Distinguishing and Classifying from n-ary Properties

Pascal Préa, Monique Rolbert

Distinguishing and Classifying from n-ary Properties

Pascal Préa
École Centrale Marseille,
Laboratoire d'Informatique Fondamentale de Marseille,
LIF, CNRS UMR 7279,
Technopôle de Château-Gombert, 38, rue Joliot-Curie,
13451 Marseille Cedex 20, France

Monique Rolbert
Aix-Marseille Université,
Laboratoire d'Informatique Fondamentale de Marseille,
LIF, CNRS UMR 7279,
Marseille, France
{prea,rolbert}@lif.univ-mrs.fr

Abstract

We present a hierarchical classification based on n-ary relations of the entities. Starting from the finest partition that can be obtained from the attributes, we distinguish between entities having the same attributes by using relations between entities. The classification that we get is thus a refinement of this finest partition. It can be computed in $O(n + m^2)$ space and $O(n \cdot p \cdot m^{5/2})$ time, where n is the number of entities, p the number of classes of the resulting hierarchy (p is the size of the output; $p < 2n$) and m the maximum number of relations an entity can have (usually, $m \ll n$). So we can treat sets with millions of entities.

Keywords: Classification, Data Analysis, Hierarchy, Ultrametric, Computational Linguistics, Generation of Referring Expressions.

1 Introduction

Many algorithms and methods in classification and data analysis are based on the attributes of the entities (see Barthélemy and Guénoche 1991; Gordon 1999; Jardine and Sibson 1971).For instance, classical taxonomy (see Buffon 1749; Linné 1735) classifies living species according to some attributes, such as having hair, feathers, or scales. Ecology, i.e. relations between animals (like *who eats who* or *who parasitizes who*) or between an animal and its surroundings, is a meta-knowledge which is not taken into account by taxonomy. Since it seems that we are living a mass extinction event,

and although such an event is measured by the number of disappearing taxa, ecology is becoming more and more important. When an animal disappears, the impact of its extinction is more related to its ecological role than to its place in the evolution tree (the quasi-extinction of tigers does not affect lions, although tigers and lions are very close species, but the extinction of gnus would have heavy consequences for lions). Another example of the importance of links and relations between entities is that, with the growing influence of social networks, it is often said that the identity of a person is more who, and how, he is linked to than his own qualities.

In this paper, we present a hierarchical classification of entities using not only attributes but also relations between the entities. Actually, this idea comes from previous work (see Rolbert and Préa 2009) in computational linguistics: we tailor a solution to a classical problem in computational linguistics (the generation of referring expressions) and show that this solution yields an efficient method for classifying objects.

We first present the general purpose of generating referring expressions. Then, in section 3, we give a precise definition of distinguishability (i.e. we give necessary and sufficient conditions for an entity to admit a referring expression) that takes into account n-ary relations between entities. We show that this definition yields a hierarchical classification of the entities. In section 4, we give an $O(n + m^2)$ space and $O(n \cdot p \cdot m^{5/2})$ time algorithm to compute this hierarchy, where n is the number of entities, p the number of classes of the resulting hierarchy (p is the size of the output; $p < 2n$) and m the maximum number of relations an entity can have (usually, $m \ll n$). This algorithm is efficient enough to treat data sets with millions of entities. In the penultimate section, we will apply our method to classical attribute data, namely the Fisher iris data (from Fisher 1936). In the last section, we will mention some related work and possible extensions.

2 The generation of referring expressions

The *generation of referring expressions* has been a classical task in natural language processing for more than twenty years. Its aim is to generate a definite description (the *referring expression*) which designates one and only one entity among others in a context set like a scene or a discourse (see Dale 1989; Deemter 2002; Krahmer et al. 2003; Croitoru and Deemter 2007). An entity which admits such a description is said to be *distinguishable*.

Let us see this more precisely in the example in Figure 1.

In this scene, the entities are tables (t_1 and t_2), cups ($c_1 \dots c_6$), balls ($b_1 \dots b_5$), a flower (f) and the floor. For the sake of simplicity, let us suppose that this scene is described using unary relations (like *is a table* or *is a flower*), and only two binary relations: *being in/containing*, *being on/having on*. So, we have:

t_1 is a table which has c_1 and c_2 on it.

c_1 is a cup which is on t_1 and contains b_1

b_2 is a ball which is in c_2

b_3 is a ball which is in c_3

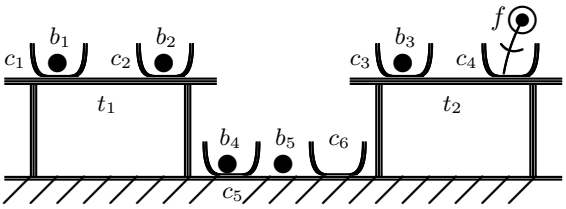

Figure 1: A simple scene

b_5 is a ball which is on the floor.

...

From this description, one can designate these entities by using the following referring expressions:

f: *The flower*

b_5: *The ball on the floor*

c_4: *The cup with a flower in it*

c_5: *The cup on the floor with a ball in it*

c_6: *The empty cup*

b_3: *The ball in a cup which is on a table on which there is a cup containing a flower*

...

The aim of the generation of referring expressions is to make a computer do the same thing, and more precisely to produce an expression close to what would be produced by a human being. Following this goal, most works in generation of referring expressions do not focus on formal determination of what is a distinguishable entity, especially when this distinguishability does not have a "natural expression" (see Gardent 2002; Khan et al. 2009; Mitchell 2009). For instance, the entity b_3 may be considered as non-distinguishable, since its referring expression is rather long (a human being would not use it). Other works (see Khramer et al. 2003; Deemter and Krahmer 2006; Rolbert and Préa 2009) tend to explore, in a systematic and complete way, all the relations which can yield distinguishability. We will apply this approach to classification.

3 An iterative definition for distinguishability and a hierarchical classification of entities

Intuitively, an entity e_1 is distinguishable from an entity e_2 in two cases:

- e_1 and e_2 do not have the same set of properties (we will say that e_1 is *0-distinguishable* from e_2). In the example of Figure 1, the entity b_3 is 0-distinguishable from f since b_3 has a property (*is a ball*) that f does not have. More generally, in the example of Figure 2, the tables t_1 and t_2 are 0-distinguishable one from the other since t_1 has 5 cups on it and t_2 has 3 cups on it.

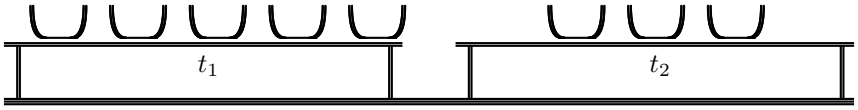

Figure 2: Another simple scene

- otherwise, e_1 and e_2 are in relation (we will see precisely how below) with at least two distinguishable entities e'_1 and e'_2. In this case, we will say that e_1 is $(k+1)$-*distinguishable* from e_2 if e'_1 is *k-distinguishable* from e'_2. In the example of Figure 1, c_3 is 1-distinguishable from c_4 since c_3 contains b_3 which is 0-distinguishable from f which is in c_4.

Let $E = \{e_1, e_2, \ldots\}$ be a finite set (whose elements are called *entities*) with a finite set $F = \{f_1, f_2, \ldots\}$ of boolean functions ($f_i : \overbrace{E \times \ldots \times E}^{n_i} \rightarrow \{True, False\}$) called *relations*.

A *property* is a relation, together with a rank (the argument's position); we denote by p_q the property built from relation p and rank q. For instance, the fact e_1 *gives* e_2 to e_3 corresponds to the relation $\mathsf{give} : E \times E \times E \rightarrow \{True, False\}$ such that $\mathsf{give}(e_1, e_2, e_3) = True$, and e_1 has the property give with rank 1 (denoted by give_1), e_2 has the property give_2 and e_3 has the property give_3. So, e_1, e_2 and e_3 do not have the same set of properties. Conversely, if e_1 gives x_1 to y_1 and e_2 gives x_2 to y_2, e_1 and e_2 have the same property (give_1).

For an entity e, we denote by $\mathcal{P}(e)$ the set of its properties. Given an entity e and a property p_q of $\mathcal{P}(e)$, we will say that a tuple of entities $t = (x_1, \ldots, x_{q-1}, x_{q+1}, \ldots, x_p)$ *matches* p_q with e if $p(x_1, \ldots, x_{q-1}, e, x_{q+1}, \ldots, x_p)$ is true. For instance, with the fact e_1 *gives* e_2 to e_3 (that we represent by $\mathsf{give}(e_1, e_2, e_3) = True$), the tuple (e_1, e_3) matches give_2 with e_2.

For an entity e and a property p_q of $\mathcal{P}(e)$, we denote by $\mathcal{T}(e, p_q)$ the set of all tuples that match p_q with e. If $p_q \notin \mathcal{P}(e)$, $\mathcal{T}(e, p_q) = \emptyset$. Given a set X, we denote by $|X|$ the number of elements of X.

Definition 1. *k-distinguishability D_k:*
*An entity e_1 is **0-distinguishable** from an entity e_2 (we denote it by $e_1\ D_0\ e_2$) if there exists a property p_q such that $|\mathcal{T}(e_1, p_q)| \neq |\mathcal{T}(e_2, p_q)|$.*
*An entity e_1 is **k-distinguishable** ($k > 0$) from an entity e_2 (we denote it by $e_1\ D_k\ e_2$) if $e_1\ D_{k-1}\ e_2$ or if there exists a property p_q in $\mathcal{P}(e_1)$ such that, if we set $\mathcal{T}(e_1, p_q) = \{t_1, \ldots, t_s\}$:*

For every ordering $(t'_1, \ldots, t'_s)$ of $\mathcal{T}(e_2, p_q)$[1]:

There exists i in $[1 \ldots s]$, j in $[1, \ldots r]$ such that x_j is $(k-1)$-distinguishable from y_j, where $t_i = (x_1, \ldots x_j \ldots x_r)$ and $t'_i = (y_1, \ldots y_j \ldots y_r)$.

[1]We suppose that $|\mathcal{T}(e_2, p_q)| = |\mathcal{T}(e_1, p_q)|$ since otherwise $e_1\ D_0\ e_2$.

We say that an entity e is *distinguishable* from an entity e' if $e\, D_k\, e'$ for some $k \geq 0$, that e is k-*confusable* with e' if e is not k-distinguishable from e', and that e is *confusable* with e' if e is not distinguishable from e'. We denote this, respectively, by $e\, D\, e'$, $e\, C_k e'$ and $e\, C\, e'$.
We denote by $\kappa(e_1, e_2)$ the smallest k such that $e_1 D_k e_2$.

Claim 1. *For any entities e_1 and e_2, and every $k \geq 0$, $e_1 D_k e_2$ if and only if $e_2 D_k e_1$. Equivalently, $\kappa(e_1, e_2) = \kappa(e_2, e_1)$.*

Definition 1 may seem rather complicated, especially the *There exists a property/ For every ordering* combination. One way to explain the underlying ideas is to interpret this definition through a game theoretic point of view (see Osborne 2009). We consider e_1 and e_2 as players of the following game[2]: At "round" k, if e_1 is not already distinguishable from e_2, his goal is to be k-distinguishable from e_2, and the goal of e_2 is to be k-confusable with e_1. To achieve their goals, e_1 exhibits a set of relationships it thinks to be specific to tself, and e_2 replies by exhibiting a set of relationships which is identical to those of e_1.

Let us see some examples. In the scene in Figure 3, there are tables, cups, balls and flowers. We use the same hypothesis as in Figure 1, i.e. we suppose that the scene is described using only unary relations (like *is a table*) and the binary relations $\mathsf{is\_in}_1$, $\mathsf{is\_in}_2$ (*is_in/has_in*), $\mathsf{is\_on}_1$ and $\mathsf{is\_on}_2$ (*is_on/has_on*).

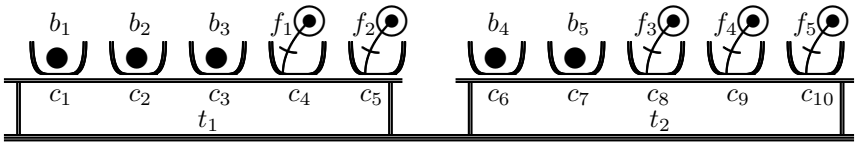

Figure 3: Another scene

As definition 1 is iterative, its application takes several steps. Each step consists in distinguishing more entities from the others: many objects are 0-confusable but few are 4-confusable. At the end, there only remain entities which are not distinguishable. These steps are the following:

1. With our definition, the tables t_1 and t_2 are 0-confusable (both are tables with five objects on them) but they are both 0-distinguishable from all the other entities. Similarly, the cups are 0 distinguishable from the flowers, the balls and the tables, and so on.

2. The cups c_1, c_2, c_3, c_6 and c_7 are 1-distinguishable with the other cups since they are in relation (via *has_in*) with entities (the balls $b_1 \ldots b_5$) which are 0-distinguishable from all the entities (the flowers) which are in relation via *has_in* with the cups c_4, c_5, c_8, c_9 and c_{10}. The cups with balls inside them are 1-distinguishable from cups with flowers inside them.

[2]Actually, there are n^2 simultaneous games, where n is the number of entities.

3. The tables t_1 and t_2 are 2-distinguishable one from the other: they both have five cups on them, but there do not exist three cups on t_2 which are simultaneously 1-confusable with c_1, c_2 and c_3 (and conversely, there do not exist three cups on t_1 which are simultaneously 1-confusable with c_8, c_9 and c_{10}). So, for all the orderings $(c'_1, c'_2, c'_3, c'_4, c'_5)$ of $\mathcal{T}(t_2, has\_on_1) = \{c_6, c_7, c_8, c_9, c_{10}\}$, it is impossible to have simultaneously $c_1C_1c'_1$, $c_2C_1c'_2$ and $c_3C_1c'_3$. Conversely, for all the orderings $(c''_1, c''_2, c''_3, c''_4, c''_5)$ of $\mathcal{T}(t_1, has\_on_1) = \{c_1, c_2, c_3, c_4, c_5\}$, it is impossible to have simultaneously $c''_3C_1c_8$, $c''_4C_1c_9$ and $c''_5C_1c_{10}$.

4. The cups c_1, c_2 and c_3 are 3-distinguishable from c_6 and c_7 (and reciprocally) since they are on t_1 which is 2-distinguishable from t_2. Similarly, the cups c_4 and c_5 are 3-distinguishable from the cups c_8, c_9 and c_{10}.

5. The balls b_1, b_2 and b_3 are 4-distinguishable from the balls b_4 and b_5 since they are in cups which are 3-distinguishable from the cups containing b_4 and b_5. Similarly, the flowers f_1 and f_2 are 4-distinguishable from f_3, f_4 and f_5.

6. No 4-confusable entity is 5-distinguishable. The iteration stops.

We notice that the flowers and the balls are used to distinguish t_1 and t_2 one from the other (steps 2 and 3) and that the distinguishability of t_1 and t_2 is used to partition the flowers and the balls (step 5). But although there is a cycle, there is no infinite loop.

It can be proven that for every $k \geq 0$, C_k is an equivalence relation. In addition, if two entities are k-distinguishable, then they are k'-distinguishable for every $k' > k$. So:

Property 1. *Definition 1 yields a hierarchy on the entity set. Equivalently:* $\mathcal{K} - \kappa$ *is a pseudo-ultrametric*[3]*, where* $\mathcal{K}$ *is the greatest* $\kappa(x, y)$.

The dendrogram obtained from the example of Figure 3 is the one in Figure 4. In this example, one can see that unary properties give the first level of the hierarchy ($\kappa = 0$): from left to right, these four classes are the tables ($\mathcal{C}_0^1 = \{t_1, t_2\}$), the cups ($\mathcal{C}_0^2 = \{c_1, c_2 \ldots, c_{10}\}$), the balls ($\mathcal{C}_0^3 = \{b_1, b_2 \ldots, b_5\}$) and the flowers ($\mathcal{C}_0^4 = \{f_1, f_2 \ldots, f_5\}$). The use of n-ary relations give four more levels of classification.

At level $\kappa = 1$, the class $\mathcal{C}_0^2$ is partitioned into $\mathcal{C}_1^1 \cup \mathcal{C}_1^2$, where $\mathcal{C}_1^1 = \{c_1, c_2, c_3, c_6, c_7\}$ is the class of the cups containing a ball and $\mathcal{C}_1^2 = \{c_4, c_5, c_8, c_9, c_{10}\}$ is the class of the cups containing a flower.

At level $\kappa = 2$, the class $\mathcal{C}_0^1$ is partitioned into $\mathcal{C}_2^1 \cup \mathcal{C}_2^2 = \{t_1\} \cup \{t_2\}$.

At level $\kappa = 3$, $\mathcal{C}_1^1$ is partitioned into $\mathcal{C}_3^1 \cup \mathcal{C}_3^2$, where $\mathcal{C}_3^1 = \{c_1, c_2, c_3\}$ is made of the cups containing a ball which are on t_1 and $\mathcal{C}_3^2 = \{c_6, c_7\}$ is made of the cups containing a ball which are on t_2. Similarly, $\mathcal{C}_1^2$ is partitioned into $\mathcal{C}_3^3 \cup \mathcal{C}_3^4 = \{c_4, c_5\} \cup \{c_8, c_9, c_{10}\}$.

At level $\kappa = 4$, $\mathcal{C}_0^3$ is partitioned into $\mathcal{C}_4^1 \cup \mathcal{C}_4^2$, where $\mathcal{C}_4^1 = \{b_1, b_2, b_3\}$ are the elements e of $\mathcal{C}_0^3$ which are contained in elements of $\mathcal{C}_3^1$, i.e. $\mathcal{C}_4^1$ are the balls which are in cups on t_1. $\mathcal{C}_4^2 = \{b_4, b_5\}$ is the class of the balls in cups on t_2. Similarly, $\mathcal{C}_0^4$ is

[3] $d = \mathcal{K} - \kappa$ is not an ultrametric since $x \neq y$ does not imply $d(x, y) \neq 0$. In order to get an ultrametric, one can define $d'(x, y) = 0$ if $x = y$ and $d'(x, y) = d(x, y) + 1$ otherwise.

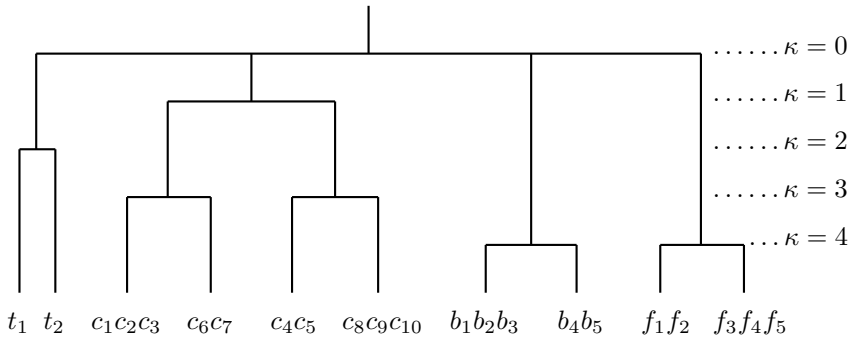

Figure 4: The dendrogram for the scene in Figure 3

partitioned into $\mathcal{C}_4^3 \cup \mathcal{C}_4^4 = \{f_1, f_2\} \cup \{f_3, f_4, f_5\}$ (the flowers in cups on t_1 and the flowers in cups on t_2).

One characteristic of our ultrametric is that the distance between two entities depends on all of the entities in the context. So if we take out one entity, the whole hierarchy may change. In the example in Figure 3, if we take out b_1 and f_5, we obtain the dendrograms in Figure 5 and Figure 6. One can see that the dendrograms in Figure

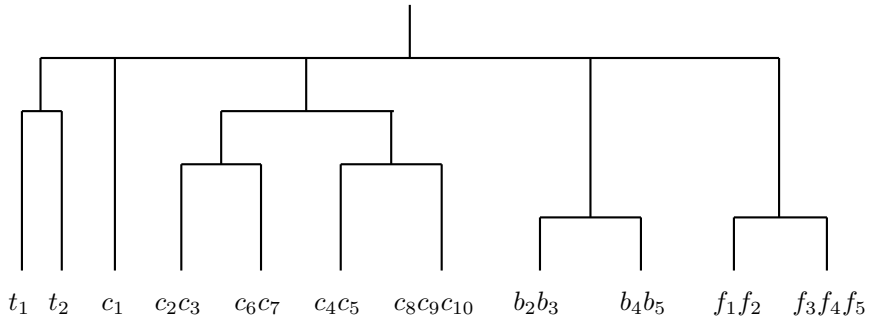

Figure 5: The dendrogram for Figure 3 if b_1 is taken out

4, Figure 5 and Figure 6 are structurally different.

Figure 5 and Figure 6 also illustrate another characteristic of definition 1. Actually, looking at the dendrogram in Figure 4, it may seem that 0-distinguishability only derives from unary properties like *being a table*, or *being a cup*, etc as it is often the case in classical taxonomy, where attributes are ordered and these ones would be considered as "main" attributes. In our work, properties are not ordered: in Figure 5 and Figure 6, empty cups (c_1 in Figure 5 and c_1, c_{10} in Figure 6) are 0-distinguishable from not empty cups, as they are 0-distinguishable from tables and flowers. Actually, in the dendrogram of Figure 4, the class $\mathcal{C}_0^1$ is made of entities which are tables and have 5

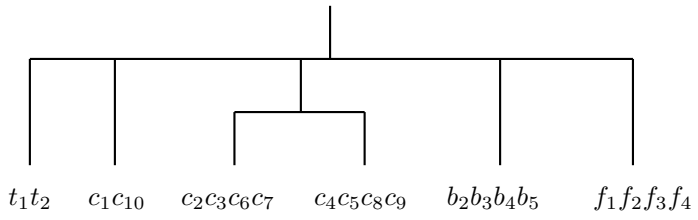

Figure 6: The dendrogram for Figure 3 if b_1 and f_5 are taken out

objects on them, the class $\mathcal{C}_0^2$ is made of cups which are on something and have an object inside them, the class $\mathcal{C}_0^3$ is made of balls which are in something, and the class $\mathcal{C}_0^4$ is made of flowers which are in something.

We now take as entity set E the elements of an circular array $T[0 \dots n-1]$, with the relations equals_1$(T[i])$ and is_near$(T[i], T[j])$, which is true if $0 < |i-j| \leq k$ (the operations are made modulo n). So every entity e has property is_near$_1$ and is_near$_2$, and, for each of these properties, there are $2k$ tuples (made of one element) that match it with e. Some entities also have property equals_1. Let us consider, with $n = 14$ and $k = 2$:

$$T = [1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]$$

- At Step 0, E is partitioned into $\{T[0], T[3], T[8]\}$ (the elements with value 1) on one hand, and the other elements on the other hand.
- At Step 1, E is partitioned into $[a, b, b, a, c, c, c, c, a, c, c, 11, c, c]$, where a denotes the elements which are equal to 1, b the elements which have two "neighbors" with value 1, c those which have one neighbor with value 1, and 11 the (unique) element all of whose neighbors have value 0.
- At Step 2, E is partitioned into $[f, b, b, f, 4, e, e, e, 8, d, d, 11, d, 13]$. The entity marked 13 is the unique one to have one neighbor in each of the classes a, b, c and 11; the entity marked 4 has one neighbor in a, one in b and two in c; the entity marked 8 is the only element of class a not to have a neighbor in class b, the class f is made of the other elements of class a; class d is made of the elements of class c having two neighbors in c, one in class 11 and one in a; class e is made of entities of class c which have three neighbors in class c and one in class a.
- At step 3, all entities become distinguishable: $T[0]$ is the unique element of class f with a neighbor in class 13; $T[2]$ is the only element of class b to have a neighbor in class 4; $T[5]$ is the unique element of class e with a neighbor in class f; and so on.

So, we have the dendrogram of Figure 7. Our method does not explicitly take into account symmetric relations, but it is possible to transform a symmetric relation into a

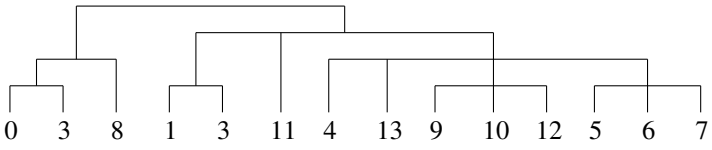

Figure 7: The dendrogram for the array T

non-symmetric one. In the last example (which corresponds to dendrogram of Figure 7), we have a symmetric relation ($\mathsf{is\_near}(e, e') \iff \mathsf{is\_near}(e', e)$). Note that we have represented a fact like e *is near* e' by $\mathsf{is\_near}(e, e')$ and $\mathsf{is\_near}(e', e)$; so we have indicated that (e') matches $\mathsf{is\_near}_1$ and $\mathsf{is\_near}_2$ with e and that (e) matches $\mathsf{is\_near}_1$ and $\mathsf{is\_near}_2$ with e'. It is possible to avoid this "duplication of information" by ignoring property $\mathsf{is\_near}_2$ which is equivalent to property $\mathsf{is\_near}_1$. This simplification is impossible for n-ary relations, and, more generally, this technique is not possible for n-ary symmetric relations (the size of the data would be multiplied by $n!$).

Another approach to represent a symmetric relation is the use of a fictive additional entity; this is similar to the transformation of a database with n-ary relations into a equivalent database with only binary relations. For instance, a symmetric n-ary relation R, such that $R(\{e_1, e_2, \ldots, e_n\})$ and $R(\{e'_1, e'_2, \ldots, e'_n\})$ are true can be represented by a binary (non-symmetric) relation B with $B(e_1, x)$, $B(e_2, x)$,... $B(e_n, x)$, $B(e'_1, x'), \ldots B(e'_n, x')$ true, where x and x' are additional entities. The size of the data then remains nearly unchanged, but the hierarchy is slightly changed: for instance, if e_1 is k-distinguishable from e'_1, then $e_2, \ldots e_n$ should be $(k+1)$-distinguishable from $e'_2, \ldots e'_n$. But in fact, x would be $(k+1)$-distinguishable from x' and $e_2, \ldots e_n$ would be $(k+2)$-distinguishable from $e'_2, \ldots e'_n$.

For entities, having a property (whatever its arity is) can be seen as an attribute (an entity has or does not have this property/attribute). We can see that our method first builds a partition (the one we get with 0-distinguishability), which is the finest partition that one can get from these attributes. Then the following steps refine this partition. It is possible to obtain the first partition (the one which corresponds to 0-distinguishability) by a "classical" method using attributes. For instance, if we look at the usual classification of animals, there is an order on the attributes: the attribute *having a notochord*, which defines the chordates, must be considered before the attribute *having a head*, which defines the craniates, which is prior to *having a backbone*, which defines the vertebrates. Our method can be used to continue this partition by taking into account the interactions between species. If we have classified animals (using only attributes), then we can, for instance, classify carnivore by taking into account what kind of animals they eat, and also herbivore (depending on what animals eat them).

4 An efficient algorithm to compute the hierarchy

In this section, we give an algorithm which computes the hierarchy. An implementation of this algorithm can be found at: `https://github.com/pascalprea/Classification`. This algorithm takes as entry a finite set E of entities, with the sets $\mathcal{P}(e)$ and $\mathcal{T}(e, p_q)$ for each $e \in E$ and $p_q \in \mathcal{P}(e)$. Sets are given with their cardinality. This algorithm relies on *testing k-confusability*, i.e. determining if two entities are k-confusable or k-distinguishable for a given integer k. We first study this core test.

4.1 Testing k-confusability

If $k = 0$, we only have to check for every property p_q if $|\mathcal{T}(e_1, p_q)| \neq |\mathcal{T}(e_2, p_q)|$. This can be done in $O(|\mathcal{P}(e_1)|)$, assuming that the sets $\mathcal{P}(e)$ are initially sorted. Sorting these sets takes $O(nm \log m)$ time, where n is the number of entities and m the maximum number of properties an entity can have. Since our algorithm takes $O(n \cdot p \cdot m^{5/2})$ time (p is the size of the output), sorting the sets $\mathcal{P}(e)$ does not increase the complexity of our algorithm.

If $k > 0$, we suppose that all the couples (e, e') of entities such that $\kappa(e, e') < k$ have been already computed; so $(k-1)$-confusability can be tested in $O(1)$.

At first glance, it may seem that definition 1 yields an exponential algorithm since we have to test all the permutations of the set $[1 \dots s]$. But in fact, testing whether $e_1 D_k e_2$ or not can be done in the following way:
For every property p_q of $\mathcal{P}(e_1)$:

1. We construct a bipartite graph $G_{p_q}^{e_1,e_2} = (V_{p_q}^{e_1,e_2}, E_{p_q}^{e_1,e_2})$, where:
   - $V_{p_q}^{e_1,e_2} = \mathcal{T}(e_1, p_q) \cup \mathcal{T}(e_2, p_q)$
   - for $t = (x_1, \dots x_r)$ in $\mathcal{T}(e_1, p_q)$ and $t' = (y_1, \dots y_r)$ in $\mathcal{T}(e_2, p_q)$, $\{t, t'\} \in E_{p_q}^{e_1,e_2}$ if for all $i \in [1 \dots r]$, x_i is $(k-1)$-confusable with y_i.

   The graph $G_{p_q}^{e_1,e_2}$ can be constructed in $O(|\mathcal{T}(e_1, p_q)| \cdot |\mathcal{T}(e_2, p_q)| \cdot r) = O(|\mathcal{T}(e_1, p_q)|^2 \cdot r)$. It captures how many tuples in relation with e_1 and tuples in relation with e_2 through the same relation are pairwise confusable.

2. We check if $G_{p_q}^{e_1,e_2}$ admits a perfect matching (i.e. if there exists a set X of edges such that every vertex is incident with exactly one edge of X). This can be done in $O((|\mathcal{T}(e_1, p_q)| + |\mathcal{T}(e_2, p_q)|)^{5/2}) = O(|\mathcal{T}(e_1, p_q)|^{5/2})$ with the algorithm of Hopcroft and Karp (1973)[4].

Claim 2. *The entities e_1 and e_2 are k-distinguishable if and only if one graph $G_{p_q}^{e_1,e_2}$ does not admit a perfect matching.*

[4]There exists an $O(n^{1.5}\sqrt{m/\log n}$ algorithm to compute a maximal matching in a bipartite graph with n vertices and m edges (see Alt et al. 2001) but this algorithm improves the one of Hopcroft and Karp (1973) only for dense graphs. In addition, the graphs $G_{p_q}^{e_1,e_2}$ are generally very small.

Proof. Suppose that $\mathcal{T}(e_1, p_q) = (t_1, \ldots, t_k)$. If $(t'_1, \ldots, t'_k)$ is an ordering of $\mathcal{T}(e_2, p_q)$ which does not satisfy the condition of Definition 1, then $\{\{t_1, t'_1\}, \{t_2, t'_2\}, \ldots \{t_k, t'_k\}\}$ is a perfect matching of $G_{p_q}^{e_1,e_2}$; and reciprocally. □

Claim 3. *Knowing $(k-1)$-confusability/distinguishability of every pair of entities, testing k-confusability can be done in $O(r \cdot m^2 + m^{5/2})$ time and $O(m^2)$ space, where m is the maximum number of properties an entity can have ($m = \max_{e \in E} \sum_{p_q \in \mathcal{P}(e)} |\mathcal{T}(e, p_q)|$), and r the greatest arity of properties.*

Proof. Let e and e' be two $(k-1)$-confusable entities. For every property p_q of $\mathcal{P}(e_1)$ $(= \mathcal{P}(e_2))$, the algorithm first constructs the graph $G_{p_q}^{e_1,e_2}$ in $O(|\mathcal{T}(e_1, p_q)|^2 \cdot r)$. The graph $G_{p_q}^{e_1,e_2}$ can be represented by a $|\mathcal{T}(e_1, p_q)| \times |\mathcal{T}(e_1, p_q)|$ matrix. Then the algorithm tests if $G_{p_q}^{e_1,e_2}$ admits a perfect matching in $O(|\mathcal{T}(e_1, p_q)|^{5/2})$. As $\sum_{p_q \in \mathcal{P}(e)} |\mathcal{T}(e, p_q)|^2 \leq (\sum_{p_q \in \mathcal{P}(e)} |\mathcal{T}(e, p_q)|)^2$ and $\sum_{p_q \in \mathcal{P}(e)} |\mathcal{T}(e, p_q)|^{5/2} \leq (\sum_{p_q \in \mathcal{P}(e)} |\mathcal{T}(e, p_q)|)^{5/2}$, the result follows. □

One generally considers only properties with small arity: a property of arity > 10 is something very rare. In addition, if r can not be neglected, it has also to be considered when evaluating the size of the instance of the problem; not neglecting r is equivalent to multiplying both the size of the instance and the computation time by r. So we will consider r as a constant.

4.2 The complete algorithm

We now give the entire algorithm. Its output is a hierarchy $\mathcal{H}$, given as a set of nodes; each node corresponds to a class (i.e. a subset of E) and is given by a triple made of:

- Its representative: an entity of the class corresponding to the node.
- Its depth in the dendrogram (we have fixed the depth of the root to be -1, so, if the depth of a class C is k, then two elements of C are k-confusable, and elements of C are k-distinguishable from all the entities which are not in C).
- Its father, i.e. the smallest class strictly containing it, which is given by a couple (representative, depth). There may be many classes with the same representative (and many with the same depth), but a couple (representative, depth) is characteristic for exactly one class.

We call *k-class* a maximal set of entities which are pairwise k-confusable. In the dendrogram in Figure 4 one can see that $\{b_1, b_2, b_3, b_4, b_5\}$ is a 0-class. It is also a 1-class, a 2-class and a 3-class. In the resulting hierarchy $\mathcal{H}$, $\{b_1, \ldots b_5\}$ will only be considered as a 0-class, but during the progress of the algorithm, $\{b_1 \ldots b_5\}$ will also be considered as a 1-class, a 2-class and a 3-class. A k-class which is not a class of $\mathcal{H}$ corresponds to a $(k-1)$-class which has not been subdivided at step k. So every k-class is equal to a class of $\mathcal{H}$. The total number of k-classes can be much larger than the number of classes in $\mathcal{H}$. For instance, in a dendrogram like the one in Figure 8 with $n = 6$ leaves, there are $2n - 1 = 11$ "real" classes (the black circles), but the total number of k-classes is $n(n+1)/2 = 21$ (the white circles represent the k-classes

which are not classes of $\mathcal{H}$). We will see later that, although our algorithm considers all the k-classes, its complexity does not depend on the number of k-classes but is linear in the number of classes of $\mathcal{H}$. The main variables used by our algorithm are the

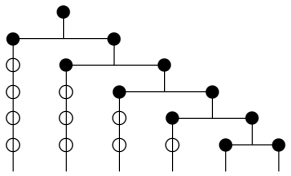

Figure 8: A dendrogram with n leaves, $2n-1$ classes and $n(n+1)/2$ k-classes

following:

- The integer k denotes the steps of the algorithm. At step k (each step goes from line 4 to 24), we look for k-distinguishability and construct all the k-classes.
- At the end of step k, $Rep[k]$ is the set of the representatives of all the k-classes. If e and e' are in $Rep[k]$, eD_ke', and if $e \notin Rep[k]$, $\exists e' \in Rep[k]$ such that eC_ke'.
- $Sets[k]$ is an array indexed by the set E of all entities. At the end of step k, for each entity e in $Rep[k]$, $Sets[k][e]$ is the k-class which admits e as a representative. If e is not a representative, $Sets[k][e]$ is not defined and is not used. $Sets$ does not need to be initialized.
- $OppSets[k]$ is an array indexed by the set E of all entities. If $e \in Sets[k][e']$, then $OppSets[k][e] = e'$. Equivalently, at the end of step k, for each entity e in E, $OppSets[k][e]$ is the representative of the k-class containing e. We will see that, at step k, $OppSets[k-1]$ is already computed; it is thus possible to test $(k-1)$-confusability in $O(1)$ ($eC_{k-1}e'$ if and only if $OppSets[k-1][e] = OppSets[k-1][e']$). So testing k-confusability at step k can be done in $O(m^{5/2})$ time and $O(m^2)$ space.
- $Depth$ is an array indexed by E. At step k, for every entity e which is representative of one or many classes of $\mathcal{H}$ of depth $\leq k$, $Depth[e]$ is the greatest depth of these classes. Equivalently, $Depth[e]$ is the depth of the smallest (yet computed) class of $\mathcal{H}$ containing e. If e is not the representative of a class, $Depth[e]$ is not defined. $Depth$ does not need to be initialized.
- $Continue$ indicates if, at step k, some classes are created. If it is not the case, the algorithm stops.

We now detail how Algorithm 1 works.

At line 2, the algorithm creates the root of the hierarchy: a unique class containing all the elements of E, whose representative is e_1 and depth -1.

At the beginning of each step, there is no k-class, so $Rep[k]$ is empty.

Algorithm 1: HIERARCHY COMPUTATION

Input: A set $E = \{e_1, \dots e_n\}$ of entities.
Output: A hierarchy $\mathcal{H}$ on E.

```
1  begin
2      H ← {(e_1, −1, ∅)} ; Rep[−1] ← e_1 ; Sets[−1][e_1] ← E ; Depth[e_1] ← −1 ;
3      k ← 0 ; Continue ← True ;
4      while Continue do
5          Rep[k] ← ∅ ;
6          foreach e ∈ Rep[k − 1] do
7              Temp ← Depth[e] ;
8              CurrentRep ← ∅ ;
9              foreach e′ ∈ Sets[k − 1][e] do
10                 New ← True ;
11                 foreach e″ ∈ CurrentRep do
12                     if e′ C_k e″ then
13                         Sets[k][e"] ← Sets[k][e"] ∪ {e′} ;
14                         OppSets[k][e′] ← e" ; New ← False ;
15                 if New then
16                     CurrentRep ← CurrentRep ∪ {e′} ;
17                     Sets[k][e′] ← {e′} ; OppSets[k][e′] ← e′ ;
18                     Depth[e′] ← k ; H ← H ∪ {(e′, k, (e, Temp))} ;
19             if CurrentRep = {e} then
20                 H ← H \ {(e, k, (e, Temp))} ; Depth[e] ← Temp ;
21             Rep[k] ← Rep[k] ∪ CurrentRep ;
22         Continue ← (|Rep[k]| ≠ |Rep[k − 1]|) ;
23         Delete (Rep[k − 1]) ; Delete (Sets[k − 1]) ; Delete (OppSets[k − 1]) ;
24         k ← k + 1 ;
25     foreach e ∈ Rep[k − 1] do
26         foreach e′ ∈ Sets[k − 1][e] do
27             H ← H ∪ {(e′, k, (e, Depth[e]))} ;
28 end
```

The k-classes are subclasses of $(k-1)$-classes. So the algorithm, on lines 6 and 9, will consider the $(k-1)$-classes independently (instead of considering all the elements of E "at the same time").

$CurrentRep$ is the set of the representatives of the (already known) subclasses of the $(k-1)$-class of e. For every entity e' of $Sets[k-1][e]$, the algorithm checks if e' is k-confusable with a representative of a class. If e' is k-confusable with a representative e'', the algorithm adds e' to the k-class of e'', i.e. to $Sets[k][e'']$. If e' is k-distinguishable with all the (already known) representative, then, at line 15, New is true and a new k-class is created with representative e' (lines 16—18). This is always

the case for the first element of $Sets[k-1][e]$.

Before the loop 9—18, $CurrentRep$ is empty. Once computed, it is added to the set $Rep[k]$ (line 21).

On lines 19 and 20 is treated the case when the $(k-1)$-class C of e is not partitioned at step k; in this case, C is also a k-class and $CurrentRep$ contains exactly one element, the representative of C. We can suppose that each time the algorithm goes through a set, it does it in the same order, so the representative of C, as a k-class, is also e. C must not appear as a k-class in $\mathcal{H}$. So the algorithm takes it out of $\mathcal{H}$ and gives the value it had before lines 9—18 to $Depth[e]$. This value has been loaded into $Temp$ at line 7.

On line 22, the algorithm checks if "something has been done at step k", i.e. if there are more k-classes than $(k-1)$-classes. If it is not the case, no other step is necessary.

After step k, the algorithm will not use what has been done at step $k-1$. So $Rep[k-1]$, $Sets[k-1]$ and $OppSets[k-1]$ are deleted in order to minimize the memory used by the algorithm.

On lines 25—27, each entity is assigned to the smallest class containing it; the hierarchy $\mathcal{H}$ is then complete.

Property 2. *Algorithm 1 runs in $O(n+m^2)$ space and $O(n \cdot p \cdot m^{5/2})$ time in worst case, where n is the number of entities, p is the number of classes in $\mathcal{H}$ ($p < 2n$) and m is the maximum number of properties an entity can have.*

Proof. The algorithm uses 9 sets[5] of elements ($Rep[k-1]$, $Rep[k]$, $Sets[k-1]$, $Sets[k]$, $OppSets[k-1]$, $OppSets[k]$, $Depth$, $CurrentRep$ and $\mathcal{H}$) which are all of size $O(n)$. It also has to build graphs with $2m$ vertices, but only one at a time. So it runs in $O(n+m^2)$ space.

The time complexity of Algorithm 1 depends on the number of tests "**if** $e'C_ke''$" (line 12), where e' is an entity and e'' a representative of a k-class (we recall that each of these tests takes $O(m^{5/2})$ time; in addition, what the algorithm does in each case takes a constant amount of time). We prove that, for each entity e, the number of these tests is $O(p)$.

Let e be a fixed entity. At Step k, e is only "compared" with subclasses of the $(k-1)$-class to which it belongs (lines 6, 9 and 11).

We first give an upper bound of the number of times e is compared with a class it does not belong to. For any k, all the k-classes to which e is compared while not belonging to them are disjoint. Moreover, since all these k-classes are subclasses of the $(k-1)$-class of e, the $(k-1)$-classes not containing e and to which e is compared are disjoint from the compared k-classes not containing e. All are subclasses of the $(k-2)$-class of e, so the compared $(k-2)$-subclasses not containing e are disjoint from the compared k-classes and $(k-1)$-classes. Thus all the classes not containing e to which e is compared are disjoint. Since every class is equal to a class of $\mathcal{H}$, e is compared with at most $p-1$ classes not containing it.

We now give the number of comparisons between e and a class to which it belongs. For every $k \leq \mathcal{K}$, e is in exactly one k-class. At each step of the algorithm (except the

[5] Actually, most of them ($Sets$, $OppSets$, $Depth$) are associative arrays.

last one), at least one class is divided and so two or more classes of $\mathcal{H}$ are "created". So $\mathcal{K} \leq p$ and e is compared with at most p classes containing it.

Each entity is compared with $O(p)$ classes, so there are $O(n \cdot p)$ tests of line 12 and Algorithm 1 runs in $O(n \cdot p \cdot m^{5/2})$ time (lines 25—27 take $O(n)$ time and do not change the overall complexity). □

The bound $O(np)$ for the number of comparisons can be reached, for instance if all the entities are 0-distinguishable. On the contrary, if the hierarchy is a balanced binary tree, at each step, every entity is compared with at most two classes; so the total number of comparisons is $O(n \log p)$.

4.3 Tests on large sets of entities

Today, one often has to treat huge data sets, i.e. millions of entities. So, although our algorithm has a polynomial time and space complexity, it is interesting to test it on large data sets and so, we have tested our algorithm on data sets up to 10 million entities. We do not pretend that our implementation is the best possible (the program is written in Python, which is an interpreted language; more precisely, these tests ran in Python 2.7 on a 16 Intel Xeon X5560 at 2.8 GHz computer with 48 Gb RAM. The computer has 16 processors, but only one was used for these tests); these tests are just "feasibility tests".

The construction of the entity sets is similar to what was done for the last example of section 3: we have generated large data sets with $\{0,1\}$-arrays of size 1 million, 2 millions, ...16 millions . More precisely, the arrays are random arrays (each cell has probability 0.5 to be one). The relations are $\mathsf{equals\_1}(e)$ and $\mathsf{is\_near}(e_1, e_2)$ (two entities $T[i]$ and $T[j]$ are "near" if $0 < |i-j| < 3$). Since relation $\mathsf{is\_near}$ is symmetric, we only consider property $\mathsf{is\_near}_1$. For each entity e, there are four tuples that match $\mathsf{is\_near}_1$ with e. For such data sets, all the entities are distinguishable, so $p \approx n$. This is the worst case for our algorithm. We obtained the results of Figure 9.

- The amount of memory used is approximatively 2.5 GigaBytes per million entities.
- The computation time grows slower than np (the time needed to treat 5 million entities is smaller than 25 times the time needed to treat one million), which is the theoretical complexity. Actually, the computation time also depends on the depth of the hierarchy. For arrays used in this test, this depth is < 15 (two entities are 15-confusable if two balls of radius 30 are identical; there are 2^{61} possible such balls).

So, we can see that our algorithm is able to treat large data sets. The classification method that we propose in this paper is thus applicable for "real world" applications.

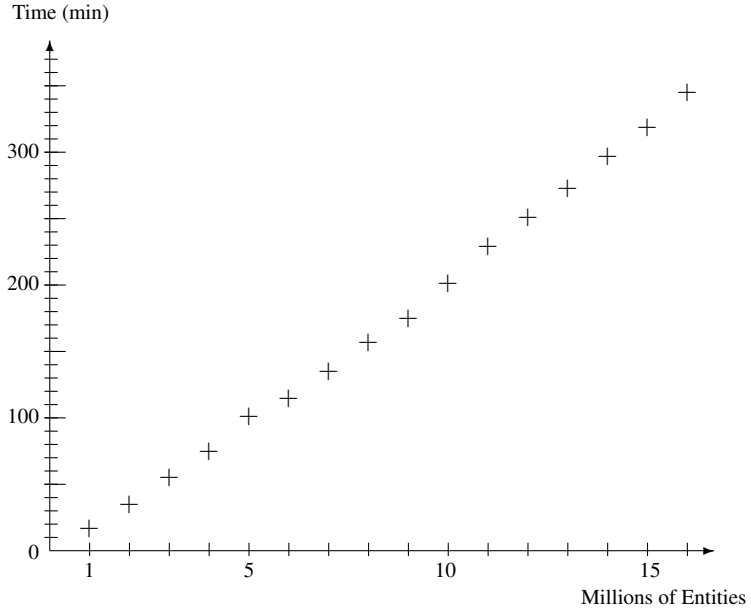

Figure 9: Time needed to treat large entity sets

5 Test on Fisher's iris data set

Our method is, basically, a discrimination method: we detect differences between entities which are, a priori, similar (since they are 0-confusable). When we apply it on a randomly generated entity set as in Section 4.3, we get a final partition made of singletons. On the contrary, when applied on real data, our method may yield a partition made (at least partly) of "large" sets. In this case, we can say that there is a "structure" inside this data set.

We have tested our method on the Fisher iris data set (from Fisher 1936). This data set is made of 150 iris flowers: 50 iris *i.setosa*, 50 iris *i.versicolor* and 50 iris *i.virginica*. Each flower is given with four parameters: the length and width of the petal and the sepal. It is very easy to recognize the *i.setosa* among the iris: they all have a petal width ≤ 0.3 while all the others have a petal width ≥ 1. There is not such an easy separation between the *i.versicolor* and the *i.virginica*. In Figure 10 is shown the result of a Principal Component Analysis of this data set.

In order to apply our method to this data set, we first have to define relations between iris. We do that in the following way:

1. We normalize the four parameters: the average value is 0 and the standard deviation is 1.

2. Given a threshold h, we say that two iris are *close* if they have two parameters which differ by less than h; so we have one binary relation: *is_close* $(iris_1, iris_2)$

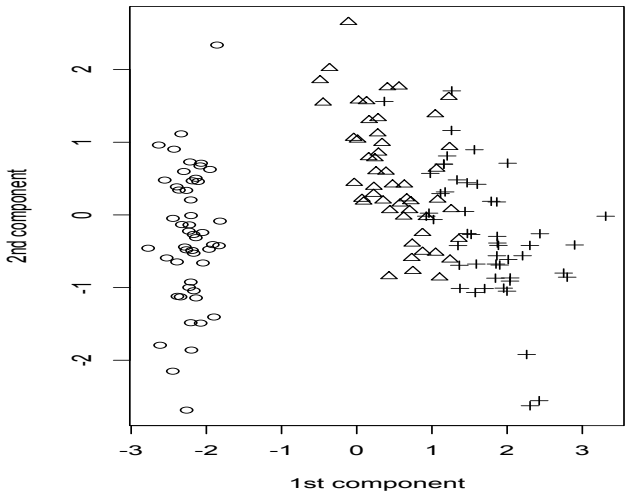

Figure 10: The result of a PCA on Fisher's iris; The *i.setosa* are marked with a circle, the *i.versicolor* with a triangle and the *i.virginica* with a cross.

and one property is_close_1 (the properties is_close_1 and is_close_2 are equivalent). Actually, we build a graph with 150 vertices (the iris) and a number of edges that will depend on h.

By applying our method, we get the following results:

- With $h = 1$, no information or structure can be detected among the iris: the final partition of the iris set is made of 130 small classes (the greatest is of size 5).
- With $h = 1/2$, we "recognize" the *i.setosa*: the final partition is made of one class with 39 elements (all *i.setosa*) one class with 5 elements (all *i.setosa*), two pairs and singletons.
- With $h = 2$, we recognize the *i.versicolor*: the final partition is made of one class with 51 elements (47 *i.versicolor* and 4 *i.virginica*), one with 6 elements (all *i.setosa*), 8 pairs and singletons.

These results can be interpreted in the following way:

- With threshold $h = 1/2$, we link each iris with its close neighbors (the standard deviation for each parameter is 1). The *i.setosa* form a dense class, which is well separated from the other iris. So each *i.setosa* has many neighbors among the *i.setosa* and none among the other classes. Actually, the "large" class is made of iris which have 49 neighbors, 38 of them having 49 neighbors. The other *i.setosa* have a little less than 49 neighbors (among which 38 have 49 neighbors).

- The threshold $h = 2$ is approximatively the radius of the set. We recognize the *i.versicolor* because they are in central position among the iris. The large class is made of the iris which are neighbors with all the other iris.

6 Related works and perspectives

Some previous works deal with n-ary functions on entities (see Agarwal et al. 2005; Diatta 2004; Préa 1994), but their goal is to define a measure on sets which is based on many points, and then to apply metric methods. Basically, our method is not metric: we do not use a n-metric to classify, but we build an ultrametric from boolean (n-ary) functions.

Although the tests have shown that it is possible to treat huge data sets, treating efficiently such data sets would require some improvements; a promising one would be parallel programming. Parallelisation is possible, since at each step, the same treatment is independently applied on all entities (more precisely, at each couple (entity, representative) of its class).

A characteristic of the ultrametric we build is its dependence on the entire set of entities, or, equivalently, the instability of the obtained classification, which is illustrated in Figures 4, 5 and 6. If we consider a social network with the relation *is_friend_with*, such an instability is critical: people often add, and sometimes remove, *friends*. In addition, for such a network, distinguishing between someone who has 511 friends, and another one who has "only" 497 is not pertinent. Actually, this precision is the cause of the instability. This precision is also better for discriminate/separate than for classify/put together. One way to correct these two defaults would be to use an imprecise measure, like fuzzy sets (see Zadeh 1965) to estimate the number of tuples matching a property with an entity. For instance, with less precision when analysing the Fisher iris, with threshold $h = 1/2$, we can characterize the *i.setosa* as the iris which have around 49 neighbors, each of these neighbors also having around 49 neighbors, and with threshold $h = 2$, we can characterize the *i.versicolor* (plus around 8 *i.virginica*) as the iris having nearly 149 neighbors.

References

[1] AGARWAL, S., LIM, J., ZELHIK-MANOR, L., PERONA, P., KRIEGMAN, D. and BELONGIE, S. (2005)," Beyond Pairwise Clustering", in *Proceedings of the IEEE Computer Conference on Computer Vision and Pattern Recognition*, Piscataway, IEEE Computer Society, vol. 2, pp 838-845.

[2] ALT, H., BLUM, N., MEHLTON, K. and PAUL, M. (2001), "Computing a maximal cardinality matching in a bipartite graph in time $O(n^{1.5}\sqrt{m/\log n})$", *Information Processing Letters*, 37 (4): 237-240.

[3] BARTHÉLEMY, J.-P. and GUÉNOCHE, A. (1991), *Trees and Proximity Representations*, Wiley.

[4] BUFFON, G.-L. LECLERC DE (1749), *Histoire Naturelle, Générale et Particulière, avec la Description du Cabinet du Roy*, available at `www.buffon.cnrs.fr`.

[5] CROITORU, M. and DEEMTER, K. VAN (2007), "A Conceptual Graph Approach to the Generation of Referring Expressions", in *Proceedings of the International Joint Conference on Artificial Intelligence*, Hyderabad, Morgan Kaufman, pp 2456-2461.

[6] DALE, R. (1989), "Cooking up Referring Expressions", in *Proceedings of the Twenty-Seventh Annual Meeting of the Association for Computational Linguistics*, Vancouver, Association for Computational Linguistics, pp. 68-75.

[7] DEEMTER, K. VAN (2002), "Generating Referring Expressions: Boolean Extensions of the Incremental Algorithm", *Computational Linguistics*, 28(1):37-52.

[8] DEEMTER, K. VAN and KRAHMER, E. (2006), "Graphs and Booleans: On the generation of referring expressions", in *Computing Meaning : Volume 3 (Studies in Linguistics and Philosophy)*, H. Bunt and R. Muskens Eds., Springer, pp 397-422.

[9] DIATTA, J. (2004), "A Relation Between the Theory of Formal Concepts and Multiway Clustering", *Pattern Recognition Letters*, 25:1183-1189.

[10] FISHER, R.A. (1936), "The use of multiple measurements in taxonomic problems", *Annals of Eugenics*, 7 (2): 179–188.

[11] GARDENT, C. (2002), "Generating Minimal Definite Descriptions", in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, Association for Computational Linguistics, pp 96-103.

[12] GORDON, A.D. (1999), *Classification*, Chapman & Hall.

[13] HOPCROFT, J.E. and KARP, R.M. (1973), "An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs", *SIAM Journal on Computing*, 2(4):225-231.

[14] JARDINE, N. and SIBSON, R. (1971), *Mathematical Taxonomy*, Wiley.

[15] KHAN, I.H., DEEMTER, K. VAN, RITCHIE, G., GATT A. and CLELAND, A.A. (2009), "A Hearer-Oriented Evaluation of Referring Expression Generation", in *Proceedings of the 12th European Workshop on Natural Language Generation*, Athens, Association for Computational Linguistics, pp. 98-101.

[16] KRAHMER, E., ERK, S. VAN and VERLEG, A. (2003), "Graph-based Generation of Referring Expressions", *Computational Linguistics*, 29(1):53-72.

[17] LINNÉ, C. VON (1735), *Systema Naturæ*, available at `gallica.bnf.fr/ark:/12148/bpt6k99004c`.

[18] MITCHELL, M., (2009), "Class-Based Ordering of Prenominal Modifiers", in *Proceedings of the 12th European Workshop on Natural Language Generation*, Athens, Association for Computational Linguistics, pp 50-57.

[19] OSBORNE, M.J. (2009), *An Introduction to Game Theory*, Oxford University Press.

[20] PRÉA, P. (1994), "A Generalization of the Diameter Criterion for Clustering", in *New Approaches in Classification and Data Analysis*, E. Diday, Y. Lechevallier, M. Schader, P. Bertrand and B. Burtschy Eds., Springer-Verlag, pp 257-262.

[21] ROLBERT, M. and PRÉA, P. (2009), "Distinguishable Entities: Definitions and Properties", *Proceedings of the 12th European Workshop on Natural Language Generation*, Athens, Association for Computational Linguistics, pp 34-41.

[22] ZADEH, L.A. (1965), "Fuzzy Sets", *Information and Control*, 8:338-353.