# Applying distances between terms to both flat and hierarchical data

J.A. Bedoya-Puerta C. Ferri J. Hernández-Orallo
M.J. Ramírez-Quintana

DSIC, Universitat Politècnica de València
Camí de Vera s/n, 46022 València, Spain.
`jorbepue@posgrado.upv.es, {cferri,jorallo,mramirez}@dsic.upv.es`

**Abstract.** Terms are the basis for functional and logic programming representations. In turn, functional and logic programming can be used for knowledge representation in a variety of applications (knowledge-based systems, data mining, etc.). Distances between terms provide a very useful tool to compare terms and arrange the search space in many of these applications. However, distances between terms have special features which have precluded them from being used for other datatypes, such as hierarchical data or propositional data. In this paper, we explore the use of distances between terms in different scenarios: propositional data using the names of the attributes to construct the term tree (hierarchy), deriving the term tree by using attribute similarity and, finally, functional data representing hierarchies. In order to do this, we perform transformations from the original data representation to XML, thus allowing the use of term distances to directly handle objects of different degrees of 'hierarchisation', from flat data to fully hierarchical data.

**Keywords:** Distance functions, classification, term-based representation, hierarchical data, granular computing, XML data.

## 1 Introduction

Tree structures and functional terms have strong similarities. A term in functional (or logic) programming can be represented as an ordered tree. Trees can be used to represent information or knowledge (e.g. ontologies), and some very popular languages for information representation are based on trees or hierarchies, such as XML and related functional-alike structures [4]. So, inductive programming can be applied to this kind of information representation in a much more natural way than other paradigms which learn from examples with a flat structure (tabular data, text, etc.).

Although semantics are crucial to understand the role of a term or an atom wrt. a program, we can also analyse terms in an isolated, purely syntactical, way. In knowledge representation, terms and atoms may represent objects and its syntactical structure and content provides semantic information by itself.

Similarities (or dissimilarities) between objects are useful to understand how close they are. Distances (also called metrics) are measures of dissimilarity with some special properties such as symmetry and the triangle inequality, which make them more advantageous for many algorithms, because the search space can be reduced by triangularisation. Distance-based methods are a popular and powerful approach to inductive inference, since distances are a proper way to measure dissimilarity. 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 [11].

There are distances for virtually any kind of object, including complex or highly structured ones, such as tuples, sets, lists, trees, graphs, images, sounds, web pages, ontologies, etc. One challenging case in machine learning, but more especially in the area of inductive 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 may not only be useful in the area of inductive logic programming (ILP) [12] (e.g. first-order clustering [3]) or inductive programming in general, 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.

In this paper, we define a *transformation procedure* to convert semi-structured data to a (functional) term-based representation in XML which is suitable for term distances. We handle different degrees of structure, from purely flat data (from which we derive a hierarchy in two different ways) to originally hierarchical data. The advantage of term distances is that they are able to consider context and, some of them, are able to consider repetitions. This may be important in some problems where the discrepancies of some arguments are more or less similar depending on whether they appear in other contexts and also the depth where discrepancy appears. We also need to address some issues about order in XML documents, since terms are always ordered. Once this transformation is done we can directly apply term-based distances.

We perform experiments with Nienhuys-Cheng's distance [13] and Estruch et al's distance [5]. We first apply our approach to two flat datasets from the UCI repository which have some implicit hierarchy that the transformation uses and the distances are able to exploit. Second, we use some techniques from granular computing to obtain a clustering (dendrogram) of attributes to construct a hierarchy which is again exploited by the term distances. We compare the results to an approach not using the hierarchical structure of these problems by using Euclidean distances. Third, we also work with an originally hierarchical problem, which is converted from a functional representation (in Lisp) to an XML representation. In all cases, experiments are performed with a distance-weighted

$k$-nearest neighbour algorithm, using several exponents for the attraction function (the distance weight). We see that in some cases these term distances can highly improve the results over a flat approach.

This paper is organised as follows. Section 2 introduces the notation and analyses some previous distances proposed in the literature for trees, semi-structured data and terms, and very especially the two distances we will use, Nienhuys-Cheng's distance [13] and Estruch et al's distance [5]. Section 3 introduces the transformation from semi-structured data into the common XML representation which is suitable for term distances. Section 4 includes experiments on several datasets showing the cases where these distances can work better than a flat Euclidean distance (in case the latter can be used). Section 5 concludes the paper and relates to future work.

## 2 Preliminaries and related work

### 2.1 Notation

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 $\Pi$) is a family indexed on $\mathbb{N}$ (non negative integers) being $\mathcal{F}_n$ ($\Pi_n$) a set of $n-$adic function (predicate) symbols. Atoms and terms are constructed from the $\Sigma$ 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 $\lambda$, 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 $o$ ($\lambda$ 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 ($\neg Compatible(s, t)$).

### 2.2 Distances over terms

Since in this paper we are not interested in dealing with non-ground terms, and we require a normalised distance which produces a single number in order to be able to compare term distances with Euclidean distances, we will only work with the following distances over atoms/terms: the Nienhuys-Cheng's distance

[13] and the Estruch et al.'s distance [5]. Apart from them, J. Ramon et al. [15] defined another distance between non-ground atoms that relies on the *least general generalisation* (*lgg*) operator [14] and that computes a pair of real values as the distance between two atoms. A comparison of the three distances in terms of different characteristics, namely context sensitivity, ability to take repeated differences, complexity of differences and variables into account, composability, use of weights and normalisation, can be found in [5]. As mentioned, J. Ramon et al's distance will not be used because it does not retrun a real number but a pair, and it cannot be used in many algorithms using distances.

In what follows we briefly review the Nienhuys-Cheng's distance and the Estruch et al.'s distance. The first one is a bounded distance which takes the context in that differences occur into account but disregarding the number of times that these differences take place or their syntactic complexity. On the other hand, the distance proposed by Estruch et al. is a bounded distance for (non-)ground terms/atoms which takes the context of the differences into account, but also their syntactic complexity and how many times these differences occur.

**Definition 1.** ***(Nienhuys-Cheng distance [13])*** *Given two ground expressions $s = s_0(s_1, \ldots, s_n)$ and $t = t_0(t_1, \ldots, t_n)$, the Nienhuys-Cheng distance between them, denoted by $d_N(s,t)$, is recursively defined as*

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

For instance, if $s = p(a,b)$ and $t = p(c,d)$ then $d_N(s,t) = 1/4 \cdot (d(a,c) + d(b,d)) = 1/4(1+1) = 1/2$. Note that $d_N$ takes the depth of the symbol occurrences into account in such a way that differences occurring close to the root symbols count more.

Besides the context, Estruch et al.'s distance also considers other factors related to the occurrences where the differences appear and their complexity. In order to do this, the authors defined the concept of the set of syntactic differences of expressions $s$ and $t$ as:

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

Then, 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 $o \in O^\star(s,t)$ have. For this purpose, a special function called $Size'$ is provided:

**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,*

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

The context value of an occurrence $o$ in an expression $t$, $C(o;t)$, is used to consider 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. This concept 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*

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

It is proved in [5] that if $o \in O^\star(s,t)$ then $C(o;s) = C(o;t)$. So, in those cases, the context of an occurrence $o \in O^\star(s,t)$ is denoted as $C(o)$.

Repeated differences are handled through an equivalence relation ($\sim$) on the set $O^\star(s,t)$ defined as follows:

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

which produces a non-overlapping partition of $O^\star(s,t)$ into equivalence classes. Also, an order relation ($\leq$) in every equivalence class $O^\star_i(s,t)$ is defined as $\forall o_j, o_k \in O^\star_i(s,t), \; o_j \leq o_k \Leftrightarrow C(o_j) \leq C(o_k)$.

Additionally, the previous concepts are used to define another function which 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. Thus, given two expressions $s$ and $t$, the weight function $w$ is:

$$\forall o \in O^\star_i(s,t), \; w(o) = \frac{3f_i(o) + 1}{4f_i(o)}$$

where $i = \pi(o)$, $\pi(o)$ is the index of the equivalence class $o$ belongs to, and $f_i(o)$ is the position that $o$ has according to $\leq$.

Finally, the distance by Estruch et al. is defined as:

**Definition 4.** ***(Estruch et al. distance [5] )*** *Let $s$ and $t$ be two expressions, the distance between $s$ and $t$ is,*

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

For example, let $s = p(a,a)$ and $t = p(f(b), f(b))$ be two expressions. Then, $O^\star(s,t) = \{1,2\}$. Also, $C(1) = C(2) = 2 \cdot (2+1) = 6$.

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, $w(1) = 1$ and $w(2) = 7/8$. Finally,

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

### 2.3 Distances over trees and semi-structured data

The *edit distance* and the *tree alignment distance* are two well-known distances for comparing trees based on operations of deleting, inserting and relabeling nodes [2]. Given a cost function associated to each editing operation, the edit distance between two trees $T_1$ and $T_2$ is calculated as the cost of the optimal sequence of operations needed to transform $T_1$ into $T_2$, where a sequence is optimal if its cost in minimum. Analogously, the tree alignment distance between $T_1$ and $T_2$ is obtained by inserting nodes labeled by an empty label until both trees are isomorphic, then the resulting trees are superimposed giving a tree whose nodes are labeled by a pair of labels. Finally, the distance is given by the sum of the cost of each pair (using a certain cost function). Therefore, in both cases, the more complex the differences are, more symbols or nodes we need to edit or added, and, thus, the greater the computed distance is. In fact, many distances between trees that have been proposed only depend on this factor, since neither the presence of repeated differences nor their context have an effect on the value computed by the metric.

On the other hand, distances over semi-structured data (HTML, XML, and so on) have been mainly used for clustering documents and for detecting changes in XML documents. All these problems have been tackled by representing the documents as trees and, then, by applying tree distances. For instance, some approaches use the edit distance or some variant of it [17], [6], [18].

However, tree distances are not appropriate when each example is represented as a set, vector or hierarchy of features, since we need to align the features depending on their position and not merely by insert and delete operations. For instance, given the terms $t_1 = student(course('A'))$, $t_2 = student(course('B'))$ and $t_3 = teacher(course('A'))$, and considering the tree edit distance, $t_2$ and $t_3$ are at the same distance of term $t_1$ (only one relabeling operation is needed to transform $t_2$ or $t_3$ into $t_1$). However, $t_1$ and $t_2$ are "more similar" than $t_1$ and $t_3$ since they match up at the most external label (*student*). As we have shown in the previous section, for distances between atoms or terms, two subterms are just different when the topmost element of their tree representation is different. Also, the deeper the differences occur, the more similar the expressions are, and the lower their distance is. In this regard, we think that term distances may be more suitable, if the right transformation is applied to use them appropriately. Hence, in the rest of the paper we focus on them.

## 3 Transforming semi-structured data into a term-based representation using XML

Since we want to apply term distances to different kinds of data, from flat data to fully hierarchical data, we require a common language for representing all these types of data. Such a common language is XML.

Although we want to handle any degree of structure, we can distinguish two extreme situations:

– Flat data: Many datasets are given as a table of attribute-value pairs. However, a detailed inspection shows that some attributes are related to others and a hierarchy can be induced from them.
– Hierarchical data at source: Some other datasets consist of data with a hierarchical structure which is represented by a tree, a functional term (e.g. in Haskell or Lisp) or an "is-in" hiearchy.

Intermediate cases can be handled as well by using a mixture of the procedures.

## 3.1 Schema definition

XML documents and functional terms are not the same thing, so we need to make some decisions in order to use XML to represent functional terms, and also to adapt the previous types of data to a common representation. One of the key issues is how to handle depth, repetition and order, since some components in XML have order and some others not, and repetitions are allowed.

Given a hierarchy we have to be very careful when using term distances since some parts of the hierarchy might be empty, and we need to determine how an empty part is compared to a non-empty part. In other words, the structure may have different number of elements at the same level; therefore it is necessary to ensure that distance calculations are not affected by the absence of features or their order. This difficulty requires the creation of a general schema that allows each instance, with its own features, to be properly adjusted, without losing any element or content, and in a defined order. The schema we propose is an XML document that contains labels without content of all the elements that could exist for a given example.

```
...                                     ...
<COLOUR>                                <COLOUR>
  <INTERNAL>WHITISH</INTERNAL>            <INTERNAL>WHITISH</INTERNAL>
  <EXTERNAL>GREY</EXTERNAL>             </COLOUR>
</COLOUR>                               <PIGMENTATION>YES</PIGMENTATION>
...                                     ...
```

**Fig. 1.** Hierarchies using differents elements.

For instance, Figure 1 shows two hierarchies with different elements. Thus, the element <EXTERNAL> does not exist in the first hierarchy (left), whereas the element <PIGMENTATION> exists for the second hierarchy (right), but not for the first one. It is therefore necessary to create a schema that contains an integrated structure for both cases. The schema created for this example is:

```
<COLOUR></INTERNAL/></EXTERNAL/></COLOUR></PIGMENTATION/>
```

This allows, not only to have a general structure, but also to ensure the format, so that each instance must follow the schema. However, it is necessary to consider that not all instances can be easily adapted to XML; e.g. elements that are leaves with content cannot be directly adapted when the schema requires a subtree. In this situation, the element is added regardless of the schema structure. However, this fact does not affect the distance calculations because they are considered as different features.

Even though there are differences between a representation based in terms and one represented in XML, it is possible to convert one representation into the other. In XML, every term is represented as an element composed by a label and a value. That way, it is possible to convert a term into an XML element adding a label and inserting the term as the value. It is also possible to convert an XML document into a term disregarding the label and representing the value as a term. For instance, in Figures 2 and 4 below, we see how an XML document can represent two different types of hierarchies.

## 3.2 Deriving hierarchical XML schemas from flat data

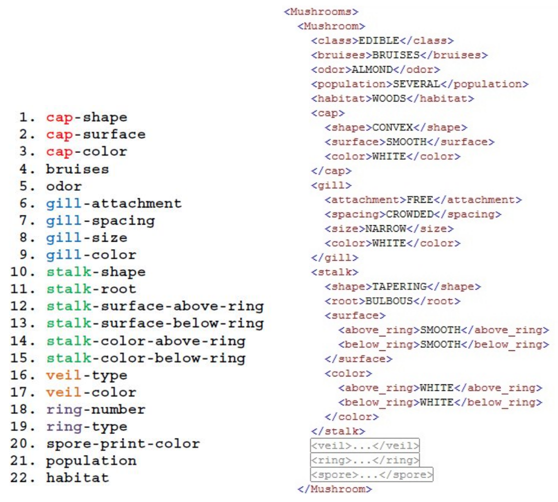
Flat data refers to attribute-value problems which are common in databases, machine learning, data mining, and other areas. In other words, data is presented in the form of a table of scalar values. In many cases, however, some structure can be inferred from these datasets, either because it was originally there and it still distills after the flattening process or because there are some features which can be grouped together according to some reason.

In fact, deriving a hierarchy from flat data is one of the problems that the area known as granular computing deals about [1]. In this work we consider three possible sources of structures from flat representations:

1. Value equality: Many datasets are given as flat data, but a detailed inspection shows that some attributes are related by the values they take. For instance, if two variables $X_1$ and $X_2$ can take the values $low, medium, high$, there is clearly a connection between them that can be exploited, especially through the use of equalities. For instance, we can define a condition or a rule using $X_1 = X_2$ which only makes sense if the datatypes are equal. This is the same as when variable repetitions are allowed in terms, like $f(a, X, X)$.
2. Name-induced hierarchies: In many cases we can find simple structures in the hierarchy of attributes, because of their names or their semantics. For instance, cap-shape, cap-surface and cap-color are attributes that contain 'cap' sub-features. In this way, sets of feature hierarchies can be created.
3. Attribute-similarity hierarchy: In other occasions, even if the names and values might be different, we can establish relations between the attributes which can be used to induce a structure. One approach is what is known as Watanabe-Kraskov variable agglomeration tree [16][9], which constructs a dendrogram (a hierarchical tree) using a similarity metric between attributes.

We will explore option 1 by the use (or not) of repetitions and, then, by applying the Estruch et al. distance and the Nienhuys-Cheng distance, respectively. We determine these relations in the Soybean dataset. The other two options will require specific transformations. Let us start with the name-induced hierarchy case.

Figure 2 shows an example of a hierarchy which is induced from the names of the original attributes. The dataset is 'mushroom', a flat dataset from the UCI machine learning repository [7], which has no structure originally (see Figure 2,



```
 1. cap-shape
 2. cap-surface
 3. cap-color
 4. bruises
 5. odor
 6. gill-attachment
 7. gill-spacing
 8. gill-size
 9. gill-color
10. stalk-shape
11. stalk-root
12. stalk-surface-above-ring
13. stalk-surface-below-ring
14. stalk-color-above-ring
15. stalk-color-below-ring
16. veil-type
17. veil-color
18. ring-number
19. ring-type
20. spore-print-color
21. population
22. habitat
```

```
<Mushrooms>
  <Mushroom>
    <class>EDIBLE</class>
    <bruises>BRUISES</bruises>
    <odor>ALMOND</odor>
    <population>SEVERAL</population>
    <habitat>WOODS</habitat>
    <cap>
      <shape>CONVEX</shape>
      <surface>SMOOTH</surface>
      <color>WHITE</color>
    </cap>
    <gill>
      <attachment>FREE</attachment>
      <spacing>CROWDED</spacing>
      <size>NARROW</size>
      <color>WHITE</color>
    </gill>
    <stalk>
      <shape>TAPERING</shape>
      <root>BULBOUS</root>
      <surface>
        <above_ring>SMOOTH</above_ring>
        <below_ring>SMOOTH</below_ring>
      </surface>
      <color>
        <above_ring>WHITE</above_ring>
        <below_ring>WHITE</below_ring>
      </color>
    </stalk>
    <veil>...</veil>
    <ring>...</ring>
    <spore>...</spore>
  </Mushroom>
```

**Fig. 2.** Name-induced hierarchy for the mushroom dataset. Left: the original attributes. Right: the induced hierarchy using common names.

left). After a grouping of common name prefixes we get a hierarchy, which is finally represented as an XML document (Figure 2, right). This document can then be processed as a functional term, e.g.:

```
    Mushroom(EDIBLE,BRUISES,ALMOND,SEVERAL,WOODS,cap(CONVEX,SMOOTH,WHITE),
gill(FREE,CROWDED,NARROW,WHITE), stalk(TAPERING,BULBOUS,surface(SMOOTH,SMOOTH),
color(WHITE,WHITE)),veil(PARTIAL,WHITE),ring(ONE,PENDANT),spore(print(BROWN)))
```

A second approach, which does not rely on the names of the attributes, is based on the idea of finding the similarities among attributes (previously referred as option 3). In the case of numerical attributes, this is typically done through correlation measures. In the case of nominal attributes, other measures of association can be used, such as a chi-square test. In any case, once the similarity matrix is given, we can convert it into a dissimilarity matrix and construct a dendrogram, as mentioned above.

This process is illustrated in Figure 3, where we see (on the left) a dendrogram for the 22 attributes of the mushroom dataset, using the chi-square measure as (dis)similarity. From this dendrogram, instead of using the whole hierarchy, which would place some variables too deep in the hierarchy (and very low weight), we can just arrange them by using the longest segments into only four groups, as we see in Figure 3 (right). This process can be automatised [10].

From there, we place each variable in a group, leading to this term:

```
    Mushroom(g1(WHITE,WHITE, PENDAT, AMOND, WHITE), group2(g2(PARTIAL,WHITE,
SMOOTH, BROWN, SEVERAL), group3(g3(SMOOTH, WOODS, ONE, WHITE )), group4(
g4(TAPERING, FREE,CROWDED,NARROW, BRUISES, BULBOUS, CONVES, SMOOTH)))
```

### 3.3 Deriving hierarchical XML schemas from hierarchical data

Apparently, this situation seems more direct, but we need to determine whether order, repetitions and labels are relevant, in order to determine the features and

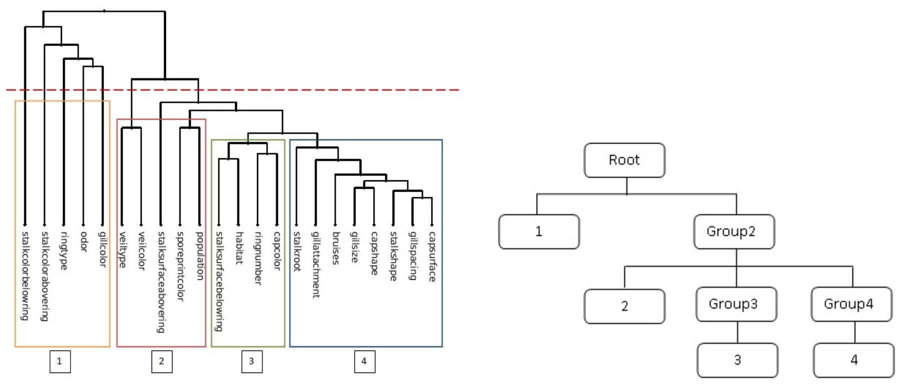


**Fig. 3.** Attribute-similarity hierarchy for the mushroom dataset. Left: the completed dendrogram. Right: the simplified hierarchy derived from it.

place them correctly in the XML document. A specific problem, as mentioned above, is how to treat empty parts in the hierarchy and how to compare them with non-empty parts. Our approach is based on a common schema for all the examples and the assumption that the based distance between an empty part and any non-empty part is 1 (and not given by the size of the non-empty part).

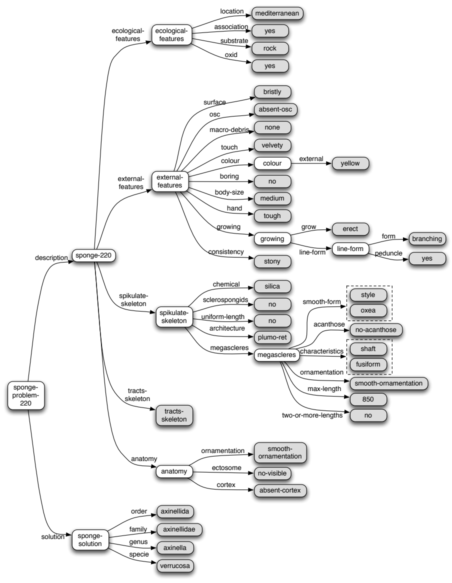
Figure 4 shows the ‘sponge’ dataset from the UCI repository [7], a complex dataset which enjoys a rich structure. Note that classical propositional methods are not applicable to this dataset. On the left of this figure we see the original hierarchy and on the right we see part of the resulting XML document (part of a single example). With this conversion, we can apply term distances.

## 4 Experiments

In this section we include some results using the distances introduced in section 2 with the several transformations seen in the previous section.

The simplest and most suitable algorithm for analysing the effect of the distances and transformations is the $k$-nearest neighbour ($k$-nn) algorithm, which just classifies an instance in the most common class of the $k$-nearest examples using the distance. The value of $k$ in the following experiment is calculated as the square root of the number of examples, which is a common choice in $k$-nn. We use a weighted $k$-nn variant, using an attraction function which gives more or less weight to each of the $k$-most nearest examples, defined as $\frac{1}{d^i}$ where $d$ is the distance and $i$ is the attraction parameter. A non-weighted $k$-nn is just given when $i = 0$. The greater $i$ is, the less important $k$ is. In the experiments, we will use several values for $i$, varying from 0 to 3. We consider three distances: the Nienhuys-Cheng distance, the Estruch et al. distance and the Euclidean distance, which are denoted by $d_N$, $d_E$ and $d_U$, respectively.

For the experimental evaluation, three datasets from the UCI repository [7] were used; two of them, Mushroom and Soybean, are flat and are used in several ways (flat and hierarchised by using two different methods); the third one, the



```
<SPONGES>
  <DEFINE-EPISODE>
    <EXTERNAL-FEATURES>...</EXTERNAL-FEATURES>
    <ECOLOGICAL-FEATURES>...</ECOLOGICAL-FEATURES>
    <SPICULATE-SKELETON>
      <DEF>SPICULATE-SKELETON</DEF>
      <CHEMICAL>SILICA</CHEMICAL>
      <ARCHITECTURE>RADIATE</ARCHITECTURE>
      <MEGASCLERES>
        <DEF>MEGASCLERES</DEF>
        <SMOOTH-FORM>
          <SET1>OXEA</SET1>
          <SET2>
            <DEF>ORTHOTRIAENA</DEF>
            <MAX-DIAMETER>50</MAX-DIAMETER>
          </SET2>
          <SET3>ANATRIAENA</SET3>
        </SMOOTH-FORM>
        <ACANTHOSE>NO-ACANTHOSE</ACANTHOSE>
        <CHARACTERISTICS>
          <DEF>CHARACTERS</DEF>
          <SHAFT>
            <SET1>NORMAL</SET1>
            <SET2>FUSIFORM</SET2>
          </SHAFT>
        </CHARACTERISTICS>
        <ORNAMENTATION>SMOOTH-ORNAMENTATION</ORNAMENTATION>
        <MAX-LENGTH>2800</MAX-LENGTH>
        <TWO-OR-MORE-LENGTH>YES</TWO-OR-MORE-LENGTH>
      </MEGASCLERES>
      <MICROSCLERES>ABSENT-MICROSCLERES</MICROSCLERES>
      <SCLEROSPONGIDS>NO</SCLEROSPONGIDS>
      <UNIFORM-LENGTH>NO</UNIFORM-LENGTH>
    </SPICULATE-SKELETON>
```

**Fig. 4.** Complex hierarchy for the sponge dataset. Left: the original hierarchy. Right: the hierarchy as an XML document.

Demospongiae (sponge) dataset, is a Lisp document which was transformed to XML by preserving the hierarchy and the values between attributes. For the Mushroom and Soybean datasets the tables include the means of a 10-fold cross validation experiment. In order to test the significance of these results, we perform a paired t-test between the methods (confidence 95%). If the difference of one method to another is significant, we include its acronym $(d_N, d_E, d_U)$ in the cell. In the sponge dataset we used a 60% of training examples and a 40% for test examples.

## 4.1 Mushroom dataset

This dataset includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. First, we compare the three distances using a flat schema, i.e. without using any kind of hierarchy, and with different values of $i$. These results are shown in Table 1. Here, the differences between distances are not significant.

Table 2 shows the results using the hierarchy induced from the attribute names. With this setting, we can find some differences. The performance of Estruch et al's and Nienhuys-Cheng's distance is optimum, while Euclidean increases the accuracy when we increase the value of $i$.

Finally, using the other method of grouping by considering the similarity between attributes, based on the chi-square measure (ChiSquaredAttributeEval

| | i=0 % | i=1 % | i=2 % | i=3 % |
|---|---|---|---|---|
| $d_N$ | 93.0 | 94.9 | 95.6 | 95.8 |
| $d_E$ | 93.0 | 94.9 | 95.6 | 95.8 |
| $d_U$ | 93.0 | 94.4 | 94.9 | 95.5 |

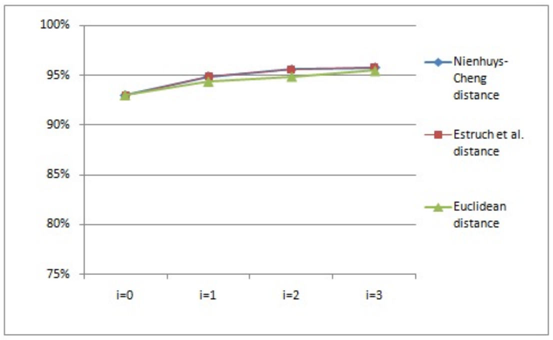


**Table 1.** Accuracies for mushroom without hierarchies.

| | i=0 % | i=1 % | i=2 % | i=3 % |
|---|---|---|---|---|
| $d_N$ | 99.8 | 99.8 | 99.9 | 99.9 |
| $d_E$ | 99.8 | 99.8 | 99.8 | 99.8 |
| $d_U$ | 93.0 | 94.4 | 94.9 | 95.5 |

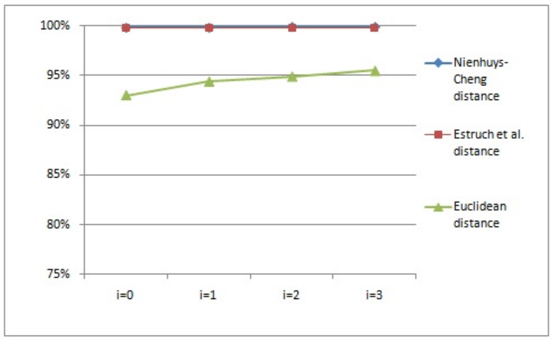


**Table 2.** Accuracies for mushroom with a hierarchy derived from the attribute names.

+ Ranker in Weka [8]) and the groups shown in Figure 3, we get the results shown in Table 3. Here, again, the distances between terms exploit this structure and get much better results than the Euclidean distance. Using similarity between attributes in the Euclidean distance could improve significantly its performance.

## 4.2 Soybean dataset

The soybean dataset is another 'flat' dataset and we do similarly as for the mushroom dataset. This dataset contains 307 instances and each instance has 35 attributes. Table 4 shows the results of three distances using the plain version of the dataset. In this case, the Euclidean distance obtains worse results than the two term distances. Table 5 shows the performance using a hierarchy which is derived from the attribute names. Here the behaviour of the three distances is similar and the differences are not statistically significant.

Finally, we used chi-square (ChiSquaredAttributeEval + Ranker in Weka [8]) to calculate the similarities, and grouping the variables from the dendrogram in

|  | i=0 % | i=1 % | i=2 % | i=3 % |
|---|---|---|---|---|
| $d_N$ | 99.5 | 99.8 | 100 | 100 |
| $d_E$ | 99.5 | 100 | 100 | 100 |
| $d_U$ | 93.0 | 94.4 | 94.9 | 95.5 |

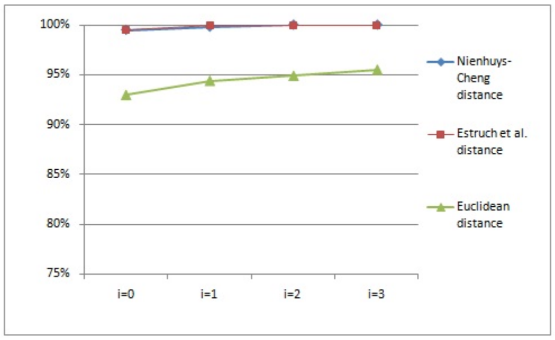


**Table 3.** Accuracies for mushroom with a hierarchy according to the similarity between attributes.

|  | i=0 % | i=1 % | i=2 % | i=3 % |
|---|---|---|---|---|
| $d_N$ | 75.3 | 91.3 | 93.9 | 95.8 |
| $d_E$ | 75.3 | 91.3 | 93.9 | 95.8 |
| $d_U$ | 75.3 | 87.1 | 91.3 | 92.6 |

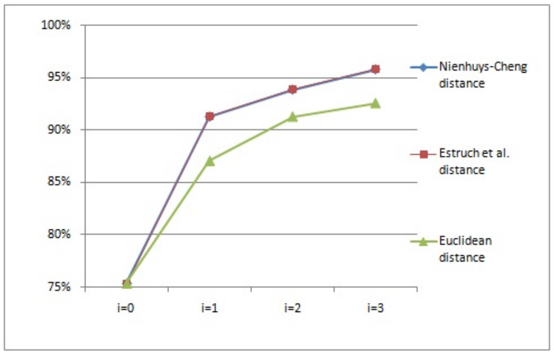


**Table 4.** Accuracies for soybean without hierarchies.

a similar way as we did with mushroom. With this process, we get the results shown in Table 6. In this setting, the atom-based distances obtain much better results than the Euclidean distance. Using similarity between attributes in the Euclidean distance could improve significantly its performance.

### 4.3 Demospongiae Dataset

Finally, we use the 503 examples of the Demospongiae dataset, which is originally hierarchical. Each instance is represented as a tree using terms in the language Lisp. Each tree has a depth between 5 and 8 levels and its number of leaves varies between 17 and 51 (see Figure 4).

From this dataset, a well formed XML structure that preserves the original structure was extracted, as discussed in the previous section. In order to do the transformation from Lisp, each line in Lisp was converted into one or several well-formed XML elements by assigning names and values to the elements for

|  | i=0 % | i=1 % | i=2 % | i=3 % |
|---|---|---|---|---|
| $d_N$ | 78.6 | 89.6 | 92.6 | 93.2 |
| $d_E$ | 76.0 | 88.0 | 91.6 | 93.5 |
| $d_U$ | 75.3 | 87.1 | 91.3 | 92.6 |

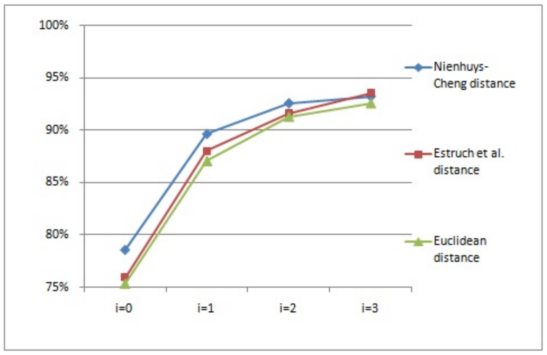


**Table 5.** Accuracies for soybean with a hierarchy derived from the attribute names.

|  | i=0 % | i=1 % | i=2 % | i=3 % |
|---|---|---|---|---|
| $d_N$ | 86.1 $d_U$ | 99.0 $d_U$ | 99.4 $d_U$ | 99.4 $d_U$ |
| $d_E$ | 86.1 $d_U$ | 99.0 $d_U$ | 99.4 $d_U$ | 99.4 $d_U$ |
| $d_U$ | 75.3 $d_N d_E$ | 87.1 $d_N d_E$ | 91.3 $d_N d_E$ | 92.6 $d_N d_E$ |

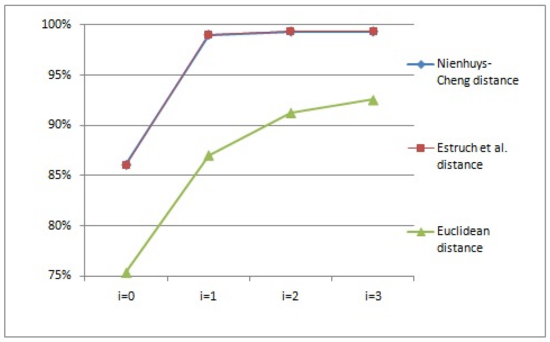


**Table 6.** Accuracy results of soybean with a hierarchy according to the similarity between attributes.

each feature described in Lisp in accordance to their hierarchy and order. Some specific processing was required:

– Each example in Lisp defined by label "define-episode" and an identifier "sponge :ID SPONGE-0" was transformed into an XML element: <DEFINE-EPISODE>, which in turn adds a child element: <SPONGE_ID>SPONGE-0</SPONGE_ID>.
– Each feature definition as "EXTERNAL-FEATURES DEFINE (EXTERNAL-FEATURES" was converted into a main element container of various elements in XML: <EXTERNAL-FEATURES>.
– Each simple feature such as "(BODY-SIZE SMALL)" was converted into an element <BODY-SIZE>SMALL</BODY-SIZE>.
– Additionally collections such as "(SET) GREY WHITISH", were converted to XML as follows <SET1>GREY</SET1> <SET2> WHITISH </SET2>.

Table 7 shows the classification results using the XML document. With this dataset we cannot apply directly Euclidean distance. We can see that (for values of $i$ greater than 0) Estruch et al.'s distance shows a better accuracy than Nienhuys-Cheng's distance. This is the best example to illustrate the differences between these two atom-based distances because data is purely semi-structured, and we perceive the effect of repetition. In the end, this dataset is useful to take advantage of the intrinsic properties of atom-based distances.

| | i=0 % | i=1 % | i=2 % | i=3 % |
|---|---|---|---|---|
| Nienhuys-Cheng distance | 60.9 | 71.3 | 74.3 | 73.8 |
| Distance between atoms | 60.9 | 73.3 | 78.7 | 77.7 |

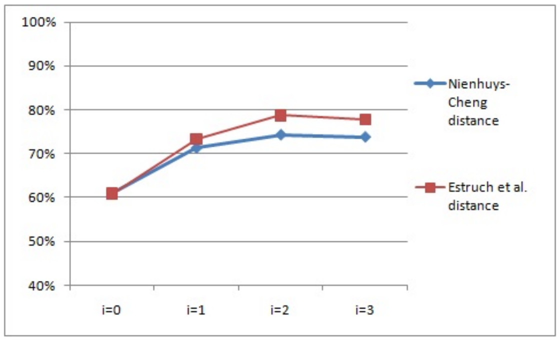


**Table 7.** Accuracies for the sponge dataset using its original hierarchy.

## 5 Conclusions

Tree distances are not generally appropriate for handling hierarchical datasets since they typically do not consider attribute names (the value for variable $X_3 = a$ can be matched with the same value for a different variable). In addition, they are based on the number of transformations to convert one tree into another, which is a meaningless concept in many hierarchy comparisons. We have seen that term distances, although initially defined in the area of inductive programming, can be used in a broader range of applications, provided we are able to find apropriate transformations for their term-based representation.

In this paper we have seen three transformations to adapt different degrees of structures and hierarchical data to be used with term distances. We have seen a method for constructing the hierarchy from the attribute names. This does not seem to provide good results. A second method uses the similarity between the attributes to construct a dendrogram from which a hierarchy is constructed. This second method improves the results of the flat dataset. A third, different transformation takes place when the original dataset has already a hierarchy. Here we can see the relevance of using repetitions or not. All these transformations are applied over XML schemas, so any dataset with any mixture or degree of flat and hierarchical information could be transformed and used with the term distances.

Summing up, we have seen a promising application of term distances to different types of datasets, which suggests that the use of term distances can be broader than it is now. In fact, as a future work, we plan to investigate the use of these distances for clustering and for other tasks. Nonetheless we believe that distances may have more potential applications in general inductive programming and also in other areas in programming language theory, such as 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).

## Acknowledgments

We thank the Spanish MICINN for projects TIN2010-21062-C02 and Consolider-Ingenio CSD2007-00022, and Generalitat Valenciana for project Prometeo/2008/051.

## References

1. A. Bargiela and W. Pedrycz. *Granular computing: an introduction.* Springer, 2003.
2. Philip Bille. A survey on tree edit distance and related problems. *Theoretical Computing Science*, 337:217–239, 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 *Proc. 13th ACM SIGPLAN Intl. Conf. on Functional programming, ICFP*, pages 3–14. ACM, 2008.
5. V. Estruch, C. Ferri, J. Hernández-Orallo, and M. Ramírez-Quintana. An Integrated Distance for Atoms. *Functional and Logic Prog.*, pages 150–164, 2010.
6. F. De Francesca, G. Gordano, R. Ortale, and A. Tagarelli. Distance-based clustering of XML documents. In *1st Intl. Workshop on Mining Graphs, Trees and Sequences (MGTS-2003)*, pages 75–78. L. De Raedt and T. Washio, editors, 2003.
7. A. Frank and A. Asuncion. UCI machine learning repository, 2010.
8. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten. The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.
9. A. Kraskov, H. Stogbauer, R.G. Andrzejak, and P. Grassberger. Hierarchical clustering based on mutual information. *Arxiv preprint q-bio/0311039*, 2003.
10. Peter Langfelder, Bin Zhang, and Steve Horvath. Defining clusters from a hierarchical cluster tree. *Bioinformatics*, 24:719–720, March 2008.
11. T. M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.
12. S. Muggleton. Inductive Logic Programming. *New Generation Computing*, 8(4):295–318, 1991.
13. S.H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Artificial Intelligence.* Springer, 1997.
14. G. Plotkin. A note on inductive generalisation. *Machine Intelligence*, 5:153–163, 1970.
15. 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.
16. S. Watanabe. *Knowing and guessing: A quantitative study of inference and information.* Wiley New York, 1969.
17. G. Xing, Z. Xia, and J. Guo. Clustering xml documents based on structural similarity. In *Advances in Databases: Concepts, Systems and Applications*, volume 4443 of *LNCS*, pages 905–911. Springer, 2007.
18. J. Y. Cai Y. Wang, D. J. Dewitt. X-diff: an effective change detection algorithm for xml documents. In *19th Intl. Conf.on Data Engineering*, pages 519–530, 2003.