O \times O \to \Re$ on a set of objects O with the following properties $(a, b, c \in O)$:

1. $d(a, a) = 0$ and $d(a, b) \geq 0$.
2. $d(a, b) = d(b, a)$ (symmetry).
3. $d(a, b) \leq d(a, c) + d(c, b)$ (triangle inequality).

Definition 2 (Order preserving semi-distance). A semi-distance $d : O \times O \to \Re$ on a partially ordered set $(O, \preceq)$ is *order preserving* iff for all $a, b, c \in O$, such that $a \preceq b \preceq c$ it follows that $d(a, b) \leq d(a, c)$ and $d(b, c) \leq d(a, c)$

Definition 3 (Join/Meet semi-lattice). A *join/meet semi-lattice* is a partially ordered set $(A, \preceq)$ in which every two elements $a, b \in A$ have an infimum/supremum.

Definition 4 (Diamond inequality). Let $(A, \preceq)$ be a join semi-lattice. A semi-distance $d : A \times A \rightarrow \Re$ satisfies the *diamond inequality* iff the existence of $sup\{a, b\}$ implies the following inequality: $d(inf\{a, b\}, a) + d(inf\{a, b\}, b) \leq d(a, sup\{a, b\}) + d(b, sup\{a, b\})$.

Definition 5 (Size function). Let $(A, \preceq)$ be a join semi-lattice. A mapping $s : A \times A \rightarrow \Re$ is called a *size function* if it satisfies the following properties:

S1. $s(a, b) \geq 0, \forall a, b \in A$ and $a \preceq b$.

S2. $s(a, a) = 0, \forall a \in A$.

S3. $\forall a, b, c \in A$, such that $a \preceq c$ and $c \preceq b$ it follows that $s(a, b) \leq s(a, c) + s(c, b)$ and $s(c, b) \leq s(a, b)$.

S4. Let $c = inf\{a, b\}$, where $a, b \in A$. For any $d \in A$, such that $a \preceq d$ and $b \preceq d$ it follows that $s(c, a) + s(c, b) \leq s(a, d) + s(b, d)$.

Consider for example the partially ordered set of first order atoms under θ-subsumption. A size function $s(a, b)$ on this set can be defined as the number of different functional symbols (a constant is considered a functional symbol of arity zero) occurring in the substitution θ mapping a onto b ($a\theta = b$). A family of similar size functions is introduced in [2], where they are called a *size of substitution*. Although well defined these functions do not account properly for the variables in the atoms and consequently cannot be used with non-ground atoms.

Theorem 1. Let $(A, \preceq)$ be a join semi-lattice and s – a size function. Then the function $d(a, b) = s(inf\{a, b\}, a) + s(inf\{a, b\}, b)$ is a *semi-distance* on $(A, \preceq)$.

Proof.

1. d is non-negative by $S1$ and $d(a, a) = s(inf\{a, a\}, a) + s(inf\{a, a\}, a) = s(a, a) + s(a, a) = 0$.

2. d is symmetric by definition.

3. We will show that $d(a_1, a_2) \leq d(a_1, a_3) + d(a_3, a_2)$. Let $c = inf\{a_1, a_2\}$, $b_1 = inf\{a_1, a_3\}$, $b_2 = inf\{a_2, a_3\}$, $d = inf\{b_1, b_2\}$ (Figure 1). By $S3$ we have $s(c, a_1) \leq s(d, a_1) \leq s(d, b_1) + s(b_1, a_1)$. And by analogy $s(c, a_2) \leq s(d, b_2) + s(b_2, a_2)$. Then $d(a_1, a_2) = s(c, a_1) + s(c, a_2) \leq s(d, b_1) + s(b_1, a_1) + s(d, b_2) + s(b_2, a_2) \leq s(b_1, a_1) + s(b_1, a_3) + s(b_2, a_3) + s(b_2, a_2) = d(a_1, a_3) + d(a_2, a_3)$. $\square$

A widely used approach to define a semi-distance is based on an order preserving size function and the diamond inequality instead of property $S4$. The use of property $S4$ however is more general because otherwise we must assume that (1) all intervals

$$\begin{tikzcd}
& d \arrow[dl] \arrow[dr] & \\
b_1 \arrow[dr] \arrow[dd] \arrow[ddr] & c \arrow[ddl] \arrow[ddr] & b_2 \arrow[dl] \arrow[ddl] \arrow[dd] \\
& e & \\
a_1 & a_3 & a_2
\end{tikzcd}$$

Fig. 1. A semi-lattice structure

in the lattice are finite and (2) if two elements have an upper bound they must have a least upper bound (supremum) too. An illustration of this problem is shown in Figure 1, where a_3 is an upper bound of b_1 and b_2 and $e = sup\{b1, b2\}$. Generally the interval $[e, a_3]$ may be infinite or e may not exist. This however does not affect our definition of semi-distance.

Further, a size function can be defined by using the so called *height functions*. The approach of height functions have the advantage that it is based on estimating the object itself rather than its relations to other objects.

Definition 6 (Height function). A function h is called *height* of the elements of a partially ordered set $(A, \preceq)$ if it satisfies the following two properties:

H1. For every $a, b \in A$ if $a \preceq b$ then $h(a) \leq h(b)$ (isotone).

H2. For every $a, b \in A$ if $c = inf\{a, b\}$ and $d \in A$ such that $a \preceq d$ and $b \preceq d$ then $h(a) + h(b) \leq h(c) + h(d)$.

Theorem 2. Let $(A, \preceq)$ be a join semi-lattice and h be a height function. Let $s(a, b) = h(b) - h(a), \forall a \preceq b \in A$. Then s is a *size function* on $(A, \preceq)$.

Proof.

1. $s(a, b) = h(b) - h(a) \geq 0$ by $H1$.
2. $s(a, a) = h(a) - h(a) = 0$.
3. Let $a, b, c \in A$, such that $a \preceq c, c \preceq b$. Then $s(a, b) = h(b) - h(a) = (h(b) - h(c)) + (h(c) - h(a)) = s(a, c) + s(c, b)$.
4. Let $a, b, c \in A : a \preceq c, c \preceq b$. Then $s(c, b) \leq s(c, b) + s(a, c) = s(a, b)$ by 3.
5. Let $c = inf\{a, b\}$ and $d \in A$, such that $a \preceq d$ and $b \preceq d$. Then $s(c, a) + s(c, b) = (h(a) - h(c)) + (h(b) - h(c)) = h(a) + h(b) - 2h(c) = 2(h(a) + h(b)) - h(a) - h(b) -$

$2h(c) \leq 2(h(c)+h(d)) - h(a) - h(b) - 2h(c) = (h(d) - h(a)) + (h(d) - h(b)) = s(a,d) + s(b,d)$. $\square$

Corollary 1. Let $(A, \preceq)$ be a join semi-lattice and h be a height function. Then the function $d(a,b) = h(a) + h(b) - 2h(inf\{a,b\}), \forall a,b \in A$ is a *semi-distance* on $(A, \preceq)$.

3. Semantic semi-distance on join semi-lattices

In this section we introduce a family of *semantic* height functions which are later used to define the corresponding semi-distance. The basic idea originates from the notion of ground coverage in first order languages. We generalize this notion and develop a universal language independent coverage-based height function.

Let A be a set of objects and let $\preceq_1$ and $\preceq_2$ be two binary relations on A. Let also $\preceq_1$ be a partial ordering and $(A, \preceq_1)$ – a join semi-lattice.

Definition 7 (Ground elements of a join semi-lattice (GA)). GA is the set of all maximal elements of A w.r.t. $\preceq_1$, i.e. $GA = \{a | a \in A \text{ and } \neg\exists b \in A : a \preceq_1 b\}$.

Definition 8 (Ground coverage). For every $a \in A$ the *ground coverage* of a w.r.t $\preceq_2$ is $S_a = \{b | b \in GA \text{ and } a \preceq_2 b\}$.

The ground coverage S_a can be considered as a definition of the semantics of a. Therefore we call $\preceq_2$ a *semantic relation* by analogy to the Herbrand interpretation in first order logic used to define the semantics of a given term. The other relation involved, $\preceq_1$ is called *constructive (or syntactic) relation* because it is used to build the lattice from a given set of ground elements GA.

The basic idea of our approach is to use these two relations, $\preceq_1$ and $\preceq_2$ to define the semi-distance according to Corollary 1. We use the syntactic relation $\preceq_1$ to find the infimum and the semantic relation $\preceq_2$ to define the height function h. This allows us to define a proper distance measure even when the semantic relation is intractable, computationally expensive or even not a partial order, i.e. when it is impossible to use it as a constructive relation too (an example of such a relation is logical implication).

Not any two relations however can be used for this purpose. The following theorem states the necessary conditions for two relations to form a correct height function.

Theorem 3. Let A be a set of objects and let $\preceq_2$ and $\preceq_1$ be two binary relations in A such that:

1. For every $a, b \in A$ if $a \preceq_1 b$ then $|S_a| \geq |S_b|$[*]

2. For every $a, b \in A$ and $c = inf\{a,b\}$ such that there exists $d = sup\{a,b\}$ one of the following must hold:

[*]Generally an isotone property is required here. However we skip the other case, $|S_a| \leq |S_b|$ since it is analogous.

C1. $|S_d| < |S_a|$ and $|S_d| < |S_b|$

C2. $|S_d| = |S_a|$ and $|S_c| = |S_b|$

C3. $|S_d| = |S_b|$ and $|S_c| = |S_a|$

Then there exists a family of *height functions* $h(a) = x^{-|S_a|}$, where $a \in A$, $x \in \Re$ and $x \geq 2$.

Proof.

1. Let $a, b \in A$, $a \preceq_1 b$. Then by the assumptions $|S_a| \geq |S_b|$ and hence $h(a) \leq h(b)$.

2. Let $a, b \in A$, $c = inf\{a, b\}$ and $d = sup\{a, b\}$.

   (a) Assume that C1 is true. Then $|S_d| < |S_a|$ and $|S_d| < |S_b| \Rightarrow |S_a| \geq |S_d| + 1$ and $|S_b| \geq |S_d| + 1 \Rightarrow -|S_a| \leq -|S_d| - 1$ and $-|S_b| \leq -|S_d| - 1$. Hence $h(a) + h(b) = x^{-|S_a|} + x^{-|S_b|} \leq x^{-|S_d|-1} + x^{-|S_d|-1} = 2x^{-|S_d|-1} \leq x.x^{-|S_d|-1} = x^{-|S_d|} = h(d) \leq h(c) + h(d)$.

   (b) Assume that C2 is true. Then $|S_d| = |S_a|$ and $|S_c| = |S_b|$. Hence $h(a) + h(b) = h(c) + h(d)$.

   (c) Assume that C3 is true. Then $|S_d| = |S_b|$ and $|S_c| = |S_a|$. Hence $h(a) + h(b) = h(c) + h(d)$. $\square$

4. Applications

4.1. *General algorithm*

In this section we outline an algorithm for building concept hierarchies based on the ideas discussed in the previous sections. Various versions of this algorithm are reported elsewhere [3,4]. Using a given set of examples E the algorithm builds a concept hierarchy G (partial semi-lattice), where E is the set of all maximal elements of G. The algorithm is as follows ($\preceq_2$ is the semantic relation according to Theorem 3):

1. Initialization: $G = E$, $C = E$;

2. If $|C| = 1$ then exit;

3. $T = \{h | h = lgg_\theta(a, b), (a, b) = argmin_{a,b \in C} d(a, b)\}$;

4. $T = T \setminus \{h | h \in T, \exists a \in C_1, \exists b \in C_2, C_1 \neq C_2, h \preceq_2 a, h \preceq_2 b\}$ (for concept learning only);

5. $DC = \{a | a \in C$ and $\exists h \in T : h \preceq_2 a\}$;

6. $C = C \setminus DC$;

7. $G = G \cup T$, $C = C \cup T$, go to step 2.

The *computational complexity* of the algorithm can be estimated as follows. The most expensive step in the algorithm is the computation of T at step 3. Generally there are two possibilities:

1. $|T| = 1$. Then in the worst case DC in step 5 will contain only two elements (a and b from which T is generated in step 3). Thus the number of elements in C decreases by 1 at every step and assuming that $|E| = n$ the outermost loop of the algorithm executes $n - 1$ times. If the set C contains k elements then there are $k(k-1)/2$ different pairs for which lgg's must be calculated. As n is an upper bound of k, the complexity of the algorithm is $O(n^3)$.

2. If $|T| > 1$ then C may grow and the complexity of the algorithm becomes much higher. Generally the number of elements in T depends on the quality of the distance measure used and the number of examples E. If we use a more sophisticated distance measure the chance to have more than one pair of elements with the same distances between them is lower. In practical experiments we always choose a single element from T. For this purpose some heuristics can be applied too.

The algorithm can be used to solve both *concept learning* and *conceptual clustering* tasks. For conceptual clustering, step 4 is omitted and in this case G represents the concept hierarchy, where the successors of the root represent the splitting of the initial set of examples E. For concept learning, the class membership for the examples is used in step 4 to prevent the generation of incorrect hypotheses. The construction of the lgg and the computation of the distance function d are language specific and will be discussed in the following subsections.

4.2. *Attribute-value (propositional) language*

An example or a hypothesis in this language is a conjunction of propositions (attribute-value pairs) which can also be represented as a set of propositions. The partial ordering $\preceq$ here is the set inclusion $\subseteq$ between hypotheses (examples). The lgg (infimum) of two elements is defined as their intersection, i.e. $lgg(a,b) = a \cap b$.

Two types of height functions can be used here. The first one is $h(a) = |a|$. This function implements the most common way of representing generality of hypotheses with nominal attributes - the so called dropping condition. Thus a *syntactic* distance measure can be defined as $d(a,b) = 2^{-|a|} + 2^{-|b|} - 2 \times 2^{-|a \cap b|}$.

The *semantic* distance introduced in the previous sections can be defined by using a *coverage* function $cov(a) = |\{b | b \in E, a \subseteq b\}|$, where E is the set of examples. Then according to Theorem 3 (relation $\subseteq$ plays the role of both relations $\preceq_1$ and $\preceq_2$) we have $d(a,b) = 2^{-cov(a)} + 2^{-cov(b)} - 2 \times 2^{-cov(|a \cap b|)}$.

In this framework we discuss two experiments both performed with the semantic distance. The first one is with the well known MONK1 dataset [5]. We use only the

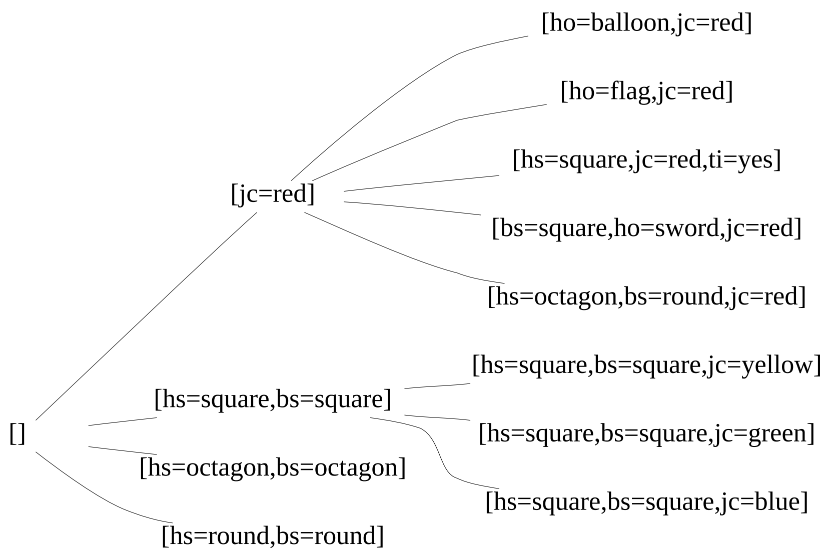
[]
[jc=red]
[hs=square,bs=square]
[hs=octagon,bs=octagon]
[hs=round,bs=round]
[ho=balloon,jc=red]
[ho=flag,jc=red]
[hs=square,jc=red,ti=yes]
[bs=square,ho=sword,jc=red]
[hs=octagon,bs=round,jc=red]
[hs=square,bs=square,jc=yellow]
[hs=square,bs=square,jc=green]
[hs=square,bs=square,jc=blue]

Fig. 2. Propositional concept hierarchy for MONK1 dataset

61 positive examples from the training sample (122 examples). As all the examples are of a single class we apply conceptual clustering (excluding step 4). The top two levels of the concept hierarcy produced by the algorihm are shown in Figure 2. We test each concept (node) in this hierarcy against all 122 examples in the training set. The pair of numbers in square brackets at each node show the distribution of positive/negative examples covered by the node. As all concepts at the second level of the hierarchy (enclosed in boxes in Figure 2) do not cover any negative examples we use them as a solution for the concept learning task. They in fact describe exactly the target theory and thus provide 100% accuracy.

The other experiment we outline here is with the DNA promoter sequence database obtained from the UCI ML repository ([6]). This is a domain mainly used as a testbed for ML systems integrating empirical and analytical learning. The dataset describes a specific property of DNA called promoter. The learning task is to identify the sequence of nucleotides that exhibit this property. 106 examples are provided (53 positive and 53 negative) each one describing a sequence of 57 nucleotides. Each attribute specifies the nucleotide (a, g, t or c) occurring at the corresonding position is the sequence. We ran the algorithm with 43 randomly chosen positive examples (the set E). The concept hierarchy obtained is shown in Figure 3. The nodes show the class distribution (positive/negative) of the examples covered by the node concept. A promoters theory can be generated by choosing some of the nodes in this hierarchy. The choice is based on the coverage of negative examples. By using a threshold 2 and then eliminating all concepts covered by others we come up with 4 concepts (enclosed in boxes in Figure 3). This theory covers

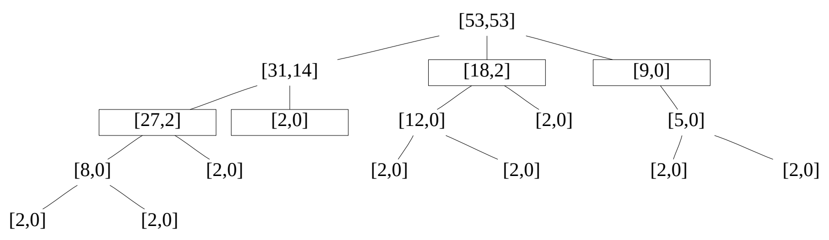

Fig. 3. The promoters data concept hierarchy. The nodes show the class distribution (positive/negative) of the examples covered by the node concept. The enclosed in boxes concepts provide 92% accuracy

4 negative and 49 positive examples thus providing 92% accuracy. Similar accuracies are reported in [7] where a number of ML systems are tested on this dataset. It is important to note however that these systems use negative examples as well. We also tested our algorithm as concept learner with negative examples (applying step 4) and got similar results. The conceptual clustering approach however is more general and provides more information about the generated concepts.

4.3. *Atomic formulae*

The language of the first order atomic formulae can be seen as an intermediate step between the propositional and the full relational language (e.g. Horn clauses). Its main advantage with respect to the propositional language is the possibility to define explicitly the *equality* of attribute values. The algebraic properties of this language are studied in [8], where the author shows that the set of atoms with same functors and arity augmented by adding a 'universal atom' and a 'null atom' forms a *complete lattice*. In this lattice the partial ordering is the *instance relation* ($\geq$) and the meet and the join operations are the well known *greatest common instance* (obtained by unification) and *least general generalization* (*lgg*, obtained by anti-unification).

The instance relation between atoms can play the role of both syntactic and semantic relation according to Theorem 3. Thus we can define the distance function $d(a,b) = 2^{-cov(a)} + 2^{-cov(b)} - 2 \times 2^{-cov(|a \cap b|)}$, where $cov(a) = |\{b | b \in E, a \geq b\}|$ and E is the set of examples.

Other height functions can also be used in this language. In [8] a function evaluating the generality of atoms is proposed. It is called $size(A)$ (note that this is not the formal size function according to Definition 5) and is defined as the number of symbol occurrences in A minus the number of distinct variables occurring in A. $A \geq B$ implies $size(B) \geq size(A)$ and $A \simeq B$ implies $size(A) = size(B)$. Also, for any A and B there is no chain from A to B whose length is greater than $size(B) - size(A)$. Unfortunately the Reynolds' *size* function does not satisfy the second formal property of a height function. Nevertheless we used this function in our experiments and got satisfactory results especially when the set of examples

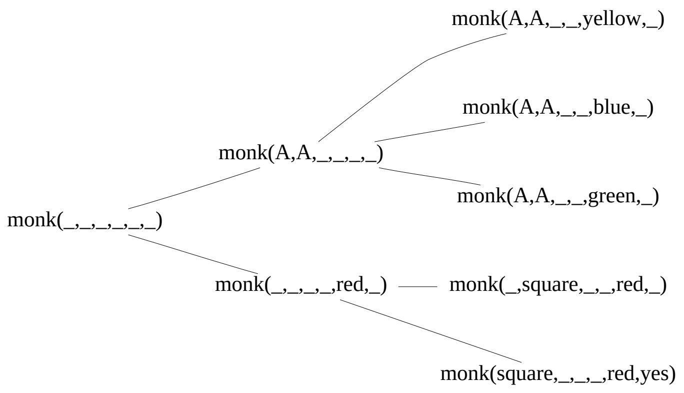

Fig. 4. Relational concept hierarchy for the MONK1 dataset

E is small (in this case the coverage function cannot provide good estimate of the generality of the hypotheses).

A simplified version of the Reynolds' *size* function is proposed in [2]. It is based on the number of functional symbols in the atom. Though formally a height, this function does not account properly for the variables in the atoms and consequently gives poor results especially at the higher levels of the concept hierarchy where more variable appear.

In the framework of atomic formulae we discuss two experiments with our algorithm. They are performed with the semantic distance and also with the Reynolds' size function and show similar results (the Hutchinson's function performed poorly).

Figure 4 shows the top portion of the concept hierarchy for the MONK1 dataset using 61 positive examples as in the propositional case. The examples are encoded as atomic formulae (for example $[hs = octagon, bs = octagon, sm = no, ho = sword, jc = red, ti = yes]$ is represented as the atom $monk(octagon,\ octagon,\ no,\ sword,\ red,\ yes)$). The generalizations are implemented by replacing constants by variables (antiunification) (_ is the anonymous variable that matches everything). To identify the target concept we use the same technique as in the propositional case - filtering out nodes based on a threshold for the negative coverage. Thus the two successors of the root in Figure 4 form the original MONK1 theory which is 100% correct.

Figure 5 shows another concept hierarchy generated from a set of instances of the popular *append* predicate defining list concatenation. The leaves of the concept hierarchy show the set of example E generated from the append theory (for example $append([1,2],[3,4],[1,2,3,4])$ means that appending lists $[1,2]$ and $[3,4]$ results in $[1,2,3,4]$). The two immediate successors of the root ($app([A|_{]},, [A|_{]})$ and $app([], A, A)$) split E into two classes that reflect the two major cases in the append theory (the base case and the recursive clause). Thus they provide 100% accuracy with respect to this theory.

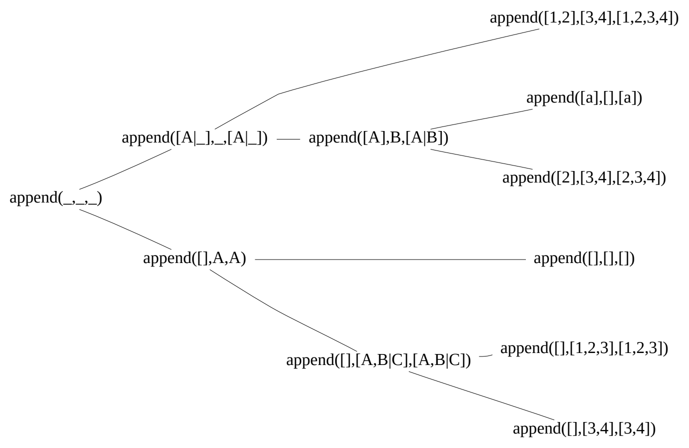

Fig. 5. Concept hierarchy built upon instances of the *append* predicate.

4.4. *Horn clauses*

A clause C is a disjunction (or set) of literals, i.e. $C = L_1 \vee L_2 \vee ... \vee L_n$, where L_i is an atom or negated atom. The language of Horn clauses restricts the number of positive literals (not negated atoms) to at most one. This allows us to use another notation for Horn clauses used also in the language Prolog, which is $A : -B_1, B_2, ..., B_{n-1}$, where A is the positive one and $B_1, B_2, ..., B_{n-1}$ are all negative literals among $L_1\ lor L_2 \vee ... \vee L_n$.

Within the language of Horn clauses we use *θ-subsumption-based lgg* (the constructive relation $\preceq_1$) and *logical implication* (semantic entailment) for the semantic relation $\preceq_2$.

Definition 9 (θ-Subsumption). A clause a θ-subsumes clause b, denoted $a \preceq_\theta b$, if there exists a substitution θ, such that $a\theta \subseteq b$.

Under θ-subsumption the set of Horn clauses with same heads is a semi-lattice. Hereafter we show that θ-subsumption and logical implication can be used to define correct height function on this semi-lattice.

Definition 10 (Model). A set of ground literals which does not contain a complementary pair is called a *model.* Let M be a model, c – a clause, and C – the set of all ground clauses obtained by replacing the variables in c by ground terms. M is a model of c iff each clause in C contains at least one literal from M.

Definition 11 (Semantic entailment). Let f_1 and f_2 be well-formed formulae. f_1 *semantically entails* f_2, denoted $f_1 \models f_2$ (or $f_1 \preceq_{\models} f_2$) iff every model of f_1 is a model of f_2.

Corollary 2. Let a and b be clauses such that $a \preceq_\theta b$. Then $S_a \supseteq S_b$ and $|S_a| \geq |S_b|$.

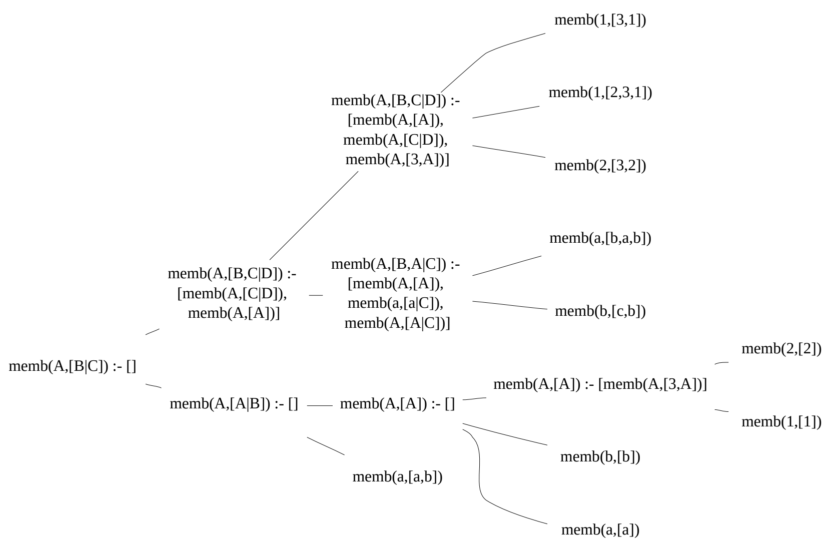
memb(1,[3,1])
memb(A,[B,C|D]) :- [memb(A,[A]), memb(A,[C|D]), memb(A,[3,A])]
memb(1,[2,3,1])
memb(2,[3,2])
memb(a,[b,a,b])
memb(A,[B,C|D]) :- [memb(A,[C|D]), memb(A,[A])]
memb(A,[B,A|C]) :- [memb(A,[A]), memb(a,[a|C]), memb(A,[A|C])]
memb(b,[c,b])
memb(2,[2])
memb(A,[B|C]) :- []
memb(A,[A]) :- [memb(A,[3,A])]
memb(A,[A|B]) :- []
memb(A,[A]) :- []
memb(1,[1])
memb(b,[b])
memb(a,[a,b])
memb(a,[a])

Fig. 6. Concept hierarchy built upon instances of the *member* predicate.

Proof. Let a and b be clauses and let a θ-subsumes b. According to Definitions 10 and 11 a semantically entails b, i.e. $a \preceq_{\models} b$. Then according to Definition 8 $S_a \supseteq S_b$ and $|S_a| \geq |S_b|$. $\Box$

Now we will show that the two assumptions of Theorem 3 hold:

1. Let a and b be clauses and let $a \preceq_1 b$. Then by Corollary 2 $|S_a| \geq |S_b|$.

2. Let $d = sup\{a, b\}$ w.r.t. $\preceq_\theta$. Then $a \preceq_\theta d$, $b \preceq_\theta d$, and by Corollary 2 $|S_d| \leq |S_a|$ and $|S_d| \leq |S_b|$. Further, we will show that actually $|S_d| < |S_a|$ and $|S_d| < |S_b|$. First, we assume that for any two clauses c_1 and c_2 if $S_{c_1} \equiv S_{c_2}$ then $c_1 \equiv c_2$. Thus, in fact instead of clauses we use *equivalence classes* of clauses w.r.t. $\preceq_{\models}$. Let $x \in S_a \triangle S_b$ (symmetric difference). Assume now that $x \in S_d$. Then by Corollary 2 $S_d \subseteq S_a$ and $S_d \subseteq S_b$, that is $x \in S_a \cap S_b$ which is a contradiction. Hence $x \notin S_d$, i.e. $S_d \subset S_a$ and $S_d \subset S_b$. $\Box$

For illustration of our approach within the domain of Horn clauses we use 10 instances of the *member* predicate and supply them to our algorithm. Figure 6 shows the complete concept hierarchy built upon this set of examples. The two successors of the top element form the well-known definition of the *member* predicate (the recursive clause contains a redundant literal).

A major problem in applying our algorithm to Horn clauses is the *clause reduction.* This is because although finite the length of the lgg_θ of n clauses can grow exponentially with n. Some techniques of avoiding this problem are proposed in [9]. By placing certain restrictions on the hypothesis language the number of literals in the lgg_{θ} clause can be limited by a polynomial function. For the above experiment we use *ij-determinate* clauses (actually 22-determinate).

5. Related work

The algebraic approach to inductive learning is a very natural way to study the generalization and specialization hierarchies in ML. These hierarchies represent hypothesis spaces that in most cases are partially ordered sets under some generality ordering. One of the first and most popular works within this approach is the Version Space framework [10]. In this framework the space of all correct conjunctive hypotheses is maintained by using the boundary sets S and G, representing correspondingly the most specific and most general hypotheses. The version space is actually an equivalence class of hypotheses with respect to the inductive task conditions, i.e. covering all positive examples and no negative ones. Thus the goal of the system is by acquiring more examples to reduce this class eventually to a single hypothesis.

In the presence of background knowledge and in case of more complex hypothesis languages usually the more general approach of *refinement operators* is applied. The refinement operators are constructive means to build generalizations or specializations of hypotheses with respect to some generality ordering. In contrast to the Version Space approach refinement operators are used to search the hypothesis space containing not only correct (not covering negative examples) and complete (covering all positive examples) hypotheses. Thus is the case of top-down refinement the system starts from the most general hypothesis and further specializes it in order to avoid covering of negative examples. Conversely the upward refinement operators are used to generalize an initial too specific hypothesis in order to ensure that it covers as many as possible positive examples. The first study of refinement operators is [11], where the so called Model Inference System is introduced. This system performs downward refinement of clauses based on θ-subsumption ordering. An in-depth overview of the refinement operators used in inductive logic programming can be found in [12].

Other refinement operators used in ML are those that take as input two hypotheses and produce their common generalization or specialization. The most popular among these is the *least general generalization (lgg)* operator which given two hypotheses build their most specific common generalization. The existence of an *lgg* in a hypothesis space (a partially ordered set) directly implies that this space is a semi-lattice. Consequently some algebraic notions as finiteness, modularity, metrics etc. can be used to investigate the properties of the hypothesis space. A complete study of least generalizations and greatest specializations within the language of clauses can be found in [14].

Lgg's exist for most of the languages commonly used in ML. However all practically applicable lgg's (i.e. computable) are based on *syntactical* ordering relations. A relation over hypotheses is syntactical if it does not account for the background knowledge and for the coverage of positive/negative examples. For example dropping condition for nominal attributes, instance relation for atomic formulae and θ-subsumption for clauses are all syntactical relations. On the other hand the evaluation of the hypotheses produced by an lgg operator is based on their coverage of positive/negative examples with respect to the background knowledge, i.e. it is based on *semantic* relations (in the sense of the inductive task). This discrepancy is a source of many problems in ML, where overgeneralization is the most difficult one.

There exists a general semantic relation over hypotheses in all languages. It can be defined by the *set inclusion* relation between the sets of examples covered by the hypotheses. In [13] it is called *empirical subsumption* relation. The empirical subsumption is a *preorder* and can be easily extended to a partial order by using the equivalence classes as elements. Unfortunately the corresponding lgg does not exists in the general case (actually the intersection of two sets is their *lgg*, however it does not always have an explicit representation in the underlying language). In [13] the empirical subsumption is used for reducing the class of equivalent hypotheses under the corresponding syntactical relation. Generally this kind of semantic relation is used as a preference criterion for evaluation of the hypotheses generated by refinement operators or *lgg*'s based on syntactical relations.

Our approach in fact also uses empirical subsumption, however we make a step further and formally define a distance measure to make use of the empirical subsumption to evaluate the similarity between hypotheses. This allows us to integrate consistently metric-based and similarity-based approaches to building concept hierarchies in ML.

6. Concluding remarks

The paper is an attempt to combine theoretical and practical research in ML. We use basic results from lattice theory to develop a unified inductive learning framework. We also introduce an algorithm and illustrate by examples its application in two areas of ML – conceptual clustering and concept learning. Compared to similar ones our approach has two basic advantages:

- It is language independent, i.e. it can be applied both within propositional (attribute-value) languages and within first order languages.
- It allows consistent integration of generalization operators with semantic distance measures.

Clearly more theoretical and practical work is needed to investigate the advantages and drawbacks of our approach. In this respect we see the following directions for future work:

- Particular attention should be paid to the clause reduction problem when using the language of Horn clauses. Other lgg operators, not based on θ-subsumption should be considered too.

- The practical learning data often involve numeric attributes. Proper relations, lgg's and covering functions should be investigated in order to extend the approach for handling numeric data.

- Though the algorithm is well founded it still uses heuristics. This is because building the complete lattice is exponential and we avoid this by employing a hill-climbing strategy (choosing a single minimal element in Step 4). Obviously this leads to incompleteness. Therefore other strategies should be investigated or perhaps the semantic relation should be refined to incorporate these additional heuristics.

- Finally, more experimental work needs to be done to investigate the behavior of the algorithm in real domains (involving large amount of xamples and noise).

Acknowledgements

The author thanks the anonymous referees for their valuable comments and suggestions that improved the paper.

References

[1] B. Monjardet, *Metrics on partially ordered sets – a survey*, Discrete Mathematics, **35** (1981) 173–184.
[2] A. Hutchinson, *Metrics on terms and clauses*, Machine Learning: ECML-97, eds. van Someren and G. Widmer, Lecture Notes in Artificial Intelligence 1224, Springer-Verlag (1997) 138–145.
[3] Z. Markov and I. Marinchev, *Metric-based inductive learning using semantic height functions*, Machine Learning: ECML 2000, eds. R. de Mantaras and E. Plaza, Lecture Notes in Artificial Intelligence 1810, Springer (2000) 254–262.
[4] Z. Markov and N. Pelov, *A framework for inductive learning based on subsumption lattices*, Proceedings of AIMSA'98, ed. F. Guinchiglia, Lecture Notes in Artificial Intelligence 1480, Springer (1998) 341–352.
[5] S. Thrun, et al, *The MONK's problems - a performance comparison of different learning algorithms*, TR CS-CMU-91-197, Carnegie Mellon University (1991).
[6] C. Blake and C. Merz, *UCI repository of machine learning databases* (1998).
[7] P. Baffes and R. Mooney, *Symbolic revision of theories with m-of-n rules*, Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93), Chambery, France 1135–1140.
[8] J. Reynolds, *Transformational systems and the algebraic structure of atomic formulas*, Machine Intelligence, **5** (1970) 135–152.
[9] S. Muggleton, *Inductive logic programming*, Inductive Logic Programming, ed. S. Muggleton, Academic Press (1992) 3–28.
[10] T. Mitchell, *Generalization as search*, Artificial Intelligence, **18** (1982) 203–226.
[11] E. Shapiro, *Algorithmic program debugging*, MIT Press (1983).
[12] P. van der Laag, *An Analysis of Refinement Operators in Inductive Logic Programming*, Ph.D. thesis, Tinbergen Institute Research (1995).

[13] M. Champesme, P. Brézellec and H. Soldano, *Empirically conservative search space reduction*, Proceedings of ILP-95, ed. L. DeRaedt, Dept. of Computer Science, K.U.Leuven (1995) 387–401.

[14] S. H. Nienhuys-Cheng and R. de Wolf, *Least generalizations and greatest specializations of sets of clauses*, Journal of Artificial Intelligence Research, **4** (1996) 341–363.