An Integrated Distance for Atoms

V. Estruch C. Ferri J. Hernández-Orallo M.J. Ramírez-Quintana

DSIC, Univ. Politècnica de València
Camí de Vera s/n, 46020 València, Spain.
{vestruch,cferri,jorallo,mramirez}@dsic.upv.es

Abstract. In this work, we introduce a new distance function for data representations based on first-order logic (atoms, to be more precise) which integrates the main advantages of the distances that have been previously presented in the literature. Basically, our distance simultaneously takes into account some relevant aspects, concerning atom-based presentations, such as the position where the differences between two atoms occur (context sensitivity), their complexity (size of these differences) and how many times each difference occur (the number of repetitions). Although the distance is defined for first-order atoms, it is valid for any programming language with the underlying notion of unification. Consequently, many functional and logic programming languages can also use this distance.

Keywords: First-order logic, distance functions, similarity, knowledge representation.

1 Introduction

Distances (also called metrics) pervade computer science as a theoretical and practical tool to evaluate similarity between objects. The definition of a distance over a set of objects allows us to consider this set as a metric space. This gives us a repertoire of tools and methods to work and analyse the objects therein. Hence, there has been a considerable effort to define distances for any kind of object, including complex or highly structured ones, such as tuples, sets, lists, trees, graphs, images, sounds, web pages, ontologies, XML documents, etc.

The notion of distance between objects allow us to reason about the amount of transformations needed to go from one object to another (and viceversa). A distance is also a frequent formalisation of the notion of error: it is not the same to output 3.3 instead of 3.4 than to output 3.3 instead of 15.2. In the area of programming languages the notion of distance is still an awkward concept, since objects which are (syntactically) different are just that, different objects, and there is not much interest in measuring how much different they are. However, potential applications of the use of distances exist in the areas of debugging (as a measure of the magnitude of the error), termination (to find similar traces or similar rewriting terms), program analysis (to find similar parts in the code that could be generalised), and program transformation (to approximate the distance between two terms). At a meta-level, the use of functional languages to implement distances has been vindicated by [1].

Functional and logic programming languages, additionally, have many important applications as languages for object (knowledge) representation. Logic, and logic programming in particular, is one of the most common formalisms to represent (relational) knowledge. Likewise, functional programming is becoming more and more usual too as a knowledge representation formalism, especially with the use of XML documents and related functional-alike structures [4]. The use of distances in the area of knowledge representation is commonplace.

Some of the areas where functional and logic programming have profusely been used as a knowledge representation formalism are machine learning and program synthesis, in intersecting areas known as Inductive Logic Programming [17][13], Inductive Functional Logic Programming [9][10][6] or, more generally, Inductive Programming [8].

Given that Inductive Programming overlaps machine learning, there has been a remarkable interest in upgrading learning techniques to deal with program-based representations. Among them, we find the so-called instance or distance-based methods. The great advantage of these methods is that the same algorithm or technique can be applied to different sorts of data, as long as a similarity function has previously been defined over them [16]. It is widely-known that the performance of these methods depends to a great extent on the similarity function employed. Thus, it is convenient that the similarity functions satisfy some well-defined properties such as positive definiteness or triangular inequality in order to ensure consistent results [20]. Overall, this makes distance functions a very appropriate tool to express (dis)similarity.

There has been a considerable effort to derive distances for any datatype, including complex or structured datatypes. Hence, we find distances for sets, lists, trees, graphs, etc. One challenging case in machine learning, but more especially in the area of functional and logic programming, is the distance between first-order atoms and terms. Although atoms and terms can be used to represent many of the previous datatypes (and consequently, a distance between atoms/terms virtually becomes a distance for any complex/structured data), they are specially suited for term-based or tree-based representations. In this way, distances between atoms are not only useful in the area of inductive logic programming (ILP) [17] (e.g. first-order clustering [3]), but also in other areas where structured (hierarchical) information is involved such as learning from ontologies or XML documents. For instance, if an XML document represents a set of cars, or houses, or customers, we may be interested in obtaining the similarities between the objects, or to cluster them according to their distances. However, it is important to remark that a distance between atoms or terms is not the same as a distance between trees (such as many other introduced in the literature, see, e.g., the Bille's survey [2]), since two subterms are just different when the topmost element of their tree representation is different, while this is not generally the case for trees.

In this work, we introduce a new distance between ground terms and atoms, which integrates the advantages that some of the existing distances between atoms have separately. In particular, we recover the context sensitivity of Nienhuys-Cheng's distance [18], which implies that the distance between two atoms depends not only on their syntactic differences but also on the positions where these differences take place. This becomes crucial for many applications where atoms represent hierarchical information (e.g. an XML document). Additionally, and like Nienhuys-Cheng's distance, our distance is also a normalised function, which is an interesting property to reduce the effect caused in the distance by noisy or irrelevant information [15], and can easily be composed with other distances in order to define metrics for more complex representations.

However, Nienhuys-Cheng's proposal shows some disadvantages in that it does not properly deal with repeated differences between atoms, which is indeed a common property when handling this datatype, and also ignores the syntactic complexity of these differences. This is considered by J. Ramon et al. distance [20] but at the expense of disregarding context-sensitivity, normalisation and composability.

Our approach does consider repetitions and complexity as J. Ramon et al. do, but in a different way which allows us to preserve context-sensitivity, normalisation and composability. This is so because we do not need to rely on the *least general generalisation operator* (*lgg*) [19] in order to manage repeated differences.

This paper is organised as follows. Section 2 introduces the notation and some previous definitions that will be used in the following sections. Section 3 reviews and analyses these two previous distances proposed in the literature which are related to our proposal. Our distance between ground terms/atoms is formally defined in Section 4. An illustrative example is presented in Section 5 in order to compare how our distance works in practice wrt. the two aforementioned distances. Section 6 concludes the paper and relates to future work. Finally, note that an important part of our work deals with proving that our proposal agrees all the axioms a distance function is supposed to satisfy. Although the main result is included in this work, the proofs of the main theorem and auxiliary results can be found in [5].

2 Preliminaries

Let $\mathcal{L}$ be a first order language defined over the signature $\Sigma = \langle \mathcal{C}, \mathcal{F}, \Pi \rangle$ where $\mathcal{C}$ is a set of constants, and $\mathcal{F}$ (respectively Π) is a family indexed on $\mathbb{N}$ (non negative integers) being $\mathcal{F}_n$ (Π_n) a set of $n-$adic function (predicate) symbols. Atoms and terms are constructed from the Σ as usual. An expression is either a term or an atom. The root symbol and the arity of an expression t is given by the functions $Root(t)$ and $Arity(t)$, respectively. Thus, letting $t = p(a, f(b))$, $Root(t) = p$ and $Arity(t) = 2$. By considering the usual representation of t as a labelled tree, the occurrences are finite sequences of positive numbers (separated by dots) representing an access path in t. We assume that every occurrence is always headed by a (implicit) special symbol λ, which denotes the empty occurrence. The set of all the occurrences of t is denoted by $O(t)$. In our case, $O(t) = \{\lambda, 1, 2, 2.1\}$. We use the (indexed) lowercase letters $o', o, o_1, o_2, \ldots$ to represent occurrences. The length of an occurrence o, $Length(o)$, is the number of items in o (λ excluded). For instance, $Length(2.1) = 2$, $Length(2) = 1$ and $Length(\lambda) = 0$. Additionally, if $o \in O(t)$ then $t|_o$ represents the subterm of t at the occurrence o. In our example, $t|_1 = a$, $t|_2 = f(b)$, $t|_{2.1} = b$. In any case, we always have that $t|_\lambda = t$. By $Pre(o)$, we denote the set of all prefix occurrences of o different from o. For instance, $Pre(2.1) = \{\lambda, 2\}$, $Pre(2) = \{\lambda\}$ and $Pre(\lambda) = \emptyset$. Two expressions s and t are compatible (denoted by the Boolean function $Compatible(s,t)$) iff $Root(s) = Root(t)$ and $Arity(s) = Arity(t)$. Otherwise, we say that s and t are incompatible ($\neg Compatible(s,t)$).

3 Related Work

As mentioned in the introduction, although distances for atoms can be used for many applications, only two relevant proposals have been generally used to compute distances between atoms.

In [18], Nienhuys-Cheng introduces a bounded distance for ground terms/atoms which takes the depth of the symbol occurrences into account in such a way that differences occurring close to the root symbols count more. Given two ground terms/atoms $s = s_0(s_1, \ldots, s_n)$ and $t = t_0(t_1, \ldots, t_n)$, this distance (denoted by d_N) is recursively defined as follows.

$$d_N(s,t) = \begin{cases} 0, & \text{if } s = t \\ 1, & \text{if } \neg Compatible(s,t) \\ \frac{1}{2n}\sum_{i=1}^{n} d(s_i,t_i), & \text{otherwise} \end{cases}$$

For instance, if $s = p(a,b)$ and $t = p(c,d)$ then $d_N(s,t) = 1/4 \cdot (d(a,c) + d(b,d)) = 1/4(1+1) = 1/2$.

A different approach is presented by J. Ramon et al. in [21]. Following [11], the authors define a distance between (non-)ground terms/atoms based on the syntactic differences wrt. their *lgg*. An auxiliary function, the so-called $Size(t) = (F,V)$, is required to compute this distance. Roughly speaking, F counts the number of predicate and function symbols occurring in t and V is the sum of the squared frequency of appearance of each variable in t. Finally, this distance (denoted by d_R) is formulated as follows. Given two terms/atoms s and t,

$$d_R(s,t) = [Size(s) - Size(lgg(s,t))] + [Size(t) - Size(lgg(s,t))]$$

Thus, one of its particularities is that d_R returns an ordered pair of integer values (F,V) instead of a single value, that expresses how different two atoms are in terms of function and variable symbols, respectively. For instance, if $s = p(a,b)$

and $t = p(c, d)$ and knowing that $lgg(s, t) = p(X, Y)$, we have

$$Size(s) = (3, 0)$$
$$Size(t) = (3, 0)$$
$$Size(lgg(s, t)) = (1, 2)$$
$$d_R(s, t) = [(3, 0) - (1, 2)] + [(3, 0) - (1, 2)] = (2, -2) + (2, -2) = (4, -4)$$

With regard to these distances for atoms, some interesting properties are analysed next.

1. *Context Sensitivity*: it is the possibility of taking into consideration where the differences between two terms/atoms occur. Intuitively, it means that the distance between two atoms such as $p(a)$ and $p(b)$ should be greater than the distance between $p(f(a))$ and $p(f(b))$, as these latter atoms have more symbols (information) in common than the two previous ones. Or equivalently, symbolic differences occurring at deeper positions count less since they provide less information.
   The Nienhuys-Cheng's distance does not always satisfy this property. Note that, by definition, the distance between atoms decreases as the differences occur at deeper positions. For instance, in our example:

   $$d_N(p(a), p(b)) = 1/2 \quad d_N(p(f(a)), p(f(b))) = 1/4$$

   Nevertheless, when differences occur at the same depth but in all of the arguments of two terms/atoms (that is, the number of differences coincides with the arity of the outermost symbol) then the distance behaves as if there was only one difference. For instance, given the atoms $e_1 = p(f(a_1, \ldots, a_n), g(a))$, $e_2 = p(f(b_1, \ldots, b_n), g(a))$ and $e_3 = p(f(a_1, \ldots, a_n), g(b))$ it would be expected that the distance between e_1 and e_2 were greater than the distance between e_1 and e_3, since there are n differences between e_1 and e_2 whereas there is only one difference between e_1 and e_3. However,

   $$d_N(e_1, e_2) = 1/8 \quad d_N(e_1, e_3) = 1/8$$

   As we have mentioned, the first component F of the J. Ramon et al.'s distance counts the differences between the functors of the two atoms. This component can be context-sensitive by giving different importance to components in different positions. The authors do that by associating a set of $(m + 1)$ positive weights with each functor $f \in \mathcal{F}_m$ and a set of $(n + 1)$ positive weights with each predicate $p \in \Pi_n$. In this way, the definition of the F-component is parametrised by these weights (see Definition 4 in [21]). Thus, J. Ramon et al.'s distance is not always context-sensitive (depending on the weights used).
2. *Normalisation*: sometimes, it is useful to work with normalised distances. In this sense, a distance function d which returns (non-negative) real numbers can be easily normalised, for instance, using the expression $d/(1 + d)$, which is known that results in an equivalent distance [12]. However, a distance like that of J. Ramon et al. is very difficult (or at least, not intuitive) to be normalised since it returns a pair of integer numbers.

3. *Repeated differences*: this concerns the fact of handling repeated differences between terms/atoms properly. Suppose that the atoms $r = p(a, a)$, $s = p(b, b)$ and $t = p(c, d)$ are given. Intuitively, it is reasonable to expect that the atoms r and s come nearer than the atoms r and t (or s and t), since r and s share that their (sub)terms (a and b, respectively) occur twice whereas no (sub)term is repeated in t.
   Only J. Ramon et al.'s distance can handle repetitions since this distance is defined via the *lgg* operator, which takes repetitions into consideration. However, this possibility is lost when the Nienhuys-Cheng's distance is used. In our example,

$$\begin{array}{ll} d_N(r,s) = 1/2 & d_N(r,t) = 1/2 \\ d_R(r,s) = (2,-2) & d_R(r,t) = (3,-4) \end{array}$$

4. *Size of the differences*: another interesting question to be treated is the complexity (the size) of the differences occurring when two terms/atoms are compared. Logically, it is expected that as the size of two terms/atoms increases, its distance will become greater. This is so regarding the J. Ramon et al.'s proposal, since it explicitly introduces a size function. However, Nienhuys-Cheng's disregards this important fact. For instance, given the atoms $p(a)$, $p(b)$ and $p(f(c))$ then,

$$\begin{array}{ll} d_N(p(a),p(b)) = 1/2 & d_N(p(a),p(f(c))) = 1/2 \\ d_R(p(a),p(b)) = (2,-2) & d_R(p(a),p(f(c))) = (3,-2) \end{array}$$

5. *Handling variables*: variables become a useful tool when part of the structure of an object is missing. J. Ramon et. al's proposal handles both constant and variable symbols indistinctly in a very elegant way. As seen, Nienhuys-Cheng's distance is defined over ground terms/atoms and for this reason needs some non-integrated extra concepts (*least Herbrand model* and *Hausdorff distance*) in order to deal with variable symbols.
6. *Composability*: A tuple is a widely used structure for knowledge representation in real applications, since examples are usually represented as tuples of values of different data types (nominal, numerical, atoms, graphs, ...). For example, a molecule can be described as a tuple composed by its breaking temperature (a real number) and its description (expressed, for instance, as a list of symbols). The property of composability allows us to define distance functions for tuples by combining the distance functions defined over the basic types from which the tuple is constructed. Typically, the combination is made as a linear combination of the underlying distances, which is well-known to be a distance. Therefore, composability requires that the computed distances are expressed as real values in order to combine them. Obviously, the Nienhuys-Cheng's distance holds this condition. However, the J. Ramon et. al's distance computes a pair of numbers, so it is necessary to first transform it into a real number before composing it. And, as we have mentioned, converting J.Ramon et al's distance into a single number seems difficult.

7. *Weights*: in some cases, it may be convenient to give higher or lower weights to some constants or function symbols, in such a way that the distance between $f(a)$ and $f(b)$ could greater than the distance between $f(c)$ and $f(d)$. In some other occasions it may be interesting to give more or less weight to specific positions in a term over others. This latter case is more general than the first one. Nienhuys-Cheng's distance does not allow weights while J.Ramon et al.'s does. Our proposal allows this possibility in an indirect way, by the use of dummy function symbols. For instance, the distance between $f(a)$ and $f(b)$ can be increased if we just rewrite them into $f(d_1(d_2(a)))$ and $f(d_1(d_2(b)))$. The weight depends on the number of dummy symbols which have been introduced.

	Nienhuys-Cheng	J. Ramon et al.	Our distance
Context	Not always	Not always (depending on the weights used)	Yes
Normalisation	Yes	Not easy	Yes
Repetitions	No	Yes	Yes
Size	No	Yes	Yes
Variables	Indirectly	Yes	Indirectly
Composability	Yes	Difficult	Yes
Weights	No	Yes	Indirectly

Table 1. Advantages and drawbacks of several distances between terms/atoms.

This analysis about Nienhuys-Cheng's and J. Ramon et al.'s distances suggests to integrate both in a new distance that inherits the best of them. Table 1 compares these three distances in terms of the properties above: that is, context-sensitivity (*Context*), ease of normalisation (*Normalisation*), handling repeated differences (*Repetitions*), complexity of the differences (*Size*), handling variable symbols (*Variables*) and the ability to be combined with other distances (*Composability*). In the table, we have also included which of these properties are satisfied and which are not by the distance we will define in the next section.

In the following sections we will show how these issues are accomplished in a novel way, by: *i*) understanding terms/atoms as rooted acyclic directed graphs, *ii*) introducing a new size function which is *iii*) weighted depending on the context and the number of times the differences occur.

4 Distance between Atoms

As said, the distance function we present in this work takes into consideration three fundamental issues concerning first-order atoms: namely, the complexity of the syntactic differences between the atoms, the number of times each syntactic difference occurs and finally, the position (or context) where each difference takes place. This all is formalised next.

First, we precisely define what we mean by syntactical differences between expressions.

Definition 1. ***(Syntactical differences between expressions)*** *Let s and t be two expressions, the set of their syntactic differences, denoted by $O^\star(s,t)$, is defined as:*

$$O^\star(s,t) = \{o \in O(s) \cap O(t) : \neg Compatible(s|_o, t|_o) \text{ and } Compatible(s|_{o'}, t|_{o'}), \forall o' \in Pre(o)\}$$

For instance, with $s = p(f(a), h(b), b)$ and $t = p(g(c), h(d), d)$, then $O^\star(s,t) = \{1, 2.1, 3\}$. Additionally, observe that if $\neg Compatible(s,t)$ then $O^\star(s,t) = \{\lambda\}$.

The complexity of the syntactic differences between s and t is calculated on the number of symbols the subterms (in s and t) at the occurrences $o \in O^\star(s,t)$ are composed of. For this purpose, we introduce a special function called $Size'$ which is defined next.

Definition 2. ***(Size of an expression)*** *Given an expression $t = t_0(t_1, \ldots, t_n)$, we define the function $Size'(t) = \frac{1}{4}Size(t)$ where,*

$$Size(t_0(t_1,\ldots,t_n)) = \begin{cases} 1, \; n = 0 \\ 1 + \frac{\sum_{i=1}^n Size(t_i)}{2(n+1)}, \; n > 0 \end{cases}$$

For instance, considering $s = f(f(a), h(b), b)$, then $Size(a) = Size(b) = 1$, $Size(f(a)) = Size(h(b)) = 1{+}1/4 = 5/4$, $Size(s) = 1{+}(5/4{+}5/4{+}1)/8 = 23/16$ and finally, $Size'(s) = 23/64$. The rationale for the denominator in definition 2 is that we expect to have that $Size(p(a)) < Size(p(a,b,c))$. Consequently, the denominator has to be greater than $2n$ in order to avoid a cancellation with the size of the arguments, as happens with Nienhuys-Cheng's distance.

The next step is devoted to find out repeated differences between atoms. To do this, we define an equivalence relation ($\sim$) on the set $O^\star(s,t)$, as follows:

$$\forall o_i, o_j \in O^\star(s,t), \; o_i \sim o_j \Leftrightarrow s|_{o_i} = s|_{o_j} \text{ and } t|_{o_i} = t|_{o_j}$$

Consequently, there exists a non-overlapping partition of $O^\star(s,t)$ into equivalence classes, that is, $O^\star(s,t) = \cup_{i \in I} O_i^\star(s,t)$. Related to this, we also introduce the auxiliary function $\pi : O^\star(s,t) \to I$ which just returns the index of the equivalence class one occurrence belongs to.

Back to our example, we can see that there only exist two equivalence classes: namely, $O_1^\star(s,t) = \{1\}$ and $O_2^\star(s,t) = \{2.1, 3\}$. Hence, $\pi(1) = 1$, $\pi(2.1) = \pi(3) = 2$.

Finally, we need to set the context of every syntactic difference. This is formalised as follows:

Definition 3. (Context value of an occurrence) *Let t be an expression. Given an occurrence $o \in O(t)$, the context value of o in t, denoted by $C(o;t)$, is defined as*

$$C(o;t) = \begin{cases} 1, \; o = \lambda \\ 2^{Length(o)} \cdot \prod_{\forall o' \in Pre(o)} (Arity(t|_{o'}) + 1), \; \textit{otherwise} \end{cases}$$

For instance, $C(\lambda; t) = 1$, $C(1; t) = 2 \cdot (3 + 1) = 8$ and $C(2.1; t) = 2^2 \cdot (1 + 1) \cdot (3 + 1) = 32$.

Therefore, the context value tells us about the relationship between $t|_o$ and t in the sense that, a high value of $C(o; t)$ corresponds to a deep position of $t|_o$ in t or the existence of superterms of $t|_o$ with a large number of arguments. As we will see later, this information will be employed to conveniently weight the syntactic differences between atoms.

The context value of an occurrence satisfies the following property.

Proposition 1. *Given two expressions s and t, if $o \in O^\star(s, t)$ then $C(o; s) = C(o; t)$*

Proof. It directly comes from the definition of $O^\star(s, t)$. If $o \in O^\star(s, t)$ then, for any $o' \in Pre(o)$, $Compatible(s|_{o'}, t|_{o'})$, hence $Arity(s|_{o'}) = Arity(t|_{o'})$.

When no doubts arise from omitting t, the short form $C(o)$ will be used instead. Definition 3 allows us to set an order relation $(\leq)$ in every equivalence class $O^\star_i(s, t)$. That is,

$$\forall o_j, o_k \in O^\star_i(s, t),\ o_j \leq o_k \Leftrightarrow C(o_j) \leq C(o_k)$$

Note that the relation order $\leq$ makes sense on the grounds of Proposition 1. Additionally, for every ordered equivalence class, $(O^\star_i, \leq)$, we define the function $f_i : (O^\star_i, \leq) \to \mathbb{N}^+$ that simply returns the position an occurrence $o \in O^\star_i$ has according to $\leq$. In the case of $C(o_i) = C(o_j)$, we can rank first either o_i or o_j, since as we will see, this decision will not affect the computation of the proposed distance.

We still require, previously to introduce our distance, another additional function. Again, given two expressions s and t, we define the function w as:

$$\begin{array}{ll} w : O^\star(s, t) \to \mathbb{R}^+ & \\ \qquad o \qquad\quad \mapsto w(o) = \frac{3 f_i(o) + 1}{4 f_i(o)}, & \text{where } i = \pi(o) \end{array}$$

Note that the function w simply associates weights to occurrences in such a way that the greater $C(o)$, the lower the weight o is assigned, i.e., the less meaningful the syntactical difference referred by o is. For instance, if we consider $(O^\star_2(s, t), \leq) = \{3, 2.1\}$ then $w(3) = 1$ and $w(2.1) = 7/8$. By $w_O(o)$, we will denote the restriction of the function $w(\cdot)$ to a subset $O \subset O^\star(s, t)$. Realise that if $o \in O \subset O^\star(s, t)$, then $w_O(o) \geq w(o)$.

Finally, the distance between atoms we propose in this work is defined as:

Definition 4. ***(Distance between atoms)*** *Let s and t be two expressions, the distance between s and t is,*

$$d(s, t) = \sum_{o \in O^\star(s,t)} \frac{w(o)}{C(o)} \big(Size'(s|_o) + Size'(t|_o)\big)$$

Theorem 1. *The ordered pair* $(\mathcal{L},d)$ *is a bounded metric space. Concretely,* $0 \leq d \leq 1$*.*

Proof. For any expressions r, s and t in $\mathcal{L}$, the function d satisfies:

1. (identity): $d(r,t) = 0 \Leftrightarrow r = t$. If $d(r,t) = 0$ then $O^\star(r,t) = \emptyset$ which necessarily means that $r = t$. As for the other implication, if $r = t$ then $O^\star(r,t) = \emptyset$ and $d(r,t) = 0$.
2. (symmetry): $d(r,t) = d(t,r)$. Simply, note that $O^\star(r,t) = O^\star(t,r)$
3. (triangular inequality): $d(r,t) \leq d(r,s) + d(s,t)$. See [7].
4. (bounded distance): $0 \leq d(r,t) \leq 1$. See [7].

Next, we provide short artificial examples illustrating how our distance works.

Example 1. $s = f(a)$ and $t = a$. We have that $O^\star(s,t) = \{\lambda\}$. Next,

$$C(\lambda) = 1$$

The sizes of the subterms involved in the computation of the distance are:

$$Size'(f(a)) = 5/16 \text{ and } Size'(a) = 1/4$$

Obviously,

$$w(\lambda) = 1$$

Finally,

$$d(s,t) = \frac{1}{1}\big(Size'(s) + Size'(t)\big) = \big(\frac{5}{16} + \frac{1}{4}\big)$$

Example 2. $s = p(a,a)$ and $t = p(f(b), f(b))$. We have that $O^\star(s,t) = \{1,2\}$. Next,

$$C(1) = C(2) = 2 \cdot (2+1) = 6$$

The sizes of the subterms involved in the computation of the distance are:

$$Size'(a) = 1/4 \text{ and } Size'(f(b)) = 5/16$$

There is only one equivalence class $O^\star = O^\star_1(s,t)$. Assume that the occurrence 1 is ranked first,

$$w(1) = 1 \text{ and } w(2) = 7/8$$

Finally,

$$d(s,t) = \frac{1}{6}\big(\frac{1}{4} + \frac{5}{16}\big) + \frac{7}{48}\big(\frac{1}{4} + \frac{5}{16}\big)$$

Example 3. $s = p(a,a,f(c))$ and $t = p(b,b,f(b))$. We already know that $O^\star(s,t) = \{1,\ 2,\ 3.1\}$, The contexts respective differences are,

$$C(1) = C(2) = 2^1 \cdot (3+1) = 8 \text{ and } C(3.1) = 2^2 \cdot (1+1) \cdot (3+1) = 32$$

The sizes of the subterms involved in the computation of the distance are:

$$Size'(a) = Size'(b) = Size'(c) = 1/4$$

Additionally, we have seen that $(O_1^\star(s,t), \leq) = \{1\}$, $(O_2^\star(s,t), \leq) = \{2, 3.1\}$. Consequently,

$$w(1) = 1,\ w(3.1) = 1 \text{ and } w(2) = 7/8$$

Finally,

$$d(s,t) = \frac{1}{8}\left(\frac{1}{4} + \frac{1}{4}\right) + \frac{7}{64}\left(\frac{1}{4} + \frac{1}{4}\right) + \frac{1}{32}\left(\frac{1}{4} + \frac{1}{4}\right)$$

5 Discussion

Next, we present a simple but illustrative comparison on how our distance performs wrt. those distances reported in Section 3. Basically, we aim to analyse the notion of similarity shaped by the different distances. For this purpose, we will use a toy XML dataset containing several car descriptions (see Table 2) and we will see how similar these descriptions are depending on the distance employed.

Our XML dataset contains structured information about 8 different cars. More concretely, for every car, we know the company, the model, a list of certifications that some organisations have granted to the car, the engine and several other features. As we can see, every description can directly be represented as an atom, except from some attributes: the photo which cannot be properly represented, and two numerical values (the power and the baseprice) which can be represented inside an atom, but that any of the atom distances is not going to handle appropriately (directly). Table 3 shows a term-based representation of the whole dataset.

Ignoring the photo and the two numerical values for the moment, we see that if we focus on cars 1, 2 and 3, we intuitively see that the car 1 looks more similar to the car 2 than 1 to 3, although, both pairs of cars $(1, 2)$ and $(1, 3)$ have an identical number of differences. Namely, the difference between the cars 1 and 2 relies on the *engine traits* (occurrences 4.3.1 and 4.3.2) whereas cars 1 and 3 differ both in *company* and *model* (occurrences 1 and 2). Therefore, the reason why car 2 comes nearer to 1 than car 3 is due to a qualitative criterion rather than quantitative, in that *company* and *model* results in a more meaningful difference than the *engine traits*.

In general, it makes sense to assume that differences at top positions in the atoms are more important than differences at inner positions. In our case, as well as in Nienhuys-Cheng's proposal, the position of the differences, the so-called context, between atoms is taken into account when computing the distance. Note that, this aspect is disregarded by the unweighted J. Ramon et al.'s distance. For this distance, cars 2 and 3 are equally similar to car 1.

Furthermore, note that a context-sensitive distance allows us to indirectly use the position in the atom/term in order to set different levels of importance for every trait of the car. For instance, moving the trait *colour* to a higher position in the atom implies that differences involving this attribute become more

```
<?xml version='1.0' ?>
<!DOCTYPE root SYSTEM "cars.dtd">
<root>
  <car>
    <company> Chevrolet </company>
    <model> Corvette </model>
    <certifications> E3 </certifications>
    <certifications> D52 </certifications>
    <certifications> RAC </certifications>
    <features>
       <color> red </color>
       <brake> abs </brake>
       <power> 250 </power>
       <airbag>
          <front> full </front>
          <rear> mid </rear>
       </airbag>
       <engine>
          <type> diesel </type>
          <turbo> yes </turbo>
       </engine>
    </features>
    <baseprice> 60,000 </baseprice>
    <photo> ChevCorv.jpg </photo>
  </car>
  . . .
</root>
```

Table 2. A representative extract from the XML dataset.

1	car(Ford,Ka,cert([E3]),feats(75, red,abs,ab(full,mid),mt(gas,no)), 9000, ChevKaG.jpg)
2	car(Ford,Ka,cert([E3]),feats(80, red,abs,ab(full,mid),mt(diesel,yes)), 10000, ChevKaD.jpg)
3	car(Chev,Corv,cert([E3]),feats(250,red,abs,ab(full,mid),mt(gas,no)), 60000, ChevCorv.jpg)
4	car(Ford,Ka,cert([E3]),feats(100, blue,abs,ab(mid,mid),mt(diesel,yes)), 10000, ChevKaD2.jpg)
5	car(Ford,Ka,cert([E3]),feats(125, blue,abs,ab(full,full),mt(diesel,yes)), 10500, ChevKa3.jpg)
6	car(Ford,Ka,cert([E3]),feats(125, blue,abs,ab(extra,no),mt(diesel,yes)), 11000, ChevKaD4.jpg)
7	car(Chev,Xen,cert([D52, RAC, H5]),feats(300, red,abs,ab(full,mid),mt(gas,no)), 70000, CX.jpg)
8	car(Chev,Prot,cert([RAC]),feats(300, red,abs,ab(full,mid),mt(gas,no)), 60000, ChevProt.jpg)

Table 3. An equivalent term-based representation of the XML dataset.

meaningful. In this line, we could also endow our representation language with artificial constructors, namely $art(\cdot)$, which allow us to reduce the importance of a trait. For instance, a nested expression such as $art(art(art(Ford)))$ would decrease the importance of the trait *company*.

Additionally, the size of the differences is also taken into account. If we observe the differences between cars 3, 7 and 8, our intuition gives more similarity to 3 and 8 because they have only one certification while 7 has three. Nienhuys-Cheng's distance disregards this and gives that the three cars are at the same distance to each other. Our distance and J. Ramon et al.'s distance place cars 3 and 8 closer than any of them with 7.

Finally, let us consider the remaining group of cars. We can see that cars 4, 5 and 6 differ in the airbag description (occurrences 4.4.1 and 4.4.2) in such a way that 4 and 5 have an homogeneous airbag equipment but not 6. According to this observation, we can affirm that cars 4 and 5 are more similar than 5 and 6. Here, the rationale is that those differences occurring repeatedly are less significant.

Our distance as well as J. Ramon et al.'s distance are capable of coping with this (repeated differences) and hence, the computed distances are in agreement with this fact. Nevertheless, Nienhuys-Cheng's proposal ignores repeated differences, and for that reason, cars 4 and 5 are at the same distance that cars 5 and 6.

If now we consider the photo and the two numerical values, we see that J. Ramon et al.'s distance is not able to handle them. If we exclude these three values and compute J. Ramon et al.'s distance with the rest, we have as a result a pair such as (n, m). If, next, we compute the distances for the photo and the numerical values, we get three scalar values d_1, d_2 and d_3. We do not know how these four results can be combined and integrated into a single value. In contrast, Nienhuys-Cheng's distance and ours can handle the whole XML description. In both cases, one simple way to compose atom with non-atom representations (such as the picture) is to construct a tuple, taking out all the non-term-based representations, such as pictures and numerical values. The resulting tuple-based representation is shown in Table 4:

1	⟨ 75, 9000, ChevKaG.jpg, car(Ford,Ka,cert([E3]),feats(red,abs,ab(full,mid),mt(gas,no))) ⟩
2	⟨ 80, 10000, ChevKaD.jpg, car(Ford,Ka,cert([E3]),feats(red,abs,ab(full,mid),mt(diesel,yes))))⟩
3	⟨ 250, 60000, ChevCorv.jpg, car(Chev,Corv,cert([E3]),feats(red,abs,ab(full,mid),mt(gas,no))))⟩
4	⟨ 100, 10000, ChevKaD2.jpg, car(Ford,Ka,cert([E3]),feats(blue,abs,ab(mid,mid),mt(diesel,yes))))⟩
5	⟨ 125, 10500, ChevKa3.jpg, car(Ford,Ka,cert([E3]),feats(blue,abs,ab(full,full),mt(diesel,yes))))⟩
6	⟨ 125, 11000, ChevKaD4.jpg, car(Ford,Ka,cert([E3]),feats(blue,abs,ab(extra,no),mt(diesel,yes))))⟩
7	⟨ 300, 70000, CX.jpg, car(Chev,Xen,cert([D52, RAC, H5]),feats(red,abs,ab(full,mid),mt(gas,no))))⟩
8	⟨ 300, 60000, ChevProt.jpg, car(Chev,Prot,cert([RAC]),feats(red,abs,ab(full,mid),mt(gas,no))))⟩

Table 4. An equivalent tuple-based representation of the atom representation.

The two first attributes use distances for real numbers (e.g. the absolute difference), the third attribute can use any distance for images (e.g. the Earth Mover's Distance, Mallows Distance or Kantorovich distance [14]) and the fourth attribute use a distance for atoms. Using a proper weighting of the four attributes in the tuple (by normalising them and then using their original depth as a way to determine their weight), we can now compute the distances between the atoms and then aggregate the four distance values into a single distance. This shows that our distance allows the composability, an important requirement when trying to integrate data which is represented not only as atoms, but also using other data representations.

6 Conclusions and Future Work

In this paper we have presented a new distance for ground terms/atoms which integrates the most remarkable traits in Nienhuys-Cheng's and J. Ramon et al.'s proposals. That is, context-sensitivity in the former case and complexity and repeated differences in the latter without losing the general and convenient feature of returning a single number, instead of two such as J.Ramon et al.'s distance. This can directly be seen from the formulation of the distance where the function $Size'$, which takes the complexity of the differences into account, is weighted by the quotient $\frac{w(o)}{C(o)}$ where the numerator controls the frequency of the repeated difference and the denominator the context where this difference takes place.

Apart from the direct application of a distance between atoms in areas such as machine learning and inductive programming (very especially in inductive logic programming), the distance can also be used when atoms are employed to represent other structures, as we have illustrated for XML documents. Additionally, we hope that a proper measure for terms could foster the application in different areas inside the logic and functional programming communities. In order to show this, we need to implement the distance to conduct experiments in these areas of application.

In this proposal, there is an easy way to assign weights at different positions, by using dummy function symbols. Relating this problem with the limitation of not handling variables directly, as future work, we are working on an extension to consider weights directly and to handle variables directly as J. Ramon et al.'s distance does. In fact, this extension can be done in two different ways. First, upgrading the function $Size'$ as well as the definition of syntactical differences between terms/atoms ($O^\star$) in order to take variable symbols into account. For instance, the size of a variable could be weighted half of the value assigned to a constant symbol. We could also give different weights to different constants or function symbols or to different positions. Second, following J. Ramon et al.'s approach, we could seek to integrate a syntactic difference search guided by the *lgg* into our setting. With this extension (especially with the first approach), our distance would include all the positive features (the desiderata) that we showed on Table 1.

Additionally, we are studying how the ideas of size and context-sensitive could be adapted in order to improve other distances for nested data types (e.g. sequences of sets, or lists of lists, etc.).

7 Acknowledgments

The authors thank the funding from the Spanish Ministerio de Educación y Ciencia (MEC) for projects CONSOLIDER-INGENIO 26706 and TIN 2007-68093-C02, and GVA project PROMETEO/2008/051.

References

1. D. Aleksovski, M. Erwig, and S. Dzeroski. A functional programming approach to distance-based machine learning. In *Conference on Data Mining and Data Warehouses (SiKDD 2008)*. Jozef Stefan Institute, 2008.
2. P. Bille. A survey on tree edit distance and related problems. *Theoretical computer science*, 337(1-3):217–239, 2005.
3. H. Blockeel, L. De Raedt, and J. Ramon. Top-down induction of clustering trees. In *Proc. of the 15th International Conference on Machine Learning (ICML'98)*, pages 55–63. Morgan Kaufmann, 1998.

4. J. Cheney. Flux: functional updates for xml. In *Proceeding of the 13th ACM SIGPLAN international conference on Functional programming, ICFP*, pages 3–14. ACM, 2008.
5. V. Estruch, C. Ferri, J. Hernández-Orallo, and M.J. Ramírez-Quintana. A new context-sensitive and composable distance for first-order terms. Technical report, Departament de Sistemes Informatics i Computacio, Universitat Politecnica de Valencia, http://users.dsic.upv.es/∼flip, 2009.
6. C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Incremental learning of functional logic programs. In *In Proc. of the 5th Int. Symposium in Functional and Logic Programming (FLOPS'01)*, volume 2024 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2001.
7. C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Learning MDL-guided decision trees for conctructor based languages. In *Work in progress track in the 11th Int. Conference on Inductive Logic Programming (ILP'01)*, volume 3625 of *LNCS*, pages 87–102. Springer, 2001.
8. Pierre Flener and Ute Schmid. An introduction to inductive programming. *Artificial Intelligence Review*, 29(1):45–62, 2008.
9. J. Hernández and M. J. Ramírez. Inverse narrowing for the induction of functional logic programs. In *Proceedings of the Joint Conference on Declarative Programming*. Univ. de la Coruña, 1998.
10. J. Hernández-Orallo and M. J. Ramírez-Quintana. A strong complete schema for inductive functional logic programming. In *Proc. of the 9th Int. Conference on Inductive Logic Programming (ILP'1999)*, volume 1634 of *Lecture Notes in Computer Science*, pages 116–127. Springer, 1999.
11. A. Hutchinson. Metrics on terms and clauses. In *Proc. of the 9th European Conference on Machine Learning (ECML'97)*, pages 138–145. Springer-Verlag, 1997.
12. J. Nagata K.P. Hart and J.E. Vaughan. *Encyclopedia of General Topology*. Elsevier, 2003.
13. N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.
14. Elizaveta Levina and Peter J. Bickel. The earth mover's distance is the mallows distance: Some insights from statistics. In *8th International Conference on Computer Vision*, pages 251–256. IEEE, 2001.
15. A. Marzal and E. Vidal. Computation of normalized edit distance and applications. *IEEE Transactions on Pattern Analysis and Machine Learning Intelligence*, 15(9):915–925, 1993.
16. T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
17. S. Muggleton. Inductive Logic Programming. *New Generation Computing*, 8(4):295–318, 1991.
18. S.H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Artificial Intelligence*. Springer, 1997.
19. G. Plotkin. A note on inductive generalisation. *Machine Intelligence*, 5:153–163, 1970.
20. J. Ramon and M. Bruynooghe. A framework for defining distances between first-order logic objects. In *Proc. of the 8th Int. Conference on Inductive Logic Programming (ILP'98)*, pages 271–280. Springer, 1998.
21. J. Ramon, M. Bruynooghe, and W. Van Laer. Distance measures between atoms. In *CompulogNet Area Meeting on Computational Logic and Machine Learing*, pages 35–41. University of Manchester, UK, 1998.