Analogical Dissimilarity: definition, algorithms and first experiments in machine learning

Laurent Miclet, Arnaud Delhay

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Analogical Dissimilarity: definition, algorithms and first experiments in machine learning

Laurent Miclet, Arnaud Delhay

N° 5694

Septembre 2005

Thème COG

ISSN 0249-6399 ISRN INRIA/RR--5694--FR+ENG

Analogical Dissimilarity: definition, algorithms and first experiments in machine learning

Laurent Miclet, Arnaud Delhay*

Thème COG — Systèmes cognitifs
Projet CORDIAL

Rapport de recherche n° 5694 — Septembre 2005 — 60 pages

Abstract: This paper defines the notion of analogical dissimilarity between four objects, with a special focus on dissimilarity between objects structured as sequences. Firstly, it studies the case where the four objects have a null analogical dissimilarity, i.e. are in an analogical relation. Secondly, when one of these objects is unknown, it gives algorithms to compute it. In particular, it studies a new formulation of solving analogical equations on sequences, based on the edit distance between strings. Thirdly, it tackles the problem of defining analogical dissimilarity, which is a measure of how close four objects are from being in analogical relation. To finish, it gives learning algorithms, i.e. methods to find the triple of objects in a learning sample which has the least analogical dissimilarity with a given object.

Key-words: Analogy, Sequences, Machine Learning, Analogical Dissimilarity, Learning by analogy

* {Laurent.Miclet,Arnaud.Delhay}@univ-rennes1.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex (France)
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Dissemblance analogique : définition, algorithmes et premières expériences en apprentissage

Résumé : Ce papier définit la notion de dissemblance analogique entre quatre objets, et se concentre plus particulièrement sur la dissemblance entre objets structurés en séquences. Nous étudions tout d'abord le cas où les quatre objets ont une dissemblance analogique nulle, c'est-à-dire où il sont en relation d'analogie. Ensuite, quand un des quatre objets est inconnu, nous donnons un algorithme pour le calculer. En particulier, nous étudions une nouvelle façon de formuler la résolution des équations d'analogie entre séquences, basée sur la distance d'édition entre chaînes. Puis, nous abordons le problème de la définition de la dissemblance analogique qui est une mesure de combien quatre objets sont proches d'être en relation d'analogie. Pour terminer, nous donnons des algorithmes d'apprentissage, c'est à dire des méthodes pour trouver le triplet d'objets dans un ensemble d'apprentissage qui a la plus faible dissemblance analogique avec un objet donné.

Mots-clés : Analogie, Séquences, Apprentissage, Dissemblance analogique, Apprentissage par analogie

Contents

1 Introduction.

1.1 Reasoning and learning by analogy.

Analogy is a way of reasoning which has been studied throughout the history of philosophy and has been widely used in Artificial Intelligence ([28]) and Linguistics. Lepage ([19]) has given an extensive description of the history of this concept and its applications in science and linguistics.

1.1.1 Analogical relation between four objects.

An *analogy* or *analogical relation* between four objects A, B, C and D in the same universe is usually expressed as follows : "*A is to B as C is to D*". Depending on what are the objects, analogies can have very different meanings. For example, natural language analogies could be: "`a crow` *is to* `a raven` *as* `a merlin` *is to* `a peregrine`" or "`vinegar` *is to* `wine` *as* `a sloe` *is to* `a cherry`". These analogies are based on the *semantics* of the words. By contrast, in the formal universe of sequences, analogies such as "`abcd` *is to* `abc` *as* `abbd` *is to* `abb`" or "`g` *is to* `gt` *as* `gg` *is to* `ggt`" are *morphological*.

Whether morphological or not, the examples above show the intrinsic ambiguity in defining an analogy. We could as well accept, for other good reasons: "`g` *is to* `gt` *as* `gg` *is to* `ggtt`" or "`vinegar` *is to* `wine` *as* `cheese` *is to* `milk`". Obviously, such ambiguities are inherent in semantic analogies, since they are related to the meaning of words (the concepts are expressed through natural language). Hence, it seems important, as a first step, to focus on formal morphological properties. Moreover, resolving syntactic analogies in sequences is an operational problem in several fields of linguistics, such as morphology and syntax, and provides a basis to learning and data mining by analogy in the universe of sequences.

Several formal definitions of the analogical relations "is to" and "as" will be defined in this article, but the basic idea can be grasped through examples on sequences of letters: the analogy "`began` *is to* `begun` *as* `ran` *is to* `run`" holds true, from the linguistic point of view as well as from a pure "sequence of letters" or "morphological" point of view. From this second point of view only, "`xan` *is to* `xun` *as* `yzan` *is to* `yzun`" is also true, while "`began` *is to* `begun` *as* `move` *is to* `moved`" is true only from the first point of view.

In this paper, we will firstly consider analogies in sets of letters and secondly how they may be transferred to morphological analogies between sequences of letters.

1.1.2 Solving analogical equations.

When one of the four elements is unknown, an analogical relation turns into an equation. For instance, on sequences of letters: "`wolf` *is to* `leaf` *as* `wolves` *is to* x". Resolving this equation consists in computing the (possibly empty) set of sequences x which satisfy the analogy. The sequence `leaves` is both an obvious linguistic and morphological solution. We shall see that, however, it is not straightforward to design an algorithm able to solve this kind of equation.

Solving analogical equations on sequences is useful for linguistic analysis tasks and has been applied (with empirical resolution techniques, or in simple cases) mainly to lexical analysis tasks. For example, Yvon([34], [35]) presents an analogical approach to the grapheme-to-phoneme conversion, for text-to-speech synthesis purposes. More generally, the resolution of analogical equations can also be seen as a basic component of *learning by analogy* systems, which are part of the *lazy learning* techniques [9].

1.1.3 Learning by analogy.

Let $\mathcal{S}$ be a set of training examples $\mathcal{S} = \{(x, u(x))\}$, where x is the description of an example (x may be a sequence or a vector in $\mathbb{R}^d$, for instance) and $u(x)$ its label in a finite set. Given the description y of a new pattern, we would like to assign to y a label $u(y)$, based only from the knowledge of $\mathcal{S}$. This is the problem of learning a classification rule from examples ([25]), which consists in finding the value of u at point y. The nearest neighbor method, which is the most popular lazy learning technique, simply finds in $\mathcal{S}$ the description $x^\star$ which minimizes some distance to y and hypothesizes $u(x^\star)$, the label of $x^\star$, for the label of y.

Moving one step further, analogical learning searches in $\mathcal{S}$ for a triple $(x^\star, z^\star, t^\star)$ such that "$x^\star$ *is to* $z^\star$ *as* $t^\star$ *is to* $y^\star$" and predicts for y the label $\hat{u}(y)$ which is a solution of the equation "$u(x^\star)$ *is to* $u(z^\star)$ *as* $u(t^\star)$ *is to* $\hat{u}(y)$". If more than one triple is found, a voting procedure can be used. Such a learning technique is based on the resolution of analogical equations. [26] discusses at length the relevance of such a learning procedure for various linguistic analysis tasks. It is important to notice that y and $u(y)$ are in different domains: for example, in the simple case of learning a classification rule, y may be a sequence and u is merely a class label.

A further step in learning by analogy is to find in $\mathcal{S}$ for a triple $(x^\star, z^\star, t^\star)$ such that "$x^\star$ *is to* $z^\star$ *as* $t^\star$ *is to* $y^\star$" holds *almost* true, or, when a closeness measure is defined, the triple which is the closest to y in term of analogical relation. We study in this article how to quantify this measure, in order to provide a more flexible method of learning by analogy.

1.2 Situation of the paper and bibliography.

This paper is related with several domains of artificial intelligence. Obviously, the first one is that of reasoning by analogy. Much work has been done on this subject from a cognitive science point of view, which had led to computational models of reasoning by analogy (see the book [13] and, for example, the classical paper [12]). Usually, these works use the notion of *transfer*, which is not within the scope of this article. It means that some knowledge on solving a problem in a domain is transported to another domain. Since we work on four objects that are in the same space X, we implicitly ignore the notion of transfer between *different* domains. Technically speaking, this restriction allows us to use an axiom called "exchange of the means" to define what is an analogy (see Definition 2.1). For example, if we work on strings, the four strings have to be written on the same alphabet. However, we share with these works is the following idea: there may be a similar relation between two couples of structured objects even if the objects are apparently quite different. We are interested in giving a formal and algorithmic definition of what such a relation can be.

Our work aims also at defining some supervised machine learning process ([25], [7]), in the spirit of *lazy* learning ([2]). It means that we do not seek to extract a model from the learning data, but merely conclude what is the class, or more generally the supervision, of a new object by inspecting (a part of) the learning data. Usually, lazy learning, like the k-nearest neighbors technique, makes use of unstructured objects, such as vectors. Since distance measures can be also defined on strings, trees and even graphs, this technique has also been used on structured objects, in the framework of structural pattern recognition (see for example [5, 4, 3]). We extend here the search of the nearest neighbor in the learning set to that of the best triple (when combined with the new object, it is the closest to make an analogy). This requires to define what is an analogy on structured objects, like sequences, but also to give a definition of how far a four-uple of objects is from being in analogy (that we call analogical dissimilarity).

Learning by analogy on sequences has already being studied, in a more restricted manner, on linguistic data ([34, 36, 16, 15], etc.). Reasoning and learning by analogy has proven useful in tasks like grapheme to phoneme conversion, morphology and even translation. Sequences of letters and/or of phonemes are a natural application to our work, but we are also interested in the future on other type of data, structured as sequences or trees, like prosodic representations for speech synthesis, biochemical sequences, etc.

Analogical relations between four structured objects of the same universe, mainly strings, have been studied with a mathematical and algorithmic approach, like ours, by Mitchell and Hofstadter ([24, 14]), Dastani et al. ([10]), Schmid et al. ([31]), Yvon et al. ([38]). To the best of our knowledge, the use of the edit distance as a method of comparison between sequences is original in the framework of analogy, as is our formal definition of what can be a analogical dissimilarity between four objects, *a fortiori* between sequences.

To connect with another field of A.I., let us quote A. Aamodt and E. Plaza ([1]) about the use of the term "analogy" in Case-Based Reasoning (CBR): "Analogy-based reasoning: This term is sometimes used, as a synonym to case-based reasoning, to describe the typical case-based approach. However, it is also often used to characterize methods that solve new

problems based on past cases from a different domain, while typical case-based methods focus on indexing and matching strategies for single-domain cases." CBR is close to reasoning by analogy, since it focusses on the transfer between different domains. Again, we are interested here in formal analogies between four objects of the same domain.

1.3 Organization of the paper

This paper is organized in eight sections. After this introduction, we present in section 2 the general principles which govern the definition of an analogy between four objects in the same universe. This is applied firstly to three sets of objects: objects defined by binary features, finite cyclic groups and vector space $\mathbb{R}^n$. Section 3 describes what is an analogy between four sequences, firstly according to a definition by Lepage, and Yvon and Stroppa, secondly giving an original extension to sequences of letters on which analogies are defined, and using the edit distance for the "is to" relation.

Sections 4 and 5 study the resolution of analogical equations, in sets and in sequences. Concerning sequences, we recall the algebraic method by Yvon and we present a definition and two different algorithms to compute the set of solutions with a more general concept of analogy. We show that both algorithms (called SEQUANA1 and SEQUANA2) produce the set of all the solutions according to our definition.

Sections 6 and 7 introduce the new concept of *analogical dissimilarity* (AD) between four objects, by measuring in some way how much these objects are in analogy. In particular, it must be equivalent to say that four objects are in analogy and that their analogical dissimilarity is null. We define this concept in the three sorts of sets that we use as alphabets and we extend it to sequences. We give an algorithm to compute the value of AD between four sequences, which is based on the same idea than SEQUANA2. We also show that an algorithm based on SEQUANA1 cannot produce the same result. To finish, we give a few experiments to measure how AD can cope with noisy data.

Section 8 begins to explore the use of the concept of analogical dissimilarity in supervised machine learning. We extend the AESA algorithm of fast search of the nearest neighbor to that of the fast search of the *best* analogical triplet in the learning set. This algorithm, called FADANA, is tested on artificial data to measure its efficiency.

The last section is a conclusion and presents work to be done, particularly in presenting some possible real world application of learning by analogy in the universe of sequences.

2 Analogy in finite sets.

2.1 The axioms of analogy.

There is no general definition of an analogical relation "`A` *is to* `B` *as* `C` *is to* `D`" between four objects in a set X, the "is to" and the "as" relations depending on the nature of X. However, according to the usual meaning of the word "analogy" in philosophy and linguistics, three basic axioms are generally required ([19]):

Definition 2.1 (Analogy.) *An analogy on a set X is a relation on X^4, i.e. a subset $\mathcal{A} \subset X^4$. When $(A, B, C, D) \in \mathcal{A}$, the four elements A, B, C and D are said* to be in analogy*, and we write: "the analogical relation $A : B :: C : D$ holds true", or simply $A : B :: C : D$, which reads "A is to B as C is to D". For every four-uple in analogy, we have the following properties:*

Symmetry of the "as" relation*:* $C : D :: A : B$

Exchange of the means*:* $A : C :: B : D$

*A third axiom (*determinism*) requires that one of the two following implication holds true (the other being a consequence):*

$$\begin{array}{lcl} A : A :: B : X & \Rightarrow & X = B \\ A : B :: A : X & \Rightarrow & X = B \end{array}$$

According these axioms, five other formulations are proven to be equivalent to $A : B :: C : D$:

$$\begin{array}{ccc} B : A :: D : C & D : B :: C : A & C : A :: D : B \\ D : C :: B : A & \text{and} & B : D :: A : C \end{array}$$

A consequence of the first two axioms is that there are only three different possible analogies between four objects, with the canonical forms:

$$A : B :: C : D \qquad A : C :: D : B \qquad A : D :: B : C$$

2.2 Definition of a distance coherent with analogy.

X is a *metric set* if there exists a distance δ on X, according to the classical following definition.

Definition 2.2 (Distance on a set X.) *A distance δ on a set X is a mapping of $X \times X$ on $\mathbb{R}$, with the following properties:*

Reflexivity. $\forall x \in X, \delta(x, x) = 0$

Strict positiveness. $\forall x, y \in X, x \neq y \Rightarrow \delta(x, y) > 0$

Symmetry. $\forall x, y \in X, \delta(x, y) = \delta(y, x)$

Triangle inequality. $\forall x, y, z \in X, \delta(x, y) \leq \delta(x, z) + \delta(z, y)$

If there exists an analogy on a metric space X, a relation between the distance and the analogy on X can be defined. Its usefulness will appear later in this paper, especially when defining analogies on sequences (section 3 and 5) and analogical dissimilarities (sections 6 and 7).

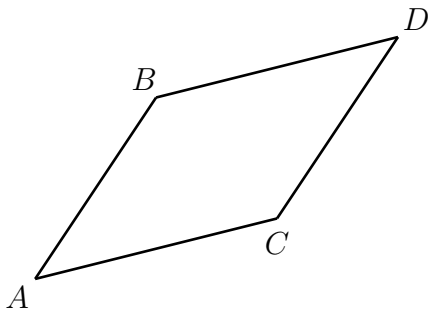

Figure 1: Analogy and coherent distance in $\mathbb{R}^n$. When the analogy $A : B :: C : D$ stands true, the four elements form a parallelogram and the euclidian distance δ_2 has the properties : $\delta_2(u, v) = \delta_2(w, x)$ and $\delta_2(u, w) = \delta_2(v, x)$

Definition 2.3 (Distance coherent with analogy.) *A distance δ is said* coherent *with analogy if for every four-uple A, B, C and D in the analogical relation:*

$$A : B :: C : D$$

δ has the properties :

$$\delta(A, B) = \delta(C, D) \quad and \quad \delta(A, C) = \delta(B, D)$$

It is clearly the case, for example, if $X = \mathbb{R}^n$, four elements being in analogy if they form a parallelogram (see Figure 1), and if δ is the euclidian distance. This example will be developed at section 2.4.

2.3 Analogy in sets.

2.3.1 Defining finite sets by binary features.

Let X be a finite set or *alphabet*, composed of elements that we will call *objects*. We assume that there exists a set $\mathcal{F}$, with cardinal n, of binary features such that every object $x \in X$ can be defined by a binary vector $(f_1(x), \ldots, f_n(x))^\top$. For every x and every $i \in [1, n]$, $f_i(x) = 1$ (resp. $f_i(x) = 0$) means that the binary feature takes the value $TRUE$ or 1 (resp. $FALSE$ or 0) on the object x. We call such a set X *a finite set defined by binary features.*

Equivalently, an object $x \in X$ can be seen as a subset of $\mathcal{F}$, composed of the elements f_i such that $f_i(x) = 1$. Therefore, studying what is analogy between four objects in an alphabet defined by binary features is equivalent to studying what is analogy between four sets.

A first definition. When the "as" relation is the equality between sets, Lepage has given a definition of an analogical relation between sets coherent with the axioms.

Definition 2.4 (Analogy between sets.) *Four sets A, B, C et D are in analogy* $A : B :: C : D$ *if and only if A can be transformed into B and C into D by adding and subtracting the same elements to A and C.*

This is the case, for example, of the four sets : $A = \{t_1, t_2, t_3, t_4, \}$, $B = \{t_1, t_2, t_3, t_5\}$ and $C = \{t_1, t_4, t_6, t_7\}$, $D = \{t_1, t_5, t_6, t_7, \}$, where t_4 has been taken off from, and t_5 has been added to A et C, giving B and D.

With this definition, Lepage ([18]) has shown a double necessary condition of inclusion between four sets to be in analogical relation:

$$A \subset B \cup C \quad \text{and} \quad A \supset B \cap C \tag{2.1}$$

In section 4.2 we will see how, under this condition, a unique solution D can be given to the equation $A : B :: C : x$, with respect to the axioms of analogy:

$$x = ((B \cup C) \backslash A) \cup (B \cap C)$$

A second equivalent definition. Stroppa and Yvon have given another definition of the analogy between four sets, which proves to be equivalent to that of Lepage ([32]).

Definition 2.5 (Analogy between sets.) *Four sets A, B, C et D are in analogy* $A : B :: C : D$ *if and only if there exists four sets X, Y, Z et T such that :*

$$\begin{aligned} A &= X \cup Y \\ B &= X \cup Z \\ C &= T \cup Y \\ D &= T \cup Z \end{aligned}$$

We have given in [23] the complete proof that the two definitions are equivalent. The sketch is straightforward: the inclusion conditions of equation 2.1 imply that, among the 16 disjoint sets created by the intersection of A, B, C and D, only 5 are non empty. They can be combined by union either according to the first definition or to the second.

The transitivity of analogy in sets. In sets, the analogy has the property of transitivity:

Property 2.1 (Transitivity of analogy in sets.) *Let A, B, C, D, E and F be six sets. Then the following implication holds:*

$(\ A : B :: C : D \quad \textit{and} \quad C : D :: E : F \ ) \Rightarrow \quad A : B :: E : F$

A distance in sets defined by binary features coherent with analogy. Let X be a set defined by binary features. We can see an element A of X either as the set of features that are $TRUE$ on A or as a binary vector of size n, where n is the total number of features (the cardinal of the set $\mathcal{F}$). In the first case, we define the distance $\delta(A, B)$ between two elements A and B of X as the cardinal of symmetrical difference between the two sets. In the second case, $\delta(A, B)$ is the Hamming distance between the two binary vectors. Obviously, the two definitions are equivalent. We have the following property :

Property 2.2 (Coherence of the symmetrical difference.) *Let $\delta(A, B)$ be the cardinal of the symmetrical difference between the two sets A and B. Then δ is coherent with the analogy on sets.*

The proof is straightforward, from the definition of analogy on X.

2.3.2 Defining finite alphabets as cyclic groups.

In this section, we define an analogy and a coherent distance on finite cyclic groups. We start from a finite set with an inner operator and we examine what properties are requested in connexion with the analogy. This construction is sufficient to eventually insure that every analogical equation has a unique solution (this point is developed in section 4.3).

Let $(X, \oplus)$ be a set with an inner operator and an analogy. Let a, b, c and d be four elements of X. We connect the operator $\oplus$ to the analogy on X by requiring the following property :

$$\big(a \oplus d = b \oplus c\big) \Leftrightarrow \big(a : b :: c : d\big)$$

Properties of the operator $\oplus$ according to the analogy. We have given in definition 2.1 the axioms of analogy as described by Lepage. From each axiom, we deduce an algebraic property for the operator $\oplus$.

Symmetry.

$a : b :: c : d \Rightarrow c : d :: a : b$, that is: $a \oplus d = b \oplus c \Rightarrow c \oplus b = d \oplus a$

Exchange of the means.

$a : b :: c : d \Rightarrow a : c :: b : d$ that is $a \oplus d = b \oplus c \Rightarrow a \oplus d = c \oplus b$

From this, we conclude that the operator $\oplus$ must be commutative, since $c \oplus b = b \oplus c$ if $a : b :: c : d$.

Determinism.

$a : a :: c : x \Rightarrow x = c$ and $a : b :: a : x \Rightarrow x = b$. It can be expressed with $\oplus$ by : $a \oplus x = a \oplus c \Rightarrow x = c$ and $a \oplus x = b \oplus a \Rightarrow x = b$

The first equation expresses the property of *left regularity*. Because of the commutativity, we can state that $\oplus$ must be *regular*.

Uniqueness of the solution. We anticipate here on section 4.3 to go along with our construction. To *solve an analogical equation* $a : b :: c : x$ is to find every element x which verifies this relation. In the case of finite alphabets with an operator $\oplus$ defined as above, we can manage so that every analogical equation has a unique solution. For this purpose,

we need to consider a special element of X, namely $\backsim$, to be the neutral element of $(X, \oplus)$, i.e. $a\oplus \backsim = \backsim \oplus a$. In this case, the analogical equation $a :\backsim :: \backsim : x$, which can also be expressed as : $a \oplus x = \backsim \oplus \backsim = \backsim$ would have a unique solution if every element a in X has a unique symmetric. Assuming that X is a *group* [27] is sufficient to get this properties. Moreover, this group is *abelian* since $\oplus$ is commutative. A sufficient manner to give the final construction of this group is to take it as an additive cyclic group. The table of the operator $\oplus$ is given on a group of size 7 in Figure 2.

$\oplus$	$\backsim$	a	b	$\boxed{c}$	d	e	$\boxed{f}$
$\backsim$	$\backsim$	a	b	c	d	e	f
$\boxed{a}$	a	b	c	$\boxed{d}$	e	f	$\backsim$
b	b	c	d	e	f	$\backsim$	a
c	c	d	e	f	$\backsim$	a	b
d	d	e	f	$\backsim$	a	b	c
$\boxed{e}$	e	f	$\backsim$	a	b	c	$\boxed{d}$
f	f	$\backsim$	a	b	c	d	e

Figure 2: A table for an analogical operator on an alphabet of 6 elements plus $\backsim$, seen as the additive cyclic group $\mathcal{G}_7$.

A distance in alphabets defined as finite groups coherent with analogy. We want here to build a distance δ on the alphabet which is coherent with the analogy. That is, for a quadruple (x, y, z, t) in analogy, we want to have the equality : $\big((x : y :: z : t) \Leftrightarrow (x \oplus t = y \oplus z)\big) \Rightarrow \big(\delta(x, y) = \delta(z, t)\big)$.

For example, we know from Figure 2, considering the element c, that δ is such that:

$$\begin{aligned}
c = c\oplus \backsim = b \oplus a &\Rightarrow \delta(c, b) = \delta(a, \backsim)\\
c = c\oplus \backsim = f \oplus d &\Rightarrow \delta(c, f) = \delta(d, \backsim)\\
c = c\oplus \backsim = e \oplus e &\Rightarrow \delta(c, e) = \delta(e, \backsim)\\
c = b \oplus a = f \oplus d &\Rightarrow \delta(b, f) = \delta(d, a)\\
c = b \oplus a = e \oplus e &\Rightarrow \delta(b, e) = \delta(e, a)\\
c = f \oplus d = e \oplus e &\Rightarrow \delta(f, e) = \delta(e, d)
\end{aligned}$$

Considering all such equations deduced from analogical equations given by the analogy table, we can deduce constraints on the distance δ. In that way we can show (see [22]) that the distance table has only $\lfloor \frac{n}{2} \rfloor$ different values and has a circulant structure.

The table in Figure 3 represents a distance if the values of the variables (α, β, γ) verify the triangle inequality (positivity, symmetry and identity are verified if the values are positive).

We have a way to construct such a distance table by using the geometrical representation of a finite cyclic group in $\mathbb{R}^2$: we place the letters regularly on a circle (see Figure 4) and

δ	$\backsim$	a	b	c	d	e	f
$\backsim$	0	α	β	γ	γ	β	α
a	α	0	α	β	γ	γ	β
b	β	α	0	α	β	γ	γ
c	γ	β	α	0	α	β	γ
d	γ	γ	β	α	0	α	β
e	β	γ	γ	β	α	0	α
f	α	β	γ	γ	β	α	0

Figure 3: The table of the distance δ on the finite group $\mathcal{G}_7$.

we define the distance between letters as the euclidian distance in $\mathbb{R}^2$. This is sufficient to construct a distance, since every triple of elements is a triangle in $\mathbb{R}^2$.

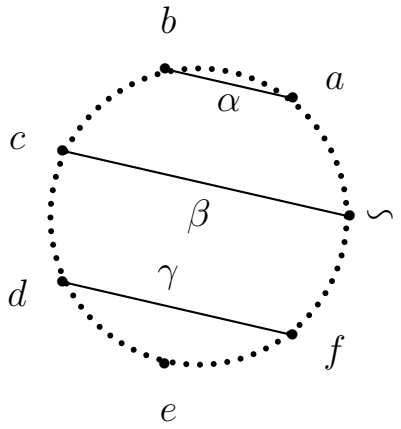

Figure 4: Representing $\mathcal{G}_7$, the additive cyclic finite group with 7 elements, and defining a distance on $\mathcal{G}_7$.

2.4 Analogy in the vector space $\mathbb{R}^n$.

We have shown in the previous sections how to build analogies and coherent distances in two different finite alphabets, the first one being defined by binary features, the second being defined as a finite group. We are interested now in the case where X is the vector space $\mathbb{R}^n$. An analogy between four objects, or vectors, in $\mathbb{R}^n$ is usually (see [32]) informally defined as follows: a, b, c, and d are four vectors in analogical relation if and only if they construct a parallelogram in $\mathbb{R}^n$.

2.4.1 An analogical relation in $\mathbb{R}^n$.

Let O be the origin of the vector space. Let $a = (a_1, a_2, \ldots, a_n)^\intercal$ be a vector of $\mathbb{R}^n$, as defined by its n coordinates. Let a, b, c and d be four vectors of $\mathbb{R}^n$. The interpretation of an analogy $a : b :: c : d$ is usually that a, b, c, d are the summits of a parallelogram, a and d being opposite summits. In the form of an analogical equation, written in the vectorial manner:

Definition 2.6 (Analogy in $\mathbb{R}^n$.) *Four vectors of $\mathbb{R}^n$ are in the analogy $a : b :: c : d$ if and only if they form a parallelogram:*

$$\overrightarrow{Oa} - \overrightarrow{Ob} = \overrightarrow{Oc} - \overrightarrow{Od}$$

which can be also written:

$$\overrightarrow{ab} = \overrightarrow{cd}$$

or equivalently

$$\overrightarrow{ac} = \overrightarrow{bd}$$

It is straightforward that the axioms of analogy, given in section 2.1 derive from this definition.

2.4.2 The transitivity of analogy in vector spaces.

In vector spaces, the analogy has also the property of transitivity:

Property 2.3 (Transitivity of analogy in $\mathbb{R}^n$.) *Let a, b, c, d, e and f be six vectors of $\mathbb{R}^n$. Then the following implication holds:*

$$(\ a : b :: c : d \quad \textit{and} \quad c : d :: e : f\ ) \Rightarrow \quad a : b :: e : f$$

The proof comes from definition 2.6.

2.4.3 A set of coherent distances.

We recall that a distance δ is said *coherent* with analogy if for every four-uple a, b, c and d which is in the analogical relation :

$$a : b :: c : d$$

the distance δ has the properties :

$$\delta(a, b) = \delta(c, d) \ \text{ and } \ \delta(a, c) = \delta(b, d)$$

In $\mathbb{R}^n$, any distance δ_p defined from the norm $\| \ \|_p$

$$\delta_p(a, b) = \|a - b\|_p = \big(\sum_{i=1}^{n} |a_i - b_i|^p\big)^{1/p}$$

is coherent with the analogy defined by $\overrightarrow{ab} = \overrightarrow{cd}$.

This is directly proven from a classical property of euclidian spaces: for every distance δ_p in $\mathbb{R}^n$, $\overrightarrow{ab} = \overrightarrow{cd}$ implies $\delta_p(a, b) = \delta_p(c, d)$.

3 Analogy between sequences.

In this section, we present two different manners to define an analogical relation between four sequences of objects. After having given some classical notations about sequences, we will firstly give a definition by Yvon, which refines and consolidates that by Lepage. Then we present a definition of ours, that we show to be more general. The objects which built the sequences can be elements of a finite alphabet (for our purpose, either defined by binary features or being a finite group) or vectors of an euclidian space.

3.1 Notations.

A *sequence*[1] is a finite series of symbols from an alphabet Σ. $\Sigma^\star$ is the set of all words. For x, y in $\Sigma^\star$, xy is the concatenation of x and y. We also denote $|\,x\,|$ the length of x, and $x = x_1 \dots x_{|x|}$ or $x = x[1] \dots x[n]$, with x_i or $x[i] \in \Sigma$ and $n = |\,x\,|$. We denote ϵ the empty word, of null length, and $\Sigma^+ = \Sigma^\star \backslash \{\epsilon\}$. Finally, we denote $\mathcal{L}(x)$ the subset of Σ in which are taken the letters of the word x and $\overline{\mathcal{L}(a)}$ the subset of Σ composed of the letters that do not appear in x.

A *factor* (or subword) f of a sequence x is a sequence in $\Sigma^\star$ such that there exists two sequences u and v in $\Sigma^\star$ with: $x = ufv$. For example, *abb*, *bbac* and *abbacbaba* are factors of *abbacbbaba*.

A *subsequence* of a sequence $x = x_1 \dots x_{|x|}$ is composed of the letters of x with the indices $i_1 \dots i_k$, such that $i_1 < i_2 \dots < i_k$. For example, *ca* and *aaa* are two subsequences of *abbacbaba*.

3.2 A first definition.

Yvon ([37]) gives the following definition of analogy between sequences:

Definition 3.1 (Analogy between sequences, a first definition.)
$(x, y, z, t) \in \Sigma^+$ *are in analogical relation, noted* $x : y :: z : t$ *if and only if* $\exists n > 0, \alpha_i, i \in [1, n], \beta_i, i \in [1, n] \in \Sigma^\star$ *such that, either:*

$$x = \alpha_1 \dots \alpha_n, t = \beta_1 \dots \beta_n, y = \alpha_1 \beta_2 \alpha_3 \dots \alpha_n, z = \beta_1 \alpha_2 \beta_3 \dots \beta_n$$
or
$$x = \alpha_1 \dots \alpha_n, t = \beta_1 \dots \beta_n, y = \beta_1 \alpha_2 \beta_3 \dots \alpha_n, z = \alpha_1 \beta_2 \alpha_3 \dots \beta_n$$

and $\forall i, \alpha_i \beta_i \neq \epsilon$.

The smallest integer n for which this property holds is called the degree *of the relation.*

For instance, *reception* : *refection* :: *deceptive* : *defective* , is an analogy between sequences, with $n = 3$ and the factors: $\alpha_1 = re$, $\alpha_2 = cept$, $\alpha_3 = ion$, $\beta_1 = de$, $\beta_2 = fect$, $\beta_3 = ive$.

We could also have chosen the following factors:

[1] More classically in language theory, a *word*.

$\alpha_1 = r$, $\alpha_2 = ecept$, $\alpha_3 = ion$ and accordingly $\beta_1 = d$, $\beta_2 = efect$ and $\beta_3 = ive$.

The degree of an analogical relation can be seen as a measure of its complexity: the smaller the degree, the better the analogy. This matches the intuition that good analogies should preserve large portions of the original words; the trivial analogy involving identical words.

Given this definition, the following properties hold (see also [17]):

$$\forall x \in \Sigma^+, x : x :: x : x \tag{3.1}$$

$$\forall x, y \in \Sigma^+ : x : x :: y : y \tag{3.2}$$

$$\forall x, y, z, t \in \Sigma^+ : x : y :: z : t \Rightarrow z : t :: x : y \tag{3.3}$$

$$\forall x, y, z, t \in \Sigma^+ : x : y :: z : t \Rightarrow x : z :: y : t \tag{3.4}$$

which proves that this definition of analogy is consistent with the axioms given in section 2.1.

Lepage and Yvon have proven that the two following conditions are necessary for the sequences x, y, z and t to be in analogy :

- Symbol inclusion:
$$\overline{\mathcal{L}(x)} \cup \overline{\mathcal{L}(t)} = \overline{\mathcal{L}(y)} \cup \overline{\mathcal{L}(z)} \tag{3.5}$$
- Similarity: $|x| + |t| = |y| + |z|$.

3.3 A second definition using analogy in alphabets.

3.3.1 Motivation.

The definition of analogy between strings given in the previous section is quite strict in the sense that the fourth term is constructed with letters that have appeared in the three others terms. Moreover, letters are considered as independent objects. In particular, if there is some analogical relation on the alphabet, it cannot be transmitted to sequences. We study now how to consider this possibility.

For example, assume that we have the alphabet $\Sigma = \{a, b, \alpha, \beta, B, C\}$ in which there exists the analogical relations: $a : b :: A : B$, $a : \alpha :: b : \beta$, $A : \alpha :: B : \beta$, and that the following analogical equation is proposed on $\Sigma^\star$: $aaBAB : \alpha\alpha bab :: bbBAB : x$ We certainly would conclude, by examining letter by letter, that $x = \beta\beta bab$. Such a solution cannot be obtained in the framework given in the last section, since the letter β appears nowhere in the first three terms of the equation.

Therefore, we would like to extend the definition of analogy between sequences to such cases. We also want also to accept analogies on sequences with no constraints on their lengths. This is why we have previously studied analogies on sets, which will be used as the alphabets of the sequences.

3.3.2 Analogy between sequences based on alignments.

We give here a more general definition of analogy between four sequences and show that it satisfies the axioms given in section 2.1.

Let Σ be an alphabet. We add a new letter to Σ, that we denote $\backsim$, giving the augmented alphabet Σ'. The interpretation of this new letter is simply that of an "empty" symbol, that we will need when computing edit distances between sequences in subsequent sections.

Definition 3.2 (Semantic equivalence.) *Let x be a sequence of $\Sigma^\star$ and y a sequence of $\Sigma'^\star$. x and y are* semantically equivalent *if the subsequence of y composed of letters of Σ is x. We denote this relation by $\equiv$.*

For example, $ab \backsim a \backsim a \equiv abaa$.

Let us assume that there is an analogy in Σ', i.e. that for every 4-uple a, b, c, d of letters of Σ', the relation $a : b :: c : d$ is defined as being either $TRUE$ or $FALSE$. Let δ be a distance on Σ', coherent with the analogy.

Definition 3.3 (Alignment between sequences.) *An* alignment *between two sequences $x, y \in \Sigma^\star$, of lengths m and n, is a word z on the alphabet $(\Sigma') \times (\Sigma') \backslash \{(\backsim, \backsim)\}$ which first projection is semantically equivalent to x and which second projection is semantically equivalent to y.*

Informally, an alignment represents a one-to-one letter matching between the two sequences, in which some letters $\backsim$ may be inserted. The matching $(\backsim, \backsim)$ is not permitted. An alignment can be presented as an array of two rows, one for x and one for y, each word completed with some $\backsim$, resulting in two words of Σ' having the same length.

For instance, here is an *alignment* between $x = abgef$ and $y = acde$:

$$\begin{array}{cccccccc} x' & = & a & b & \backsim & g & e & f \\ & & | & | & | & | & | & | \\ y' & = & a & c & d & \backsim & e & \backsim \end{array}$$

We can define in the same way an alignment between more sequences. The following definition uses alignments between four sequences.

Definition 3.4 (Analogy between sequences, a second definition.) *Let u, v, w and x be four sequences on $\Sigma^\star$, on which an analogy is defined. We say that u, v, w and x are in* analogy *in $\Sigma^\star$ if there exists four sequences u', v', w' and x' of same length n in Σ', with the following properties:*

1. *$u' \equiv u$, $v' \equiv v$, $w' \equiv w$ and $x' \equiv x$.*
2. *$\forall i \in [1, n]$ the analogies $u'_i : v'_i :: w'_i : x'_i$ hold true in Σ'.*

We prove here that this definition verifies the axioms of analogy.

These axioms hold true for each 4-uple u'_i, v'_i, w'_i and x'_i, i.e. :

1. $w'_i : x'_i :: u'_i : v'_i$
2. $u'_i : w'_i :: v'_i : x'_i$
3. $u'_i = v'_i \Rightarrow w'_i = x'_i$

Therefore, by concatenation of the n terms, we have in $\Sigma'^{\star}$:

1. $w' : x' :: u' : v'$
2. $u' : w' :: v' : x'$
3. $u' = v' \Rightarrow w' = x'$

And, by semantic equivalence, we have in $\Sigma^{\star}$:

1. $w : x :: u : v$
2. $u : w :: v : x$
3. $u = v \Rightarrow w = x$

which ensures that the axioms of analogy are verified for definition 3.4.

For example, let $\Sigma' = \{a, b, \alpha, \beta, B, C, \backsim\}$ with the analogies $a : b :: A : B$, $a : \alpha :: b : \beta$ and $A : \alpha :: B : \beta$. The following alignment between the four sequences aBA, αbBA, ba and βba is an analogy on $\Sigma^{\star}$:

$$\begin{array}{cccc} a & \backsim & B & A \\ | & | & | & | \\ \alpha & b & B & A \\ | & | & | & | \\ b & \backsim & a & \backsim \\ | & | & | & | \\ \beta & b & a & \backsim \end{array}$$

3.3.3 Connection between the two definitions.

We establish here the following property:

Property 3.1 *The analogy between sequences based on alignments that we have given at Definition 3.4 is strictly more general than that defined by Yvon and Stroppa (Definition 3.1).*

The demonstration consists in redefining an analogy by Yvon and Stroppa in terms of alignments. It is easy to see that the alignments corresponding to their definition are a subset of

all the possible alignments, since every column would have one of the two following particular forms, for $a \in \Sigma$:

$$\begin{array}{c} a \\ | \\ a \\ | \\ \backsim \\ | \\ \backsim \end{array} \quad \text{or} \quad \begin{array}{c} a \\ | \\ \backsim \\ | \\ a \\ | \\ \backsim \end{array}$$

4 Solving analogical equations in sets.

4.1 Analogical equations.

To solve an analogical equation consists in finding the fourth term of an analogical relation, the first three being known.

Definition 4.1 (Analogical equation.) *t is a solution of the analogical equation: $x : y :: z : ?$ if and only if $x : y :: z : t$.*

We already know from previous sections that, depending on the nature of the objects and the definition of analogy, an analogical equation may have no solution, a unique solution[2] or several solutions. We study in the sequel how to solve analogical equations in the different sets that we have introduced. Then we give two definitions of the solving analogical equations in sequences, the second being original, and we show that it is more general than the first one.

4.2 Solving analogical equations in finite sets defined by binary features.

Considering analogy in sets, Lepage ([18]) has shown the following theorem, with respect to the axioms of analogy (section 2.1) :

Theorem 4.2 (Solution of an analogical equation in sets.) *Let A, B and C be three sets. The analogical equation $A : B :: C : D$ where D is the unknown has a solution if and only if the following conditions hold true :*

$$A \subset B \cup C \quad \textit{and} \quad A \supset B \cap C$$

The solution is then unique, given by :

$$D = ((B \cup C) \backslash A) \cup (B \cap C)$$

[2]This is the only case where the analogy is transitive, see section 5.3.

If we assume that A, B and C are subsets of a set $\mathcal{P}$, the solution can be also written, denoting $\overline{A}$ the complement of A in $\mathcal{P}$, as the union of three disjoint sets

$$D = (B \cap \overline{A}) \cup (C \cap \overline{A}) \cup (A \cap B \cap C)$$

This theorem applies also to the resolution of analogical equations in a set X defined by binary features. Recall that for each $x \in X$ and each $i \in [1, n]$, $f_i(x) = 1$ (resp. $f_i(x) = 0$) means that the binary feature f_i takes the value $TRUE$ (resp. $FALSE$) on the object x.

Let $A : B :: C : D$ be an analogical equation where D is the unknown. For each feature f_i, there are only eight different possibilities of values on A, B and C. From the theorem above we can derive the manner of computing D, with the two following principles:

- Each feature $f_i(D)$ can be computed independently.
- The following table gives the solution $f_i(D)$:

$f_i(A)$	0	0	0	0	1	1	1	1
$f_i(B)$	0	0	1	1	0	0	1	1
$f_i(C)$	0	1	0	1	0	1	0	1
$f_i(D)$	0	1	1	?	?	0	0	1

In two cases among the eight, $f_i(D)$ does not exists. This derives from the defining of X by binary features, which is equivalent to defining X as a set of features. Theorem 4.2 imposes conditions on the resolution of analogical equations on finite sets, which results in the fact that two binary analogical equations have no solution.

4.3 Solving analogical equations in finite groups.

We have constructed alphabets as finite groups in section 2.3.1 with the explicit purpose that every analogical equation has one and only one solution. We show on an example how this solution is computed.

Let $\mathcal{G}_7$ be the group defined by the operator $\oplus$ given in Table 2 and the corresponding distance (Table 3). Recall that that four elements in a finite group are in analogy when

$$a \oplus x = b \oplus c \Leftrightarrow a : b :: c : x$$

Let's take as an example the resolution of the analogical equation $e : a :: c : x$. It consists in looking in the table of Figure 2 the value of $a \oplus c$, which gives d, and in searching in the same table what is the unique element x such that $a \oplus c = e \oplus x = d$, which gives $x = f$.

4.4 Solving analogical equations in $\mathbb{R}^n$.

Solving the analogical equation $u : v :: w : x$, where u, v and w are vectors of $\mathbb{R}^n$ and x is the unknown derives directly from the definition of analogy in vector spaces: the four vectors must form a parallelogram. There is always one and only one solution given by the equation:

$$\overrightarrow{Ox} = \overrightarrow{Ov} + \overrightarrow{Ow} - \overrightarrow{Ou}$$

5 Solving analogical equations in sequences.

5.1 Solving analogical equations in sequences : an algebraic method.

Solving an analogical equation consists in computing the fourth term of an analogical relation, given the three others. We consider in this section the first definition of analogy in sequences as defined by Yvon and Lepage (see section 3.2).

In this framework, not all analogical equations have a solution: for instance, the equation $abc : def :: ijk : ?$ does not have any solution. We have given in section 3.2 a couple of necessary conditions for an analogical equation to hold true, namely *symbol inclusion* and *similarity*.

Symbol inclusion requires that x, y, z and t can only be in analogy when all symbols in x occur either in y or in z. Then t contains precisely those symbols in y and z that are not found in x. Similarity requires that all the solutions of an analogical equation have the same length.

Conversely, some analogical equations may have more than one solution. For instance, $c : ac :: bc : ?$, has two equally acceptable solutions: abc and bac.

Yvon and Stroppa have shown that the set of solutions of an analogical relation on words, according to definition 3.2 can be expressed in the terms of two basic constructions on words and languages, the *shuffle* and the *complementary set* constructions.

5.1.1 Shuffle.

The notion of *shuffle* is introduced (eg. in [29]) as follows.

Definition 5.1 (Shuffle.) *If u and v are two words in $\Sigma^\star$, their shuffle is the language defined as:*

$$u \bullet v = \{u_1 v_1 u_2 v_2 \dots u_n v_n, \ \textit{with } u_i, v_i \in \Sigma^\star, u = u_1 \dots u_n, v = v_1 \dots v_n\}$$

Informally, the shuffle of two words u and v contains all the words w which can be composed using all the symbols in u and v, with the constraint that if a symbol a precedes b in u (or in v), then it must precede b in w.

For instance, if $u = abc$ and $v = def$, the words $abcdef$, $abdefc$, $adbecf$ and many others are in $u \bullet v$; this is not the case with $abefcd$, in which d occurs after, rather than before, e.

The shuffle operation has the following basic properties:

$$u \bullet \epsilon = \{u\} \ (\epsilon \text{ is "neutral"}) \tag{5.1}$$

$$u \bullet v = v \bullet u \ \text{(commutativity)} \tag{5.2}$$

$$(u \bullet v) \bullet w = u \bullet (v \bullet w) \ \text{(associativity)} \tag{5.3}$$

$$u(v \bullet w) \subset (uv) \bullet w \tag{5.4}$$

The shuffle is generalized to languages according to the following definition:

$$K \bullet L = \bigcup_{u \in L, v \in L} u \bullet v$$

As a simplification, we will identify $u \bullet v$ and $\{u\} \bullet \{v\}$.

5.1.2 Complementary subsequences and complementary sets.

The notion of the *complementary subsequence* is, in some respect, the converse of the shuffle operations, and is defined as follows:

Definition 5.2 (Complementary subsequences and set.) *If x is a subsequence of w, the complementary set of x with respect to w is defined as:*

$w\backslash x = \{y \in \Sigma^*, \exists I = \{i_1 \dots i_k\}, i_1 < \dots i_k,$ *st.* $y = w_{i_1} \dots w_{i_k}$ *and* $x = w_1 \dots$ $\dots w_{i_1-1} w_{i_1+1} \dots w_{i_2-1} \dots\}$. *If x is not a subsequence of w, $w\backslash x$ is empty.*

The complementary set of x with respect to w contains what "remains" of w when the symbols in x are removed. If y is in $w\backslash x$, we will say that y is *a complementary subsequence* of x in w. For instance, the complementary set of $false$ wrt. $falsehood$ is the singleton $\{hood\}$; the complementary subsequences of ive wrt. $derivative$ is the set: $\{derativ, dervati, derivat\}$. This operation can be turned into a symmetric binary relationship as follows:

Definition 5.3 (Complementary relationship.) *$w \in \Sigma^\star$ define a binary relationship denoted $\backslash_w$ and defined as: $u \backslash_w v$ if and only if $u \in w\backslash v$.*

The complementary set of a word x wrt. a language L generalizes this notion and is defined as:

$$L\backslash x = \bigcup_{w \in L} w\backslash x$$

Similarly, one can also define the complementary set of a language K wrt. a word w as:

$$w\backslash L = \bigcup_{x \in L} w\backslash x$$

This operation is finally extended to languages: if K and L are two languages, the complementary $L\backslash K$ of K with respect to L is the union over words in L of their complementary set with respect to K:

$$L\backslash K = \bigcup_{w \in L, x \in K} w\backslash x = \bigcup_{w \in L} w\backslash K = \bigcup_{x \in K} L\backslash x$$

The notions of complementary set and shuffle are related through the following properties:

Property 5.1

$$w \in u \bullet v \Leftrightarrow u \in w\backslash v$$

We will also need the following property:

Property 5.2

$$\forall u, v \in \Sigma^\star, \forall w \text{ subsequence of } u : (u \backslash w) \bullet v \subset (u \bullet v) \backslash w$$

The following theorem yields a formal definition of the set of solutions of an analogical equation ([37]):

Theorem 5.4

$$t \text{ is a solution of } \quad x : y :: z : ? \quad \Leftrightarrow t \in y \bullet z \backslash x$$

Not all analogical equations have a solution in this framework. The solution must fulfill the necessary conditions given in section 3.2. In particular, the symbol inclusion condition implies that every letter of the solution, if there is one, must appear in one of the three first terms of the equation.

5.2 Solving analogical equations in sequences using the edit distance.

We show here how to use the *edit distance* to define what a sequence is to another one, and to define the "as" relation of the analogy from the way the edit distance is computed. Informally, in the analogy $aaBAB : \alpha\alpha bab :: bbBAB : \beta\beta bab$, the first sequence is transformed into the second with the series of transformations of a letter x into another letter y, denoted $\mathcal{S}_{x \to y}$:

$$\mathcal{S}_{a\to\alpha}\mathcal{S}_{a\to\alpha}\mathcal{S}_{B\to b}\mathcal{S}_{A\to a}\mathcal{S}_{B\to b}$$

Similarly, the sequence $bbBAB$ is transformed into the sequence $\beta\beta bab$ with the sequence:

$$\mathcal{S}_{b\to\beta}\mathcal{S}_{b\to\beta}\mathcal{S}_{B\to b}\mathcal{S}_{A\to a}\mathcal{S}_{B\to b}$$

Expressing that the four sequences are in analogy is, in this simple case, merely matching the two sequences of transformations one to one from left to right, and noting that the corresponding 4-uple of letters are in analogy in the alphabet:

$\mathcal{S}_{a\to\alpha}$	$\mathcal{S}_{a\to\alpha}$	$\mathcal{S}_{B\to b}$	$\mathcal{S}_{A\to a}$	$\mathcal{S}_{B\to b}$
$\mathcal{S}_{b\to\beta}$	$\mathcal{S}_{b\to\beta}$	$\mathcal{S}_{B\to b}$	$\mathcal{S}_{A\to a}$	$\mathcal{S}_{B\to b}$
$a : \alpha :: b : \beta$	$a : \alpha :: b : \beta$	$B : b :: B : b$	$A : a :: A : a$	$B : b :: B : b$

Actually, the edit distance can not only deal with substitutions between letters, but also with insertions and deletions of letters, as explained in the following section. This is why we have introduced a special symbol $\backsim$ when defining the analogy between sequences at section 3.3.2.

5.2.1 The edit distance between sequences.

To introduce the edit distance, we have to give more definitions and quote a theorem, demonstrated by Wagner and Fischer in [33]. Firstly, we present the notion of *edition* between sequences, based on three *edit operations* between letters: the insertion of a letter in

the target sequence, the deletion of a letter in the source sequence and the substitution, which means replacing a letter in the source sequence by another letter in the target sequence. Each of these operations can be associated to some positive cost. We denote $\mathcal{S}_{a\to b}$ the substitution from a into b, with a positive cost $\delta(a,b)$, $\mathcal{S}_{a\to\backsim}$ the deletion of a with a positive cost $\delta(a,\backsim)$, and $\mathcal{S}_{\backsim\to b}$ the insertion of b with a positive cost $\delta(b,\backsim)$. The cost of the edition between sequences is the sum of the costs of the operations between letters required to transform the source sequence into the target one.

We recall what is an alignment (Definition 3.3).

An *alignment* between two words $x, y \in \Sigma^*$, is a word z on the alphabet $(\Sigma\cup\{\backsim\})\times(\Sigma\cup\{\backsim\})\setminus\{(\backsim,\backsim)\}$ which projection on the first component is semantically equivalent to x and which projection on the second component is is semantically equivalent to y.

We introduce now the edit distance between sequences, and show a sufficient condition for four sequences to be in analogy by using this distance.

Definition 5.5 (Distance on augmented alphabets.) *We denote now $\Sigma' = \Sigma \cup \{\backsim\}$ and we say that δ is a distance*[3] *on the augmented alphabet Σ' iff :*

1. *δ is an application of $\Sigma' \times \Sigma'$ on $\mathbb{R}^+$, defined for every couple of elements, except for $\delta(\backsim,\backsim)$.*
2. *$\forall a, b \in \Sigma'$ for which δ is defined : $\delta(a,b) = 0 \Leftrightarrow a = b$*
3. *$\forall a, b \in \Sigma'$ for which δ is defined : $\delta(a,b) = \delta(b,a)$*
4. *$\forall a, b, c \in \Sigma'$ for which δ is defined : $\delta(a,b) \leq \delta(a,c) + \delta(c,b)$*

Theorem 5.6 (Edit distance between sequences([33]).) *Let δ be a distance on Σ', and x and y be sequences of $\Sigma^\star$. The* edit distance *D is the cost of an* alignment *with the lowest cost that transforms x into y.*

An alignment corresponding to the edit distance, that of lowest cost, is called *optimal*. We denote $\mathcal{S}(x,y)$ the sequence of transformations corresponding to the optimal alignment[4].

It is now possible to use the classical dynamic programming Wagner and Fisher algorithm ([33], algorithm 1) which computes the edit distance and the optimal alignment.

A consequence of this algorithm is the following remarkable result [8], which justifies the name of edit *distance* :

Theorem 5.7 *If δ is a distance on the augmented alphabet Σ' then D is a distance*[5] *on Σ^*.*

This algorithm can be completed in constructing the optimal alignment between x and y, or all the optimal alignments if there are more than one. This is done by keeping more

[3] We keep the word "distance", since this definition is only slightly adapted from the classical one.

[4] There may be several optimal alignments. We will examine this case in section 5.2.6. For the sake of simplicity, we will presently assume that there is an unique optimal alignment.

[5] In the usual sense, given at definition 2.2.

Algorithm 1 Computing the edit distance $D(x, y)$ between two sequences x and y in Σ^* with the distance δ defined on Σ'.

begin
$M(0,0) \leftarrow 0$;
for $i = 1, m$ **do**
  $M(i,0) \leftarrow \sum_{k=1}^{k=i} \delta(x_k, \backsim)$;
end for
for $j = 1, n$ **do**
  $M(0,j) \leftarrow \sum_{k=1}^{k=j} \delta(\backsim, y_k)$;
end for
for $i = 1, m$ **do**
  for $j = 1, n$ **do**

$$M(i,j) \leftarrow \min \begin{cases} M(i-1,j) + \delta(x_i, \backsim) \\ M(i,j-1) + \delta(\backsim, y_j) \\ M(i-1,j-1) + \delta(x_i, y_j) \end{cases}$$

  end for
end for
$D(x,y) \leftarrow M(m,n)$
end

information during the computation and by backtracking on the optimal paths in the final matrix M (see [30]) computed by the algorithm.

For example, on the augmented alphabet $\Sigma = \{\backsim, a, b, c, d, e, f, g\}$, if the costs $\delta(.,.)$ are all equal to unity, the unique optimal alignment between $x = abgef$ and $y = acde$ is given by:

$$\begin{array}{ccccccc} x' & = & a & b & g & e & f \\ & & | & | & | & | & | \\ y' & = & a & c & d & e & \backsim \end{array}$$

It is defined by the sequence

$$\mathcal{S}(x,y) = \mathcal{S}_{a\to a}\mathcal{S}_{b\to c}\mathcal{S}_{g\to d}\mathcal{S}_{e\to e}\mathcal{S}_{f\to\backsim}$$

and the edit distance between x and y is :

$$D(x,y) = \delta(a,a) + \delta(b,c) + \delta(g,d) + \delta(e,e) + \delta(f,\backsim) = 0 + 1 + 1 + 0 + 1 = 3$$

5.2.2 Edit distance and analogy

Let $\mathbb{S}$ be the set of all elements $\mathcal{S}_{a\to b}$, where a and b are elements of Σ', except that the element $\mathcal{S}_{\backsim\to\backsim}$ is not in $\mathbb{S}$.

Hence, $\mathbb{S}^*$ is the set of all sequences of transformations between letters of Σ' (with the exception of $\mathcal{S}_{\backsim\to\backsim}$). We can augment $\mathbb{S}$ with a new element, giving an alphabet $\mathbb{S}'$, like we have done for Σ in section 3.3.2, to allow deletions and insertions between elements of $\mathbb{S}$.

We can also assume that there exists a distance Δ on the augmented alphabet $\mathbb{S}'$, with the same definition than in 5.5. This will allow the construction of an edit distance in $\mathbb{S}^*$.

To relate δ and Δ, we only require at the time being that $\Delta(S_{a\to b}, S_{c\to d})$ is null if and only if $a : b :: c : d$ holds true and that δ is coherent with the analogy on Σ. Then the following property can be given:

Property 5.3 *Let u, v, w and x four sequences on Σ^*. Let $\mathcal{S}(u, v)$ be the optimal sequence of edit operations between u and v, based on the distance δ defined on Σ', and similarly let $\mathcal{S}(w, x)$ be the optimal sequence of edit operations between w and x. There exists at least one optimal alignment, based on distance Δ, between $\mathcal{S}(u, v)$ and $\mathcal{S}(w, x)$. If the cost of this optimal alignment is null, then u, v, w and x are four sequences in analogy according to definition 3.4.*

The proof of this property derives directly from definition 3.4 and from the definition of the edit distance (see the array displaying an alignment in section 3.3.2).

5.2.3 Resolving analogical equations using the edit distance: a first method.

Let u, v, and w be three sequences in the alphabet Σ. We want to find a solution x to the analogical equation : $u : v :: w : x$.

We propose here a first algorithm, that we have called SEQUANA1, using three steps :

- computation of the[6] optimal alignment between u and v, producing two new sequences u' and v' of same length in Σ'. u' and v' are semantically equivalent to u and v, since some $\backsim$ have possibly been inserted according to the optimal alignment, as shown in section 3.3.2. In the same manner, we assume that u and w are processed and that the optimal alignment is established, giving two new sentences u'' and w' of same length in Σ'.
- Insertion of some $\backsim$ so as to get three sequences in Σ', denoted u''', v''' and w''', of same length, such that u''' is semantically equal to u (it is constructed in inserting some $\backsim$ in u'') and v''' is semantically equal to v.
- Resolution of the elementary analogical equations : every triple of letters with the same index, u'''_l, v'''_l and w'''_l will correspond to an elementary analogical equation on letters in Σ'.

The i^{th} letter of a sequence s is denoted here $s[i]$. The index i will be used for the sequence u', the index i' for the sequence u'', the index j for the sequence v' and the index k for the sequence w'. We analyse now the algorithm in more detail.

There are in principle sixteen different situations corresponding to the fact that u'_i, $u''_{i'}$, v'_j and w'_k may be equal to $\backsim$ or not. Only eleven are actually met, since u' and u'' are explored in synchrony: whenever the algorithm meets $u'_i \neq \backsim$ and $u''_{i'} \neq \backsim$, then u'_i is the same letter as $u''_{i'}$. In this case, we notice that there are four common situations for which we do the same operations, that is:

[6] We still assume that it is unique (we will relax this constraint in section 5.2.6).

Algorithm 2 Algorithm SEQUANA1: computing the solution to an analogical equation on sequences.

begin
STATE $i \leftarrow 1$; $i' \leftarrow 1$; $j \leftarrow 1$; $k \leftarrow 1$; $l \leftarrow 1$.
while $(u'_i \neq \backsim)$ OR $(u''_{i'} \neq \backsim)$ OR $(v'_j \neq \backsim)$ OR $(w'_k \neq \backsim)$ **do**
  According to the values of u'_i, $u''_{i'}$, v'_j and w'_k,
    assign a value to u'''_l, v'''_l and w'''_l
    and increase the values of i, i', j and k by 0 or 1
  $l \leftarrow l + 1$
end while
for $l = 1, |u'''|$ **do**
  Solve on the alphabet the analogical equation:
  $u'''_l : v'''_l :: w'''_l : x'''_l$
end for
end

- storing the values of $u'_i = u''_{i'}$ in u'''_l, of v'_j in v'''_l, of w'_k in w'''_l, and
- incrementing all indices.

The situation where $u'_i = u''_{i'} = \backsim$ and $v'_j \neq \backsim$ and $w'_k \neq \backsim$ is a particular case. This situation corresponds to an insertion from u'_i into v'_j and from $u''_{i'}$ into w'_k and we have to keep both $\backsim$ for the two insertions. As a consequence, we keep here both solutions by considering one insertion after the other and *vice versa*.

Finally we actually get only four different cases in this algorithm that are described in the following list :

Case 1.
We have $u'_i = u''_{i'} \neq \backsim$
$u'''_l \leftarrow u'_i = u''_{i'}$; $v'''_l \leftarrow v'_j$; $w'''_l \leftarrow w'_k$.
$i \leftarrow i + 1$; $i' \leftarrow i' + 1$; $j \leftarrow j + 1$; $k \leftarrow k + 1$; $l \leftarrow l + 1$.

Case 2.
We have $u'_i = \backsim$ and $u''_{i'} \neq \backsim$.
$u'''_l \leftarrow \backsim$; $v'''_l \leftarrow v'_j$; $w'''_l \leftarrow \backsim$.
$i \leftarrow i + 1$; $j \leftarrow j + 1$; $l \leftarrow l + 1$.

Case 3.
We have $u'_i \neq \backsim$ and $u''_{i'} = \backsim$.
$u'''_l \leftarrow \backsim$; $v'''_l \leftarrow \backsim$; $w'''_l \leftarrow w'_k$.
$i' \leftarrow i' + 1$; $k \leftarrow k + 1$; $l \leftarrow l + 1$.

Case 4.
We have $u'_i = \backsim$ and $u''_{i'} = \backsim$ and $v'_j \neq \backsim$ and $w'_k \neq \backsim$.
There are three possible solutions (all are *a priori* valid):

Case 4.1 If u' is chosen and not u'': $u'''_l \leftarrow\backsim$; $v'''_l \leftarrow v'_j$; $w'''_l \leftarrow\backsim$.
$i \leftarrow i+1$; $j \leftarrow j+1$; $l \leftarrow l+1$.

Case 4.2 If u'' is chosen and not u': $u'''_l \leftarrow\backsim$; $v'''_l \leftarrow\backsim$; $w'''_l \leftarrow w'_k$.
$i' \leftarrow i'+1$; $k \leftarrow k+1$; $l \leftarrow l+1$.

Case 4.3 If u' and u'' are both chosen: $u'''_l \leftarrow\backsim$; $v'''_l \leftarrow v'_j$; $w'''_l \leftarrow w'_k$.
$i \leftarrow i+1$; $j \leftarrow j+1$; $i' \leftarrow i'+1$; $k \leftarrow k+1$; $l \leftarrow l+1$.

The following table represents the dispatching of all the situations on the three cases :

Number of the case	$u'_i \neq\backsim$ $v'_j \neq\backsim$	$u'_i \neq\backsim$ $v'_j =\backsim$	$u'_i =\backsim$ $v'_j \neq\backsim$	$u'_i =\backsim$ $v'_j =\backsim$
$u''_{i'} \neq\backsim$ $w'_k \neq\backsim$	1	1	2	
$u''_{i'} \neq\backsim$ $w'_k =\backsim$	1	1	2	
$u''_{i'} =\backsim$ $w'_k \neq\backsim$	3	3	4	End
$u''_{i'} =\backsim$ $w'_k =\backsim$			End	

When the algorithm terminates, there are two possibilities :

- Either every analogical equation in Σ' that we have built has a solution. In this case, the solution x''' in Σ'^* is constructed as the concatenation of the solutions in Σ' and is semantically equivalent to a sentence x in Σ^*, which is the solution of the equation $u : v :: w : x$.
- Or some analogical equations in Σ' have no solution. We can either declare that $u : v :: w : x$ has no solution, or compute x'''_l from u'''_l, v'''_l and w'''_l by choosing for x'''_l the letter of Σ' which has the less *analogical dissimilarity* with the three others (this notion is introduced at section 6).

5.2.4 Resolving analogical equations using the edit distance: a second method.

To solve the analogical equation $u : v :: w : x$, we can also use a direct method. Rather than computing the optimal alignment between u and v, then that of u and w, and then deduce x from these two alignments, we can progress from left to right in the three sequences u, v and w and compute x during this progression, in a three-dimensional dynamic programming process. We give here an algorithm (Algorithm 3), called SEQUANA2 based on this principle and demonstrate that it gives the same result than SEQUANA1.

This algorithm computes the cost of the alignment produced by the following sum:

$$\Sigma_{i=1}^{p}(\delta(u'_i, v'_i) + \delta(u'_i, w'_i))$$

where p is the length of the alignment, and where u'_i, v'_i and w'_i are letters in Σ'.

Algorithm 3 Algorithm SEQUANA2: computing the solution to an analogical equation on sequences.

build the edit table using DPA with 3 sequences :
$D[0,0,0] \leftarrow 0$
for $i = 1, |u|$ **do**
  $D[i,0,0] \leftarrow \Sigma_{l=1}^{l=i}(\delta(u_l, \backsim) + \delta(u_l, \backsim))$
end for
for $j = 1, |v|$ **do**
  $D[0,j,0] \leftarrow \Sigma_{l=1}^{l=j}(\delta(\backsim, v_l))$
end for
for $k = 1, |w|$ **do**
  $D[0,0,k] \leftarrow \Sigma_{l=1}^{l=k}(\delta(\backsim, w_l))$
end for
for $(i,j,k) = (1,1,1)..(|u|,|v|,|w|)$ **do**

$$D[i,j,k] \leftarrow \min \begin{cases} D[i-1,j-1,k-1] + \delta(u_i, v_j) + \delta(u_i, w_k) \\ D[i,j-1,k-1] + \delta(\backsim, v_j) + \delta(\backsim, w_k) \\ D[i-1,j,k-1] + \delta(u_i, \backsim) + \delta(u_i, w_k) \\ D[i-1,j-1,k] + \delta(u_i, v_j) + \delta(u_i, \backsim) \\ D[i,j,k-1] + \delta(\backsim, w_k) \\ D[i,j-1,k] + \delta(\backsim, v_j) \\ D[i-1,j,k] + \delta(u_i, \backsim) + \delta(u_i, \backsim) \end{cases}$$

end for
//perform the backtrack through the edit table:
$i \leftarrow |u|; j \leftarrow |v|; k \leftarrow |w|;$
while $(i,j,k) \neq (0,0,0)$ **do**
  put the transformation suitable with the edit distance at the end of the sequence:
  if $D[i,j,k]$ = value of the table with i,j,k decreased from 1 or 0 when it exists + cost of the corresponding transformation **then**
    According to the transformation :
    ($u\prime_l \leftarrow u_i$ and $i \leftarrow i-1$) or $u\prime_l \leftarrow \backsim$;
    ($v\prime_l \leftarrow v_j$ and $i \leftarrow j-1$) or $v\prime_l \leftarrow \backsim$;
    ($w\prime_l \leftarrow w_k$ and $i \leftarrow k-1$) or $w\prime_l \leftarrow \backsim$;
  end if
  $l \leftarrow l-1$
end while

5.2.5 The two algorithms are equivalent.

SEQUANA1 and SEQUANA2 only differ in the production of alignments of three sequences. The resolution process is exactly the same for both algorithms.

Given a cost between three letters, the algorithm SEQUANA2 uses the same principle as the WAGNER and FISHER algorithm in three dimensions. Then it produces all the optimal alignments possible between three sequences, that is alignments of minimal cost. As a consequence, all solutions produced by SEQUANA1 can be obtained by SEQUANA2 if these solutions are optimal in the sense of SEQUANA2. So we have to prove that optimal solutions of SEQUANA1 are optimal for SEQUANA2.

In SEQUANA2, the cost of an alignment is simply the sum of costs of the triplets. In SEQUANA1, the cost of the alignments of two sequences is the classical edit distance. By combining the two alignments of the first step of SEQUANA1, we get a new alignment of three sequences which cost is simply the sum of the two edit distances found at the first step. The reason is that the combination of the two alignements of two sequences is made by adding null cost operations (namely $\delta(\backsim, \backsim)$) or just keeping both edit operations. As a consequence, the cost of alignments produced by SEQUANA1 and SEQUANA2 have exactly the same definition and the optimal solutions of SEQUANA1 are also produced by SEQUANA2.

Conversely, it remains to prove that all the alignments produced by SEQUANA2 can also be obtained with SEQUANA1. To reach this goal, we consider how SEQUANA2 produces an alignment: this alignment is a ordered sequence of triplets of letters and we saw that there are 7 possible configurations of a triplet in an alignement generated by SEQUANA2. We then have to prove that each of these 7 cases corresponds to at least one case of SEQUANA1. In the following table, we enumerate these 7 cases.

Case of SEQUANA2	Case of SEQUANA1
(u'_i, v'_i, w'_i)	Case 1
$(\backsim, v'_i, w'_i)$	Case 4.3
$(u'_i, \backsim, w'_i)$	Case 1 with $v'_j = \backsim$
$(u'_i, v'_i, \backsim)$	Case 1 with $w'_k = \backsim$
$(\backsim, \backsim, w'_i)$	Case 3 or Case 4.2
$(\backsim, v'_i, \backsim)$	Case 2 or Case 4.1
$(u'_i, \backsim, \backsim)$	Case 1 with $v'_j = w'_k = \backsim$

At this point, we know that an optimal sequence of triplets produced by SEAQUANA2 can be decomposed into two sequences of doublets. We now have to prove that this sequences of doublets could have been produced by the first step of SEQUANA1 (alignments of two sequences). We prove in the following that every decomposition of an alignment of three sequences gives two optimal alignments of two sequences, and as a consequence two alignments produced by the first step of SEQUANA1.

Let S'_3 be an optimal alignment of three sequences, and (S'_1, S'_2) its decomposition into two alignments of two sequences with the method that we have just explained above. Let

S_{1opt} and S_{2opt} be two optimal alignments of two sequences produced by the first step of SEQUANA1, and S_3 the combination of S_{1opt} and S_{2opt} by the second step of SEQUANA1. We denote $\mathcal{C}(S)$ the edit cost of a sequence.

By construction, $\mathcal{C}(S_3) = \mathcal{C}(S_{1opt}) + \mathcal{C}(S_{2opt})$ and $\mathcal{C}(S'_3) = \mathcal{C}(S'_1) + \mathcal{C}(S'_2)$. As S'_3 is optimal for SEQUANA2, we have $\mathcal{C}(S'_3) \leq \mathcal{C}(S_3)$, then

$$\mathcal{C}(S'_1) + \mathcal{C}(S'_2) \leq \mathcal{C}(S_{1opt}) + \mathcal{C}(S_{2opt}) \tag{5.5}$$

By definition of the first step of SEQUANA1, we also have

$$\mathcal{C}(S_{1opt}) \leq \mathcal{C}(S'_1) \tag{5.6}$$

$$\mathcal{C}(S_{2opt}) \leq \mathcal{C}(S'_2) \tag{5.7}$$

From equations 5.5 and 5.6, we deduce that $\mathcal{C}(S'_1) + \mathcal{C}(S'_2) \leq \mathcal{C}(S_{1opt}) + \mathcal{C}(S'_2)$ then $\mathcal{C}(S'_1) \leq \mathcal{C}(S_{1opt})$. With this last equation and equation 5.6, we have $\mathcal{C}(S'_1) = \mathcal{C}(S_{1opt})$. In the same way, from equations 5.5 and 5.7, we can deduce that $\mathcal{C}(S'_2) = \mathcal{C}(S_{2opt})$.

As a conclusion, the decomposition of an optimal alignment of SEQUANA2 is optimal for SEQUANA1, then is produced by SEQUANA1. We conclude that both algorithms produce the same optimal alignments of three sequences.

5.2.6 The case of multiple optimal solutions and the compared complexity of the two algorithms.

We are interested in this section in the time complexity of SEQUANA1 and SEQUANA2, in terms of comparisons, the basic operation in the dynamic programming algorithms that we use. We want to give an order of complexity to the task of finding all solutions to the analogical equation $u : v :: w : x$ on sequences.

So far, we have assumed that there is only one optimal alignment when computing the edit distance between sequences. Obviously, this assumption has to be relaxed since it is generally not true. We have given an example at section 5.3: the alphabet $\Sigma = \{a, b, c\}$ with all distances equal to 0 or 1 gives two alignments with the optimal cost of 2 between ab and c:

$$\begin{array}{ccc} a & b & \\ | & | & \text{or} \\ \backsim & c & \end{array} \qquad \begin{array}{ccc} a & b & \\ | & | & \\ c & \backsim & \end{array}$$

The SEQUANA1 algorithm has to take this problem into account. Let us denote $\aleph(u, v)$ the set of optimal sequences of elementary transformations between the sequences u and v. In the above example, with $u = ab$ and $v = c$, we have: $\aleph(u, v) = \{S_{a\to\backsim}S_{b\to c}\ ,\ S_{a\to c}S_{b\to\backsim}\}$ and $|\aleph(u, v)| = 2$.

When computing the solution of the analogical equation $u : v :: w : x$, SEQUANA1 has firstly to compute all the optimal alignments between u and v and all the optimal alignments between u and w. Then it has to compute all the minimum cost of aligning one element of $\aleph(u, v)$ and one element of $\aleph(u, w)$. We also know that:

- The length of an optimal sequence of elementary alignments between u and v is certainly less than $|u| + |v|$.
- The number of comparisons to compute an alignment between u and v is in $\mathcal{O}(|u| \times |v|)$.

Finally, the worst case complexity of SEQUANA1 for producing all solutions is in:

$$\mathcal{O}\big(|\aleph(u,v)| \times |\aleph(u,w)| \times ((|u| + |v|) \times (|u| + |w|)) \big)$$

This complexity is likely to be reduced by using the fact that the elements of $\aleph(u,v)$ (and $\aleph(u,w)$) are not independent, but can be factorized in a DAG structure. We have not investigated this point any further.

SEQUANA2 produces the solution with a three dimensional dynamic programming process of worst case complexity in

$$\mathcal{O}\big(|u| \times |v| \times |w| \big)$$

and the set of all solutions (which can be written $\aleph(u,v,w)$) can be listed with the same order of time complexity by backtracking in the resulting array.

The two complexities are not directly comparable, since $|\aleph(u,v)|$ and $|\aleph(u,w)|$ depend on u, v, w and on the distance in the alphabet. All that we know for sure is that:

$$|\aleph(u,v,w)| \leq |\aleph(u,v)| \times |\aleph(u,w)|$$

5.3 The transitivity of analogy in sequences.

We can now wonder whether or not the analogy in sequences has the property of transitivity, and give a negative answer. This comes from the remark that there may be several sequences in analogy with three given sequences (which was not the case in the previous cases). For example, in the alphabet $\Sigma = \{a, b, c\}$ with the distance table:

δ	a	b	c	$\backsim$
a	0	1	1	1
b	1	0	1	1
c	1	1	0	1
$\backsim$	1	1	1	

the two following analogical equations hold true: $ab : c :: aabb : abc$ and $ab : c :: aabb : acb$. The first corresponds to the alignments

$$\begin{array}{ccc} a & b & \\ | & | & \text{and} \\ \backsim & c & \end{array} \qquad \begin{array}{cccc} a & a & b & b \\ | & | & | & | \\ \backsim & a & b & c \end{array}$$

and the second to the different alignments:

$$\begin{array}{cc} a & b \\ | & | \\ c & \backsim \end{array} \quad \text{and} \quad \begin{array}{cccc} a & a & b & b \\ | & | & | & | \\ a & c & \backsim & b \end{array}$$

Such a situation implies that there is no transitivity, since it would negate the third axiom of analogy with the following chain of implications:

$$\begin{cases} ab : c :: aabb : abc \\ ab : c :: aabb : acb \end{cases} \Rightarrow \begin{cases} aabb : abc :: ab : c \\ ab : c :: aabb : acb \end{cases} \quad \text{(Transitivity)}$$

$$\Rightarrow aabb : abc :: aabb : acb \quad \Rightarrow \quad aabb : aaab :: abc : acb$$

and the third axiom would conclude: $abc = acb$. Therefore, when there are several different objects in analogy with three given objects, no transitivity is possible.

5.4 Our algorithms are more general than the algebraic method.

The following property compares the set of solutions given by what we have called the algebraic method (Theorem 5.4) and the set of solutions given by SEQUANA1 or SEQUANA2.

Property 5.4 (The set of solutions given by SEQUANA.)
SEQUANA1 or SEQUANA2 can be used to find the set of solutions given by the algebraic algorithm of Yvon (Theorem 5.4).

To demonstrate this property, it is sufficient to notice that the only analogies in Σ' which are accepted in the first definition are $a : a ::\backsim:\backsim$ and $a :\backsim:: a :\backsim$, for each letter a in Σ. Then the definition given by Yvon and Stroppa is included by ours if we can find a distance such that the only possible alignments are of these type. It merely consists in giving a very high value to $\delta(a, b)$, when $a \neq b$ and a and b are in Σ, and an equal value (say 1) to all $\delta(a, \backsim)$ and $\delta(\backsim, a)$ when a is are in Σ.

6 Analogical dissimilarity between sets.

6.1 Motivation.

In this section, we are interested in defining what could be a relaxed analogy, which linguistic expression would be "a is to b *almost as* c is to d. To remain coherent with our previous definitions, we measure the term "almost" by some positive real value, equal to 0 when the analogy stands true, and increasing when the four objects are less likely to be in analogy. We also want this value, that we call "analogical dissimilarity", to have good properties with respect to the analogy. We want it to be symmetrical, to stay unchanged when we permute the mean terms of the analogy and finally to respect some triangle inequality. These requirements will allow us, in section 8, to generalize a classical fast nearest neighbor search algorithm and to exhibit an algorithmic learning process which principle is to extract,

from a learning set, the triplet of objects that has the least AD when combined with another unknown object. This lazy learning technique is a therefore a generalization of the classical nearest neighbor method.

We firstly study the definition of the analogical dissimilarity on the same structured sets than in the previous sections, and secondly extend it to sequences.

6.2 A definition in finite sets defined by binary features.

6.2.1 Definition.

We know from section 4.2 that four elements in a finite set X defined by binary features are in analogical relation $u : v :: w : x$ if and only if, for every feature, its four values form one of the six columns of the following table:

$f_i(u)$	0	0	0	0	1	1	1	1
$f_i(v)$	0	0	1	1	0	0	1	1
$f_i(w)$	0	1	0	1	0	1	0	1
$f_i(x)$	0	1	1	?	?	0	0	1

The analogical dissimilarity between four objects must reflect in some way how far they are from being in analogical relation. We firstly define what it can be when the four objects are defined on only one feature, secondly on any number of features.

Definition 6.1 (Analogical dissimilarity between binary features.) *The analogical dissimilarity between four binary values is given by the following table:*

u	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*
v	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*
w	*0*	*0*	*1*	*1*	*0*	*0*	*1*	*1*	*0*	*0*	*1*	*1*	*0*	*0*	*1*	*1*
x	*0*	*1*	*0*	*1*	*0*	*1*	*0*	*1*	*0*	*1*	*0*	*1*	*0*	*1*	*0*	*1*
$AD(u,v,w,t)$	*0*	*1*	*1*	*0*	*1*	*0*	*2*	*1*	*1*	*2*	*0*	*1*	*0*	*1*	*1*	*0*
							1			*1*						

There are two special cases in this table where two values ("2" or "1") are indicated as possible. Taking "1" has the advantage to keep the elementary dissimilarities as binary values. Choosing "2" instead reflects the fact that these dissimilarities are somehow greater than the others, since the solution of the corresponding equation does not exists and cannot be compared to the value of the fourth term. This choice has no consequence on the properties of AD in sets defined by binary features (see property 6.1).

Definition 6.2 (Analogical dissimilarity in sets of binary features.) *The analogical dissimilarity $AD(u,v,w,t)$ between four objects u, v, w and t of a finite set X defined by binary features is the sum of the values of the analogical dissimilarities between the features.*

6.2.2 Example.

Lets us take four animals: crow, raven, cat and cheetah, as described with the following features:

f_1	Is an animal
f_2	Has a name beginning with letter "c"
f_3	Has a big size
f_4	Is a domestic animal
f_5	Is a mammal
f_6	Is a fish
f_7	Can fly
f_8	Has a letter "a" in second position of its name
f_9	May be black

This gives the following table (when choosing value 2 in the special cases):

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9
crow	1	1	0	0	0	0	1	0	1
raven	1	0	1	0	0	0	1	1	1
cat	1	1	0	1	1	0	0	1	1
cheetah	1	1	1	0	1	0	0	0	0
AD	0	1	0	1	0	0	0	2	1

And in this example: $AD\big(\text{crow}, \text{raven}, \text{cat}, \text{cheetah}) =$
$\sum_{i=1}^{9} AD(f_i(\text{crow}), f_i(\text{raven}), f_i(\text{cat}), f_i(\text{cheetah})\big) = 5$

6.2.3 Properties.

With this definition, the analogical dissimilarity has the following properties :

Property 6.1 (Properties of AD in sets of features.)

1. $\forall u, v, w, x \in X, AD(u, v, w, x) = 0 \Leftrightarrow u : v :: w : x$
2. $\forall u, v, w, x \in X, AD(u, v, w, x) = AD(w, x, u, v) = AD(v, u, x, w) =$
3. $\forall u, v, w, x, z, t \in X, AD(u, v, z, t) \leq AD(u, v, w, x) + AD(w, x, z, t)$
4. *In general,* $\forall (u, v, w, x) \in X^4 : AD(u, v, w, x) \neq AD(v, u, w, x)$

From the first two properties, we deduce that:
$AD(u, v, w, x) = AD(w, x, u, v) = AD(v, u, x, w) = AD(u, w, v, x) =$
$AD(x, v, u, w) = AD(x, w, v, u) = AD(v, x, u, w) = AD(w, u, x, v)$

The first properties are quite straightforward from the definition. The demonstration of the third one is simple as well. If the property

$$\begin{aligned} AD(f_i(u), f_i(v), f_i(z), f_i(t)) \quad \leq \quad & AD(f_i(u), f_i(v), f_i(w), f_i(x)) \\ & + AD(f_i(w), f_i(x), f_i(z), f_i(t)) \end{aligned}$$

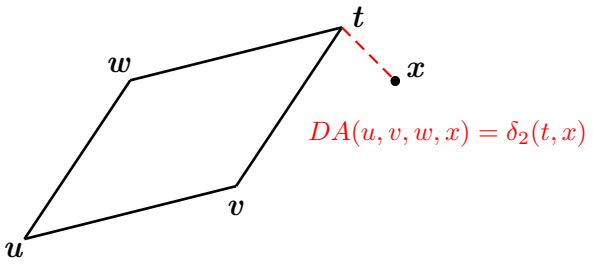

Figure 5: Analogical dissimilarity in vector spaces with distance δ_2.

holds true for every 6-uple of elements and every feature f_i, then property (3) is true.

This is easy to verify when all elementary values are equal to 1. If we choose the value 2 for the special cases, the property is still true, the demonstration being done by examining all possible cases: it is impossible to find 6 binary features a, b, c, d, e, f such that $AD(a, b, e, f) = 2$ and $AD(a, b, c, d) + AD(c, d, e, f) < 2$. More precisely, if $AD(a, b, e, f) = 2$, $AD(a, b, c, d) + AD(c, d, e, f)$ is also equal to 2 for all the four values that (c, d) can take.

6.3 A definition in $\mathbb{R}^n$.

6.3.1 Definition.

The analogical dissemblance between four vectors must reflect in some way how far they are from constructing a parallelogram. When four vectors u, v, w and x are in analogical relation (i.e., construct a parallelogram) with opposite sides $\overrightarrow{uv}$ and $\overrightarrow{wx}$ if and only if $\overrightarrow{Ou} + \overrightarrow{Ox} = \overrightarrow{Ov} + \overrightarrow{Ow}$, or equivalently $u + x = v + w$, we have chosen the following definition (see Figure 5):

Definition 6.3 (Analogical dissimilarity between vectors.) *The analogical dissimilarity between four vectors u, v, w and x of $\mathbb{R}^n$ in which is defined the norm $\| \|_p$ is given by the real positive value $AD(u, v, w, x) = \delta_p(u + x, v + w) = \|(u + x) - (v + w)\|_p$. It is also equal to $\delta_p(t, x)$, where t is the solution of the analogical equation $u : v :: w : t$.*

6.3.2 Properties.

We have the following properties :

Property 6.2 (Properties of AD between vectors.)

1. $\forall u, v, w, x \in \mathbb{R}^n, AD(u, v, w, x) = 0 \Leftrightarrow u : v :: w : x$

2. $\forall u, v, w, x \in \mathbb{R}^n, AD(u, v, w, x) = AD(w, x, u, v) = AD(u, w, v, x)$

3. $\forall u, v, w, x, z, t \in \mathbb{R}^n, AD(u, v, z, t) \leq AD(u, v, w, x) + AD(w, x, z, t)$

4. In general, $\forall (u, v, w, x) \in \mathbb{R}^n : AD(u, v, w, x) \neq AD(v, u, w, x)$

From the first two properties, we deduce that:
$AD(u, v, w, x) = AD(w, x, u, v) = AD(v, u, x, w) = AD(u, w, v, x) =$
$AD(x, v, u, w) = AD(x, w, v, u) = AD(v, x, u, w) = AD(w, u, x, v)$
The first two properties are quite straightforward from the definition. The third deserves a demonstration.

We want to prove that:

$$\delta_p(u + x, v + w) + \delta_p(w + t, x + z) \geq \delta_p(u + t, v + z)$$

We rewrite the third term using the translation property:

$$\delta_p(u + t, v + z) = \|(u + t) - (v + z)\|_p = \|(u + t + w + x) - (v + z + w + x)\|_p$$

$$= \|(u + x) - (v + w) + (w + t) - (x + z)\|_p$$

To simplify, we denote:

- $a = (u + x) - (v + w)$
- $b = (w + t) - (x + z)$

And then we have:

$$AD(u, v, z, t) = \|a + b\|_p$$

Since $\| \ \|_p$ is a norm, it respect the triangle inequality:

$$\|a + b\|_p \leq \|a\|_p + \|b\|_p$$

Giving back their values to a and b, we get:

$$\|(u + x) - (w + v) + (w + t) - (x + z)\|_p \leq \|(u + x) - (v + w)\|_p + \|(w + t) - (x + z)\|_p$$

and finally:

$$\|(u + t) - (v + z)\|_p \leq \|(u + x) - (v + w)\|_p + \|(w + t) - (x + z)\|_p$$

$$AD(u, v, z, t) \leq AD(u, v, w, x) + AD(w, x, z, t)$$

6.4 A definition in a cyclic group.

In a cyclic group, we can define the analogical dissimilarity almost in the same way than in vector spaces. Let u, v, w and x be four elements of a cyclic group X. We know that there exists an unique element $t \in X$ such that $u : v :: w : t$ and we have defined a distance δ coherent with the analogy on X. The natural definition of AD is as follows:

Definition 6.4 (Analogical dissimilarity in a cyclic group.) *The analogical dissimilarity between four elements u, v, w and x of a cyclic group X in which is defined a distance δ coherent with the analogy on X is given by:*

$$AD(u, v, w, x) = \delta(t, x)$$

where t is the solution of the analogical equation $u : v :: w : t$.

All the required properties are satisfied:

Property 6.3 (Properties of AD in cyclic groups.)

1. $\forall u, v, w, x \in X, AD(u, v, w, x) = 0 \Leftrightarrow u : v :: w : x$
2. $\forall u, v, w, x \in X, AD(u, v, w, x) = AD(w, x, u, v) = AD(u, w, v, x)$
3. $\forall u, v, w, x, z, t \in X, AD(u, v, z, t) \leq AD(u, v, w, x) + AD(w, x, z, t)$
4. *In general,* $\forall (u, v, w, x) \in X : AD(u, v, w, x) \neq AD(v, u, w, x)$

From the two first properties, we deduce that:
$AD(u, v, w, x) = AD(w, x, u, v) = AD(v, u, x, w) = AD(u, w, v, x) =$
$AD(x, v, u, w) = AD(x, w, v, u) = AD(v, x, u, w) = AD(w, u, x, v)$

The demonstration of these properties are easy, relying on the definition of a distance coherent with analogy.

6.5 Comments on defining an analogical dissimilarity in a metric space.

We have not succeeded in defining an analogical dissimilarity in metric spaces, i.e. spaces where a distance is defined, but with no other structure (as a group or a vector space).

We have proven that the following tentative definitions of AD:

$$AD(a, b, c, d) = |\delta(a, b) - \delta(c, d)| + |\delta(a, c) - \delta(b, d)|$$

and

$$AD(a, b, c, d) = \left((\delta(a, b) - \delta(c, d))^2 + (\delta(a, c) - \delta(b, d))^2\right)^{\frac{1}{2}}$$

are only such that $a : b :: c : d \Rightarrow AD(a, b, c, d) = 0$, and have not in general the triangle inequality.

On the other hand, defining AD as:

$$AD(a,b,c,d) = Min \begin{cases} \delta(a,b) + \delta(c,d) \\ \delta(a,c) + \delta(b,d) \end{cases}$$

insures the triangle inequality:

$$\Delta(S_{a\to b}, S_{c\to d}) + \Delta(S_{c\to d}, S_{e\to f}) \geq \Delta(S_{a\to b}, S_{e\to f})$$

with the same weak first property: $\Delta(S_{a\to b}, S_{c\to d}) = 0 \Rightarrow a : b :: c : d$.

This is not important as far as the problem of supervised learning by analogy is not concerned. But fast algorithm can only be applied if AD has all the properties that we require. This will be developed in section 8.

7 Analogical dissimilarity between sequences.

We know from section 5.2 that there exists two equivalent algorithms to compute the solution of an analogical equation on sequences with a dynamic programming technique, when there is an analogy on the augmented alphabet Σ' and a distance δ coherent with this analogy. We recall that SEQUANA1 computes the optimal alignments $\mathcal{S}(u,v)$ between the sequences u and v on the one hand, and $\mathcal{S}(w,x)$ between u and w on the other hand, and then deduce x from these two sets of alignments. For this purpose, a distance Δ must be defined on couples of transformations between letters of Σ', with the only constraint on Δ that $\Delta(S_{a\to b}, S_{c\to d})$ is null if and only if $a : b :: c : d$ holds true in Σ' and that δ, the distance on Σ', is coherent with the analogy.

SEQUANA2 is a direct method, progressing from left to right in the three sequences u, v and w and computing x during this progression, in a three-dimensional dynamic programming process. SEQUANA2 gives the same result than SEQUANA1 and does not requires to defining a distance Δ. We will show that the two algorithm give the same results and we will compare their time complexity.

We present in the following two algorithms for computing the analogical dissimilarity between four sequences of $\Sigma^\star$, based on the same ideas than SEQUANA1 and SEQUANA2. The difference is that they do not produce the same result, and that the second one gives solutions with better properties, especially with respect to the use of the notion of analogical dissimilarity in learning that we will develop in section 8.

7.1 A first definition.

Since we work on alphabets on which we already know how to define an analogical dissimilarity, a straightforward manner to choose Δ is to define: $\Delta(S_{a\to b}, S_{c\to d}) = AD(a,b,c,d)$. We know that $\Delta(S_{a\to b}, S_{c\to d})$ is null if and only if $a : b :: c : d$ holds true in Σ' and that δ is coherent with the analogy on Σ. Hence, $AD(u,v,w,x) = 0$ if and only if the four sequences are in analogy, according with our definition.

Moreover, the properties that we have shown on the three studied alphabets insure that Δ, defined as $\Delta(S_{a\to b}, S_{c\to d}) = AD(a,b,c,d)$ is a distance on $\mathbb{S}'^{\star}$.

We give two definitions, with the same principles that in section 5.2, but they will prove to be not equivalent: the complexity of computing an analogical dissimilarity between sequences cannot be decreased through a two steps process, while at the contrary *SEQUANA2* can be replaced by *SEQUANA1*.

7.1.1 Approximate analogical dissimilarity.

Definition 7.1 (Approximate analogical dissimilarity between sequences.) *Let $\mathcal{S}(u,v)$ be the optimal sequence of transformation obtained with distance δ between the sentences u and v in $\Sigma'^{\star}$. Similarly, let $\mathcal{S}(w,x)$ be the optimal sequence of transformation obtained with distance δ between the sentences w and x in $\Sigma'^{\star}$. We call* approximative analogical dissimilarity*, denoted $\widehat{AD}(u,v,w,x)$, between the sequences u, v, w et x, the optimal editing cost obtained with distance Δ between $\mathcal{S}(u,v)$ and $\mathcal{S}(w,x)$, where $\Delta(S_{a\to b}, S_{c\to d}) = AD(a,b,c,d)$ in Σ'.*

7.1.2 An example.

Let the alphabet be composed of four vectors in $\mathbb{R}^3$, with δ the euclidian distance.

	x_1	x_2	x_3
a	0	0	0
b	3	0	0
c	0	4	0
$\backsim$	1.5	2	$\frac{\sqrt{119}}{2}$

Then δ is computed as:

δ	a	b	c	$\backsim$
a	0	3	4	6
b	3	0	5	6
c	4	5	0	6
$\backsim$	6	6	6	

There is an unique optimal alignment between $u = cbacba$ and $v = babba$ (with cost 11):

$$\begin{array}{ccccccccc} u & = & c & b & a & c & b & a \\ & & | & | & | & | & | & | \\ v & = & \backsim & b & a & b & b & a \end{array}$$

There is also an unique optimal alignment between $w = cbacbc$ and $x = bcabbc$ (with cost 15):

$$\begin{array}{ccccccccc} w & = & c & b & a & c & b & c \\ & & | & | & | & | & | & | \\ x & = & b & c & a & b & b & c \end{array}$$

Finally, the approximate analogical dissimilarity is computed, with a value of 11, by the following alignment according to Δ:

$$\begin{array}{lcccccccc} \mathcal{S}(u,v) & = & S_{c\to\smile} & & S_{b\to b} & S_{a\to a} & S_{c\to b} & S_{b\to b} & S_{a\to a} \\ & & | & & | & | & | & | & | \\ \mathcal{S}(w,x) & = & S_{c\to b} & & S_{b\to c} & S_{a\to a} & S_{c\to b} & S_{b\to b} & S_{c\to c} \end{array}$$

This algorithms runs in $\mathcal{O}\big((|u|.|v|)+(|w|.|x|)+(|u|+|v|).(|w|+|x|)\big)$, but $\widehat{AD}$ has only the property of coherence with analogy. Simple counter-examples can be found to show that triangle inequality is generally false. We look now for a stronger definition.

7.2 A better definition.

7.2.1 Analogical dissimilarity between sequences.

Definition 7.2 (Analogical dissimilarity between sequences.) *Let Σ be an alphabet and Σ' the augmented alphabet on which there exists an analogical dissimilarity $AD(a,b,c,d)$ between the letters of Σ'. Let the cost of an alignment of the four sequences be the sum of the analogical dissimilarities of the 4-uples of aligned letters in Σ'. We define the* analogical dissimilarity *$AD(u,v,w,x)$ of four sequences in $\Sigma^\star$ as the cost of the alignment of minimal cost of the four sequences.*

7.2.2 Properties.

This definition ensures that the following properties hold true:

Property 7.1

1. $\forall u,v,w,x \in (\Sigma^\star)^4, AD(u,v,w,x)=0 \Leftrightarrow u:v::w:x$
2. $\forall (u,v,w,x) \in (\Sigma^\star)^4, AD(u,v,w,x)=AD(w,x,u,v)=AD(u,w,v,x)$
3. $\forall (u,v,w,x,z,t) \in (\Sigma^\star)^6, AD(u,v,w,x) \leq AD(u,v,z,t)+AD(z,t,w,x)$
4. *In general,* $\forall (u,v,w,x) \in (\Sigma^\star)^4$, $AD(u,v,w,x) \neq AD(v,u,w,x)$

From the two first properties, we deduce that:
$AD(u,v,w,x)=AD(w,x,u,v)=AD(v,u,x,w)=AD(u,w,v,x)=$
$AD(x,v,u,w)=AD(x,w,v,u)=AD(v,x,u,w)=AD(w,u,x,v)$
The first three properties can be formulated a bit differently:

Property 7.2 $D\big((u,v),(w,x)\big)=AD(u,v,w,x)$ *is a distance on* $(\Sigma^\star)^2$

We will not use explicitly the distance D in the sequel, but this property will be useful for ensuring the validity of algorithm FADANA, in section 8.4.6.

7.2.3 Algorithm.

We can compute $AD(u, v, w, x)$ with a dynamic programming algorithm, that progresses in synchrony in the four sequences to build the optimal alignment according to Δ, where:

$$\Delta(S_{a\to b}, S_{c\to d}) = AD(a, b, c, d)$$

The input of this algorithm is an augmented alphabet Σ' on which there exists an analogical relation, a distance δ coherent with the analogy and an analogical dissimilarity $AD(a, b, c, d)$ between the letters of Σ'. The output is the analogical dissimilarity between four sentences of $\Sigma^\star$, namely $AD(u, v, w, x)$, also equal to $AD(w, x, u, v)$, $AD(u, w, v, x)$, $AD(v, u, x, w)$, à $AD(x, v, w, u)$, $AD(x, w, v, u)$, $AD(v, x, w, u)$, $AD(w, u, x, v)$ and $AD(x, v, w, u)$.

The recurrence is as follows: :

Initialisation

$C^{u_0v_0}_{w_0x_0} \leftarrow 0$;

for $i = 1, |u|$ **do** $C^{u_iv_0}_{w_0x_0} \leftarrow \sum_{k=1}^{k=i} \Delta(S_{u_i\to\backsim}, S_{\backsim\to\backsim})$ **done** ;

for $j = 1, |v|$ **do** $C^{u_0v_j}_{w_0x_0} \leftarrow \sum_{k=1}^{k=j} \Delta(S_{\backsim\to v_j}, S_{\backsim\to\backsim})$ **done** ;

for $i = 1, |w|$ **do** $C^{u_0v_0}_{w_kx_0} \leftarrow \sum_{i=1}^{i=k} \Delta(S_{\backsim\to\backsim}, S_{w_k\to\backsim})$ **done** ;

for $j = 1, |x|$ **do** $C^{u_0v_0}_{w_0x_l} \leftarrow \sum_{k=1}^{k=l} \Delta(S_{\backsim\to\backsim}, S_{\backsim\to x(l)})$ **done** ;

Recurrence

$$C^{u_iv_j}_{w_kx_l} = Min \begin{cases} C^{u_{i-1}v_{j-1}}_{w_{k-1}x_{l-1}} + \Delta(S_{u_i\to v_j}, S_{w_k\to x_l}) & u_i : v_j :: w_k : x_l \\ C^{u_{i-1}v_{j-1}}_{w_{k-1}x_l} + \Delta(S_{u_i\to v_j}, S_{w_k\to\backsim}) & u_i : v_j :: w_k :\backsim \\ C^{u_{i-1}v_{j-1}}_{w_kx_{l-1}} + \Delta(S_{u_i\to v_j}, S_{\backsim\to x_l}) & u_i : v_j ::\backsim: x_l \\ C^{u_{i-1}v_{j-1}}_{w_kx_l} + \Delta(S_{u_i\to v_j}, S_{\backsim\to\backsim}) & u_i : v_j ::\backsim:\backsim \\ C^{u_iv_{j-1}}_{w_{k-1}x_{l-1}} + \Delta(S_{\backsim\to v_j}, S_{w_k\to x_l}) & \backsim: v_j :: w_k : x_l \\ C^{u_iv_{j-1}}_{w_kx_{l-1}} + \Delta(S_{\backsim\to v_j}, S_{\backsim\to x_l}) & \backsim: v_j ::\backsim: x_l \\ C^{u_iv_{j-1}}_{w_{k-1}x_l} + \Delta(S_{\backsim\to v_j}, S_{w_k\to\backsim}) & \backsim: v_j :: w_k :\backsim \\ C^{u_iv_{j-1}}_{w_kx_l} + \Delta(S_{\backsim\to v_j}, S_{\backsim\to\backsim}) & \backsim: v_j ::\backsim:\backsim \\ C^{u_{i-1}v_j}_{w_{k-1}x_{l-1}} + \Delta(S_{u_i\to\backsim}, S_{w_k\to x_l}) & u_i :\backsim:: w_k : x_l \\ C^{u_{i-1}v_j}_{w_kx_{l-1}} + \Delta(S_{u_i\to\backsim}, S_{\backsim\to x_l}) & u_i :\backsim::\backsim: x_l \\ C^{u_{i-1}v_j}_{w_{k-1}x_l} + \Delta(S_{u_i\to\backsim}, S_{w_k\to\backsim}) & u_i :\backsim:: w_k :\backsim \\ C^{u_{i-1}v_j}_{w_kx_l} + \Delta(S_{u_i\to\backsim}, S_{\backsim\to\backsim}) & u_i :\backsim::\backsim:\backsim \\ C^{u_iv_j}_{w_{k-1}x_{l-1}} + \Delta(S_{\backsim\to-}, S_{w_k\to x_l}) & \backsim:\backsim:: w_k : x_l \\ C^{u_iv_j}_{w_kx_{l-1}} + \Delta(S_{\backsim\to\backsim}, S_{\backsim\to x_l}) & \backsim:\backsim::\backsim: x_l \\ C^{u_iv_j}_{w_{k-1}x_l} + \Delta(S_{\backsim\to\backsim}, S_{w_k\to\backsim}) & \backsim:\backsim:: w_k :\backsim \end{cases}$$

End

When $i = |u|$ and $j = |v|$ and $k = |w|$ and $l = |x|$.

Result
$C^{u_{|u|}v_{|v|}}_{w_{|w|}x_{|x|}}$ is $AD(u,v,w,x)$ in $\Sigma^\star$.
Complexity
This algorithms runs in a time complexity in $\mathcal{O}\big(|u|.|v|.|w|.|x|\big)$.

7.3 The two dissimilarities are not the same.

Let us come back to the example of section 7.1.2 : $\widehat{AD}(cbacba, babba, cbacbc, bcabbc) = 11$. We can show that the construction of $\widehat{AD}$ does not produce an alignment of lowest cost. There actually exists two couples of alignments between u and v, one the one hand and w and x, on the other hand, that provide a better result than 11. The first one is not optimal (with cost 16), the second is the optimal alignment (with cost 15).

$$\begin{array}{ccccccccc}
u & = & c & b & a & c & b & a \\
 & & | & | & | & | & | & | \\
v & = & b & \backsim & a & b & b & a \\
 & & & & & & & \\
w & = & c & b & a & c & b & c \\
 & & | & | & | & | & | & | \\
x & = & b & c & a & b & b & c
\end{array}$$

Optimally aligning these two alignments with distance Δ gives a cost of only 6:

$$\begin{array}{cccccc}
S_{c\to b} & S_{b\to\backsim} & S_{a\to a} & S_{c\to b} & S_{b\to b} & S_{a\to a} \\
| & | & | & | & | & | \\
S_{c\to b} & S_{b\to c} & S_{a\to a} & S_{c\to b} & S_{b\to b} & S_{c\to c}
\end{array}$$

Hence, $AD(cbacba, babba, cbacbc, bcabbc) < \widehat{AD}(cbacba, babba, cbacbc, bcabbc)$. This inequality is generally strict for all examples, except that we know that:

$$\widehat{AD}(u,v,w,x) = 0 \Leftrightarrow AD(u,v,w,x) = 0$$

7.4 Experiments on the analogical dissimilarity between sequences.

We have designed a small experiment to test the practical coherence of the notion of analogical dissimilarity on sequences. The basic idea is firstly to build a set of sequences $\mathcal{S}$, in which the analogies are known by construction; secondly to select one of the sequences (say s) and to blur it by some controlled noise; thirdly to search the triplet of sequences in $\mathcal{S} - \{s\}$ that has the least AD with s. If the noise is null or weak, s must stay in (approximate) analogy with the same triplet of sentences. When the noise increases, the best triple in $\mathcal{S} - \{s\}$ will change. We describe hereafter in more details the experimental process and give some results.

7.4.1 Constructing the set $\mathcal{S}$.

The sentences are built on an alphabet Σ' of $2n+1$ letters, which is defined by $n+2$ binary features. For example, when $n=3$, the alphabet is defined as follows:

	f_1	f_2	f_3	f_4	f_5
a	1	0	0	1	0
b	0	1	0	1	0
c	0	0	1	1	0
A	1	0	0	0	1
B	0	1	0	0	1
C	0	0	1	0	1
$\backsim$	0	0	0	0	0

The first three features indicates what is the letter (for example, f_1 is true on a and A only) and the last two indicate the case of the letter (f_4 holds true for lower case letters, f_5 for upper case letters).

This gives the following distance table on Σ':

	a	b	c	A	B	C	$\backsim$
a	0	2	2	1	3	3	2
b	2	0	2	3	1	3	2
c	2	2	0	3	3	1	2
A	1	3	3	0	2	2	2
B	3	1	3	2	0	2	2
C	3	3	1	2	2	0	2
$\backsim$	2	2	2	2	2	2	

We have chosen an alphabet of 11 letters ($n=5$).

$\mathcal{S}$ has been built in three exemplaries, each composed of 400 sequences of same length $m=6$, 8 and 10. The sequences are built four by four. We randomly draw four sequences X, Y, Z et T of length $m/2$, which are concatenated to produce $u=XZ$, $v=XT$, $w=YZ$, $x=YT$. We know by construction that these four new sentences are in exact analogy (not only in our definition, but also in the "algebraic" analogy of section 5.1). We check that in $\mathcal{S}$ these 100 analogies are the only ones.

7.4.2 Running the experiment.

To follow, we extract a sentence s from $\mathcal{S}$ and add it some noise controlled by a value τ, between 0 and 1. Basically, the idea is to take each letter of s and to delete it, or insert a randomly chosen new letter before it, or change it to a randomly chosen other letter. The total probability of doing one of these operation on each letter is τ. The distribution of the probabilities between the insertion of a letter, the deletion and the modification is derived from the distance table. For example, the following noisy sequences have been created:

Original sequence	aAbdDD	adEdebbEaa	AdAcbBec
$\tau = 0.1$	aAEdDD	adEdebbEaa	AdAcBec
$\tau = 0.2$	aAbdDD	adEdebbEaA	AdAcbbec
$\tau = 0.3$	AAbdDD	aadEdBbbCeaa	adacDBcc

The experiment is done with every fourth sentence s in $\mathcal{S}$. Every time the original triple is found to be the least dissimilar, we add 1 to the score. The best score, which occurs when $\tau = 0$, is therefore 100, and it decreases when τ increases.

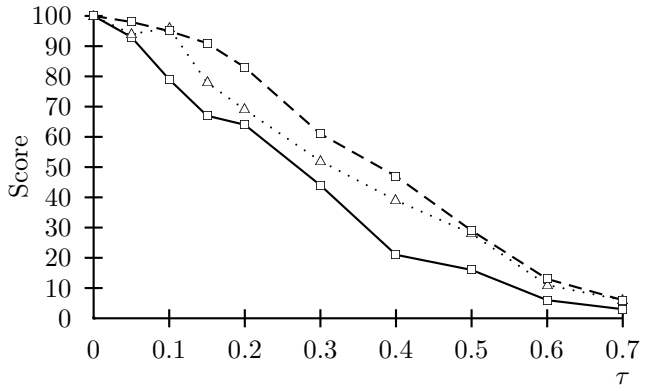

Figure 6: Variation of the score against the level of noise. $\mathcal{S}$ is composed of 100 sentences on an alphabet of 11 letters, either of length 10 (dashed line), or 8 (dotted line) or 6 (solid line)

7.4.3 Conclusion.

This experiment depends on several parameters that we have explored in more detail in [21], but with a different tentative definition of AD. We give only here a characteristic curve. It does not seem to show that the analogical dissimilarity is very resistant to the introduction of noise. Nevertheless, it provides a flexible and coherent measure of analogy, without which very little could be done in machine learning experiments on real data.

8 Analogical dissimilarity and machine learning.

8.1 Motivation.

We assume here that an analogy is defined on the set X and there exists an analogical AD dissimilarity with the following properties:

Coherence with analogy.

$\forall (u, v, w, x) \in X^4 : AD(u, v, w, x) = 0 \Leftrightarrow \quad u : v :: w : x$

Symmetry for "as". $\forall (u,v,w,x) \in (\Sigma^\star)^4 : AD(u,v,w,x) = AD(w,x,u,v)$

Triangle inequality.
$\forall (u,v,w,x,z,t) \in (\Sigma^\star)^6 : AD(u,v,w,x) \leq AD(u,v,z,t) + AD(z,t,w,x)$

Exchange of medians. $\forall (u,v,w,x) \in (\Sigma^\star)^4 : AD(u,v,w,x) = AD(u,w,v,x)$

Non symmetry for "is to".
In general, $\forall (u,v,w,x) \in (\Sigma^\star)^4 : AD(u,v,w,x) \neq AD(v,u,w,x)$

Let $\mathcal{S}$ be a set of elements of X, which is of cardinal m, and let y be another element of X with $y \notin \mathcal{S}$. The problem that we tackle in this section is to find the triple of objects (u,v,w) in $\mathcal{S}$ such that:

$$AD(u,v,w,y) = \underset{t_1,t_2,t_3 \in \mathcal{S}}{Argmin}\ AD(t_1,t_2,t_3,y)$$

8.2 The brute force solution.

An obvious solution is to examine all the triples in $\mathcal{S}$. This brute force method requires m^3 calls to a procedure computing the analogical dissimilarity between four objects of X. According to the properties of analogical dissimilarity, this number can actually be divided by 8, but it does not change the theoretical and practical complexity of the search.

8.3 Fast nearest neighbor search: the AESA algorithm.

8.3.1 The principle.

AESA ([20]) is an efficient pruning technique for searching, according to some distance d, the nearest neighbor of an object $y \in X$ among some set $\mathcal{S}$ of objects in X. Being a distance, d is an application $X \times X \to \mathbb{R}$ with verifies in particular the triangle inequality.

AESA uses this property to reduce the computation cost, which is basically of $m = Card(\mathcal{S})$ distances, with the following procedure:

1. **Initialise.** U starts as $\mathcal{S}$.
2. **Select.** Select $p_0 \in U$.
3. **Compute one distance.** Compute distance $d(p_0, y)$ between p_0 and y.
4. **Update the nearest neighbor.** If p_0 is the nearest neighbor so far, update the result.
5. **Eliminate.** Use p_0 to eliminate candidates from U.
6. **Iterate.** Repeat items 2 to 5 while U is not empty.

The efficiency of this algorithm is based on phases 2 and 5. The latter uses the properties of distance d in the following manner.

8.3.2 Elimination.

Let the nearest neighbor of y found so far in $\mathcal{S}$ be pp, with $d(y, pp) = \delta$.

Among the elements of U, all the following ones have no chance to be the nearest neighbor of y and thus can be eliminated from U:

- Those which are located inside the hypersphere centered in p_0 with radius $d(y, p_0) - \delta$, like p_1 in Figure 7.
- Those which are located outside the hypersphere centered in p_0 with radius $d(y, p_0) + \delta$, like p_2 in Figure 7.

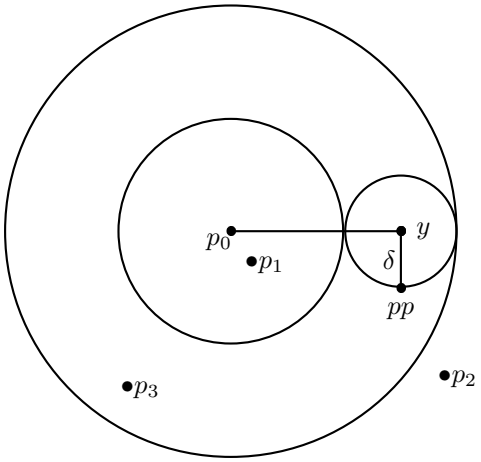

Figure 7: The geometry of "eliminate" in the AESA algorithm.

8.3.3 Selection.

The problem is now to select a element p_0 as efficient as possible, i.e. as close as possible of the actual nearest neighbor of y. For this purpose, the element chosen in U is heuristically taken close to the intersection of all hyperspheres centered on an element $u \in Q$ with radius $d(x, u)$, where Q is the subset of $\mathcal{S}$ composed of all the elements for which $d(x, u)$ has already been computed.

$$p_0 = \underset{q \in U}{Argmin} \ \underset{u \in Q}{Max} \left| \ d(q, u) - d(x, u) \ \right|$$

8.3.4 Reducing the precomputation: *LAESA*.

Principle. The *select* procedure uses terms such as $d(p_0, p_1)$, $d(p_0, p_2)$, etc., which are distances between elements of $\mathcal{S}$. If we want the *eliminate* phase to be efficient, these values have to computed and stored with $\mathcal{S}$. This gives a precomputation time and storage complexity in $\mathcal{O}(m^2)$.

A further version of *AESA* reduces this complexity to $\mathcal{O}(m)$, while experimentally loosing few in efficiency in the *eliminate* phase. This version is called *LAESA* ([20]).

It consists in choosing a subset $\mathcal{T}$ of n elements in $\mathcal{S}$, called *base prototypes*, and precompute only the $n \times m$ distances between the elements of $\mathcal{T}$ and those of $\mathcal{S}$. Then, in the simplest version, the algorithm is described as algorithm 4, in this section, where:

- $G(p)$ is a lower bound of $d(p, y)$
- $p^\star$ is the nearest neighbor of y at distance $d^\star = d(p^\star, y)$

Choosing $\mathcal{T}$. The authors of the *LAESA* algorithm recommend to compose $\mathcal{T}$ with base prototypes maximally separated by the distance d, and suggest a greedy procedure to choose them among $\mathcal{S}$. They experimentally show that there is an optimal size for $\mathcal{T}$, which depends on the dimension of the euclidian space in which they work, but not on the size of $\mathcal{S}$, once the dimension is fixed. They show that the number of distance computations with *LAESA* is about 1.5 times that of *AESA*, while the space and time preprocessing requirements are only linear in m. To give a figure, in $\mathbb{R}^8$, when $m = 1024$ vectors are randomly chosen inside an hypersphere of radius 1 to compose $\mathcal{S}$, the optimal size of $\mathcal{T}$ is around 25 and the number of euclidian distance computations is in average around 50 with LAESA.

8.4 "FADANA": FAst search of the least Dissimilar ANAlogy .

This section describes a fast algorithm to find, given a set of objects $\mathcal{S}$ of cardinal m and an object y, the three objects $(z^\star, t^\star, x^\star)$ in $\mathcal{S}$ such that the analogical dissimilarity $AD(z^\star, t^\star, x^\star, y)$ is minimal. It is based on the AESA technique, which can be extended to analogical dissimilarity thanks to property 7.2. Hence, since a analogical dissimilarity $AD(z, t, x, y)$ can be seen as a distance between the two couples (z, t) and (x, y), we will mainly work on couples of objects. We use equivalently in the sequel the terms "(analogical) distance between the two couples (u, v) and (w, x)" and "(analogical) dissimilarity between the four elements u, v, w and x" to describe $AD(u, v, w, x)$.

8.4.1 Preliminary computation.

In this part, which is done off line, we have to compute the analogical dissimilarity between every four objects in the data base. This step requires a complexity in time and space of $\mathcal{O}(m^4)$, where m is the size of $\mathcal{S}$. We will come back on that point in section 8.5, where we will progress from an *AESA*-like to a *LAESA*-like technique.

Algorithm 4 Algorithm LAESA.

begin
$U \leftarrow \mathcal{S}$; $V \leftarrow \mathcal{T}$;
$p^\star \leftarrow anything$; $d^\star \leftarrow +\infty$
for each element p of U **do**
  $G(p) \leftarrow +\infty$
end for
while $U \neq \emptyset$ **do**
  if $V \neq \emptyset$ **then**
    select the element s in V with the lowest $G(p)$
  else
    select an element s in U
  end if
  compute $d(s, y)$
  if $d(s, y) < d^\star$ **then**
    $d(s, y) \leftarrow d^\star$; $p^\star \leftarrow s$
  end if
  if $V \neq \emptyset$ **then**
    for each element in V **do**
      $G(p) \leftarrow Min \begin{cases} G(p) \\ d(p, s) - d(y, s) \end{cases}$
    end for
  end if
  for each element of U **do**
    if $G(p) \geq d^\star$ **then**
      $U \leftarrow U - \{p\}$
    end if
    if $(p \in V)$ and $(G(p) \geq d^\star)$ **then**
      $V \leftarrow V - \{p\}$
    end if
  end for
end while
end

8.4.2 Principle of the algorithm.

The basic operation is to compose a couple of objects by adding to y an object $x_i \in \mathcal{S}$ where $i = 1, m$. The goal is now to find the couple of objects in $\mathcal{S}$ having the lowest distance with (x_i, y), then to change x_i into x_{i+1}. Looping m times on an *AESA*-like select and eliminate technique insures to finally find the triple in $\mathcal{S}$ having the lowest analogical dissimilarity when associated with y.

8.4.3 Notations.

Let us denote :

- $\mathcal{C}$ the set of couples (u, v) which distance to (x_i, y) has already been computed.
- $\delta = \underset{(z,t)\in\mathcal{U}}{ArgMin}(AD(z, t, x_i, y))$
- $\delta_i = \underset{(z,t)\in\mathcal{U}, 1\leq j\leq i}{ArgMin}(AD(z, t, x_i, y))$
- $Dist = \{AD(z, t, x_i, y), (z, t) \in \mathcal{C}\}$
- $Dist(j)$ the j^{th} element of $Dist$
- $Quad_{\mathcal{U}} = \{(z, t, x_i, y), (z, t) \in \mathcal{C}\}$
- $Quad_{\mathcal{U}}(j)$ the j^{th} element of $Quad_{\mathcal{U}}$

The algorithm is constructed in three phases :

8.4.4 Initialization

Each time that x_i changes (when i is increased by 1), the set $\mathcal{U}$ is refilled with all the possible couples of objects $\in \mathcal{S}$.

The set $\mathcal{C}$ and $Dist$ which contains respectively the couples and the distances to (x_i, y) that have been measured during one loop, are initialized as empty sets.

The local minimum Min, containing the minimum of analogical dissimilarities of one loop is set to infinity.

$k = Card(\mathcal{C})$ represents the number of couples where the distance have been computed with (x_i, y) in the current loop is set to zero.

8.4.5 Selection

The goal of this function is to extract from the set $\mathcal{U}$ the couple (zz, tt) that is the more promising in terms of the minimum analogical distance with (x_i, y), using the criterion :

$$(zz, tt) = \underset{(u,v)\in\mathcal{U}}{Argmin} \underset{(z,t)\in C}{Max} \mid AD(u, v, z, t) - AD(z, t, x_i, y) \mid$$

Algorithm 5 Algorithm FADANA: initialization.

begin
$\mathcal{U} \leftarrow \{(x_i, x_j),\ i = 1, m \text{ and } j = 1, m\}$;
$\mathcal{C} \leftarrow \varnothing$;
$Min \leftarrow +\infty$;
$Dist \leftarrow \varnothing$;
$k \leftarrow 0$;
end

Algorithm 6 Algorithm FADANA: selection of the more promising couple.

`selection`$(\mathcal{U}, \mathcal{C}, (x_i, y), Dist)$
begin
$s \leftarrow 0$
for $i = 1, Card(\mathcal{U})$ **do**
  if $s \leq \sum_{j \in \mathcal{C}} |AD(z_j,\ t_j,\ u_i,\ v_i) - Dist(j)|$ **then**
    $s \leftarrow \sum_{j \in \mathcal{C}} |AD(z_j,\ t_j,\ u_i,\ v_i) - Dist(j)|$;
    $Argmin \leftarrow i$;
  end if
end for
Return $(u_{Argmin},\ v_{Argmin})$;
end

8.4.6 Elimination

During this section all the couples $(u, v) \in \mathcal{U}$ where the analogical distance with (x_i, y) can not be less than what we already found are eliminated.

The use of the two criteria below, thanks to the properties of analogy (see section 8.1), allows to find those couples (u, v) such that:

$$AD(u, v, z, t) \leq AD(z, t, y, x_i) - \delta \Rightarrow AD(u, v, x_i, y) \geq \delta$$

and

$$AD(u, v, z, t) \geq AD(z, t, y, x_i) + \delta \Rightarrow AD(u, v, x_i, y) \geq \delta$$

where $\delta = AD(z^\star, t^\star, x^\star, y)$ represents the minimum analogical dissimilarity found until now. Note that δ is updated during the whole algorithm and is not changed when i is increased.

8.5 Selection of base prototypes in *FADANA*.

So far, *FADANA* is directly derived from *AESA* and has the drawback of requiring a pre-computing time and storage in $\mathcal{O}(m^4)$, which is in practice useless for $m > 100$. We give experiments with this algorithm in the next section.

Algorithm 7 Algorithm FADANA: elimination of the useless couples.

`eliminate`$(\mathcal{U}, \mathcal{C}, (x_i, y), \delta, k)$
$(z_k,\ t_k)$ is the k^{th} element of $Quad_{\mathcal{U}}$
begin
for $i = 1, Card(\mathcal{U})$ **do**
  if $AD(z_k,\ t_k,\ u_i,\ v_i) \leq Dist(k) + \delta$ **then**
    $\mathcal{U} \leftarrow \mathcal{U} - \{(u_i,\ v_i)\}$;
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{(u_i,\ v_i)\}$;
  else if $AD(z_k,\ t_k,\ u_i,\ v_i) \geq Dist(k) - \delta$ **then**
    $\mathcal{U} \leftarrow \mathcal{U} - \{(u_i,\ v_i)\}$;
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{(u_i,\ v_i)\}$;
  end if
end for
end

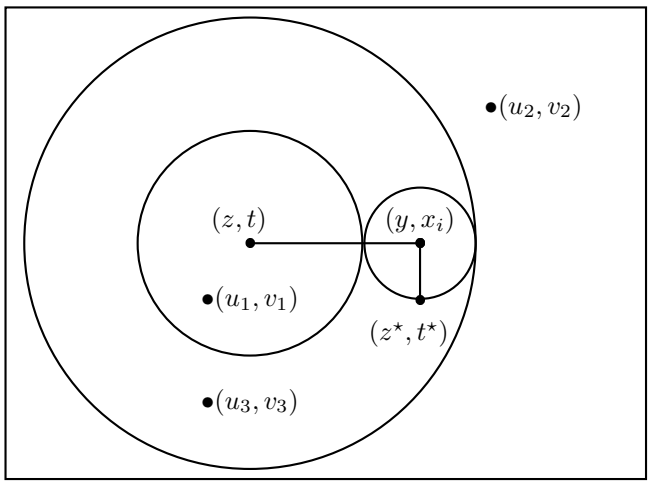

Figure 8: Elimination process in FADANA.

Algorithm 8 Algorithm FADANA: main procedure.

begin
$\mathcal{S} \leftarrow \{x_i,\ i = 1, m\}$;
$AD^\star \leftarrow +\infty$;
for $i = Card(\mathcal{S})$ **do**
  `Initialize`;
  while $\mathcal{U} \neq \varnothing$ **do**
    $(z, t) \leftarrow$ `selection`$(\mathcal{U}, \mathcal{C}, (x_i, y), Dist)$;
    $Dist(k) \leftarrow AD(z, t, x_i, y)$;
    $k = k + 1$;
    $\mathcal{U} \leftarrow \mathcal{U} - \{(z, t)\}$;
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{(z, t)\}$;
    if $Dist(k) \geq Min$ **then**
      `eliminate`$(\mathcal{U}, \mathcal{C}, (x_i, y), \delta, k)$
    else
      $Min \leftarrow Dist(k)$;
      if $Dist(k) < AD^\star$ **then**
        $AD^\star \leftarrow Dist(k)$;
        $z^\star \leftarrow z,\ t^\star \leftarrow t,\ x^\star \leftarrow x_i$;
      end if
      for $k = 1, Card(\mathcal{C})$ **do**
        `eliminate`$(\mathcal{U}, \mathcal{C}, (x_i, y), \delta, k)$
      end for
    end if
  end while
end for
The best triple in $\mathcal{S}$ is $(z^\star, t^\star, x^\star)$;
The least analogical dissimilarity is $AD^\star = AD(z^\star, t^\star, x^\star, y)$;
end

To go further, we have also devised a $LAESA$-like FADANA algorithm, in which the preliminary computation and storage is limited to $N.m^2$, where N is a certain number of *couples* of objects. The principle is similar to that of $LAESA$. The N *base prototypes* couples are selected among m^2 possibilities through a greedy process, the first one been chosen at random, the second one being as far as possible from the first one, and so on. The distance between couples of objects is, according to the definition of the analogical dissimilarity:

$$\delta\big((x,y),(z,t)\big) = AD(z,t,x,y)$$

Some preliminaries experiments are also given in the next section.

8.6 Experiments.

We have led preliminary experiments to measure the efficiency of FADANA. We have randomly drawn, according to an uniform distribution, a set $\mathcal{S}$ of size $m = 100$ vectors in the hypersphere of radius 1 in $\mathbb{R}^n$. Then we have drawn a new vector, according to the same distribution, and computed the triple in $\mathcal{S}$ which has the least AD with this point. The number of distances computed by FADANA (in the $AESA$-like version, with no base prototypes) is observed. This experiment is repeated 100 times, and the average number $M(n)$ distances computed by FADANA is obtained. It has to be compared to that of the brute force algorithm, which would be $100^3 = 10^6$ distances computed.

Figure 9 displays in solid line the number of distance computations plotted against the dimension of the vector space. It starts in an exponential shape, which is classical according to the notion of *curse of dimensionality* ([11]) and conform to most of the results in fast nearest neighbor search, including $AESA$ (see [6] for a review of these methods). Nevertheless, the method seems efficient, since only 1 distance over 50 has to be computed in dimension 16, and approximately 1 over 1000 in dimension 8.

When using the $LAESA$-like version, we can experiment on larger data bases, since we have no more to store m^4 analogical dissimilarities, but only $N.m^2$, with N the number of base prototypes. To compare the number of distance computation with the $AESA$-like version, we display in the dashed curve of Figure 9 the number of distance computations plotted against the dimension of the vector space for $m = 100$ and $N = 100$. For dimension $n = 16$, this method requires about 20 times more distance computations, and 40 times for $n = 4$.

Table 8.6 shows what happens when m increases to 200 with various values of N, the number of base prototypes, chosen among m^2 couples. The brute force method would give $m^4 = 8.10^6$ distance computations. This method does not seem to be as efficient as the previous version: as soon as n reaches 8, it has to compute up to a one sixth of what would do the brute force method. The examination of the table seems to indicate that N would have to be increased to larger values to reach a good compromise between the storage space and the speed of the decision.

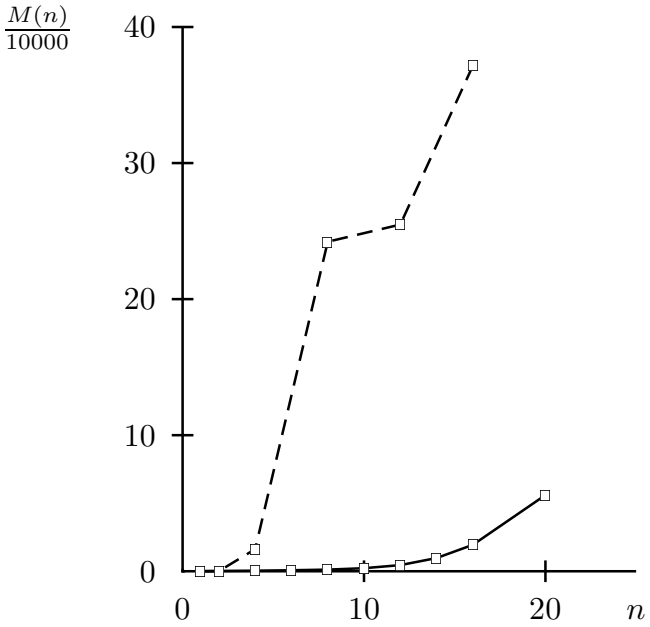

Figure 9: Efficiency of FADANA: number $M(n)$ of distance computations plotted against the dimension n of the vector space. The size m of the learning set is 100. The brute force algorithm would give $M(n) = 10^6$ for all n. The curve in solid line corresponds to the AESA-like version, with m^4 precomputed and stored distances. The curve in dashed line is the LAESA-like version, with 100 base prototypes (among 10000) and only Nm^2 stored distances.

n	1	2	4	8	12	16
$N = 50$	25	109	37984	1166240	1494387	2414179
100	25	113	37912	1164127	1491635	2410067
200	25	113	37783	1160130	1486807	2400166

Table 1: Number of distance computation with the $LAESA$-like version of FADANA, with $m = 200$.

9 Conclusion and future work.

In this article, we have investigated a formal notion of analogy between four objects in the same universe. We have given definitions of analogy, formulas and algorithms for solving analogical equations in some particular sets. We have given a special focus on objects structured as sequences, with an original definition of analogy based on the edit distance. We also have introduced, in a coherent manner, the new notion of analogical dissimilarity, which quantifies how far are four objects from being in analogy. This notion is useful for lazy supervised learning: we have shown how the time consuming brute force algorithm could be ameliorated by generalizing a fast nearest neighbor search algorithm, and given a few preliminary experiments. However, much is left to be done, and we want especially to explore further the following questions:

- What sort of data are particularly suited for lazy learning by analogy? We know from the bibliography that linguistic data has been successfully processed with learning by analogy techniques, in fields such as grapheme to phoneme transcription, morphology, translation. We are currently working on experiments on phoneme to grapheme transcription, which can be useful in some special cases in speech recognition (for proper names, for example). We also are interested on other sequential real data, such as biosequences, in which the analogical reasoning technique is (rather unformally) presently already used. The selection of the data and of the supervision are equally important, since both the search of the less dissemblant analogic triple and the labelling process are based on the same concept of analogy.

- What sort of structured data can be processed? Sequences can naturally be extended to ordered trees, in which several generalizations of the edit distance have already been defined. This could be useful, for example, in extending the nearest neighbor technique in learning prosodic trees for speech synthesis ([4]). We could also imagine that sequences models, like Hidden Markov Models (HMM) could be combined through an analogical construction.

- What sort of algorithms can be devised to let large amount of data be processed by such techniques? We have given a tentative extension of $LAESA$, but the results are not convincing yet. More experiments remain to be done with this type of algorithm. We have to notice also that not all the properties of analogical dissimilarity have been used so far. We believe that an algorithm with a precomputing and a storage in $\mathcal{O}(m)$ can be devised, and we are currently working on it.

In conclusion, we are confident in the fact that the new notion of analogical dissimilarity and the lazy learning technique that we have associated with it can be extended to real data, other structures of data and realistic sized problems.

References

[1] A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39–59, 1994.

[2] D. Aha. *Lazy Learning*. Kluwer Academic Publishers, 1997.

[3] M. Basu, H. Bunke, and A. Del Bimbo (Eds.). Syntactic and structural pattern recognition. *Special Section of IEEE Trans. Pattern Analysis and Machine Intelligence*, 27(7), 2005.

[4] L. Blin and L. Miclet. Generating synthetic speech prosody with lazy learning in tree structures. In *Proceedings of CoNLL-2000 : 4th Conference on Computational Natural Language Learning*, pages 87–90, Lisboa, Portugal, September 2004.

[5] H. Bunke and T. Caelli, editors. *Graph Matching in Pattern Recognition and Machine Vision, Special Issue of International Journal of Pattern Recognition and Artificial Intelligence*. World Scientific, 2004.

[6] E. Chávez, G. Navarro, R. Baeza-Yates, and J.-L. Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, 2001.

[7] A. Cornuéjols and L. Miclet. *Apprentissage artificiel : concepts et algorithmes*. Eyrolles, Paris, 2002.

[8] M. Crochemore, C. Hancart, and T. Lecroq. *Algorithmique du texte*. Vuibert, Paris, France, 2001.

[9] W. Daelemans. Abstraction considered harmful: lazy learning of language processing. In H. J. Van den Herik and A. Weijters, editors, *Proceedings of the sixth Belgian-Dutch Conference on Machine Learning*, pages 3–12, Maastricht, The Nederlands, 1996.

[10] M. Dastani, B. Indurkhya, and R. Scha. Analogical projection in pattern perception. *Journal of Experimental and Theoretical Artificial Intelligence*, 15(4), 2003.

[11] R. Duda, P. Hart, and D. Stork. *Pattern classification*. Wiley, 2001.

[12] B. Falkenhainer, K. Forbus, and D. Gentner. The structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, 41:1–63, 1989.

[13] D. Gentner, K. J. Holyoak, and B. Kokinov. *The analogical mind: Perspectives from cognitive science*. MIT Press, Cambridge, MA, 2001.

[14] D. Hofstadter and the Fluid Analogies Research Group. *Fluid Concepts and Creative Analogies*. Basic Books, New York, 1994.

[15] E. Itkonen and J. Haukioja. *A rehabilitation of analogy in syntax (and elsewhere)*, pages 131–177. 1997.

[16] Y. Lepage. Analogy + Tables = Conjugation. In G. Friedl and H.C. Mayr, editors, *Proceedings of NLDB'99*, pages 197–201, Klagenfurt, June 1999.

[17] Y. Lepage. Défense et illustration de l'analogie. In *Actes de la 8ème conférence sur le Traitement Automatique des Langues Naturelles (TALN)*, pages 373–377, Tours, France, 2001.

[18] Y. Lepage. *De l'analogie rendant compte de la commutation en linguistique*. Grenoble, 2003. Habilitation à diriger les recherches.

[19] Y. Lepage and S. Ando. Saussurian analogy: a theoretical account and its application. In *Proceedings of COLING-96*, pages 717–722, København, August 1996.

[20] L. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm aesa with linear preprocessing-time and memory requirements. *Pattern Recognition Letters*, 15:9–17, 1994.

[21] L. Miclet, S. Bayoudh, and A. Delhay. Définitions et premières expériences en apprentissage par analogie dans les séquences. In F. Denis, editor, *Conférence d'Apprentissage CAp 05*, pages 31–47. PUG, 2005.

[22] L. Miclet and A. Delhay. Analogy on sequences: a definition and an algorithm. Technical Report 4969, INRIA, October 2003.

[23] L. Miclet and A. Delhay. Relation d'analogie et distance sur un alphabet défini par des traits. Technical Report 5244, INRIA, July 2004. in French.

[24] M. Mitchell. *Analogy-Making as Perception*. MIT Press, Cambridge, MA, 1993.

[25] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[26] V. Pirrelli and F. Yvon. Analogy in the lexicon: a probe into analogy-based machine learning of language. In *Proceedings of the 6th International Symposium on Human Communication*, Santiago de Cuba, Cuba, 1999.

[27] T. Rowland. *Group*. Technical report, Mathworld, Wolfram, 2004. http://mathworld.wolfram.com/Group.html.

[28] S. Russel. *Use of knowledge in analogy and induction*. Pitman, 1989.

[29] J. Sakarovitch. *Eléments de théorie des automates*. Vuibert, Paris, France, 2003.

[30] D. Sankoff and J. Kruskal, editors. *Time Warps, String Edits and Macromolecules: the Theory and Practice of Sequence Comparison*. Addidon-Wesley, 1983.

[31] U. Schmid, H. Gust, K.-U. Kühnberger, and J. Burghardt. An algebraic framework for solving proportional and predictive analogies. In R. Young F. Schmalhofer and G. Katz, editors, *Proceedings of the European Conference on Cognitive Science (EuroCogSci 2003)*, pages 295–300, Osnabrück, Germany, 2003. Lawrence Erlbaum.

[32] N. Stroppa and F. Yvon. Analogical learning and formal proportions: Definitions and methodological issues. Technical Report ENST-2005-D004, École Nationale Supérieure des Télécommunications, June 2005.

[33] R.A. Wagner and M.J. Fisher. The string-to-string correction problem. *JACM*, 1974.

[34] F. Yvon. Paradigmatic cascades: a linguistically sound model of pronunciation by analogy. In *Proceedings of the 35th annual meeting of the Association for Computational Linguistics (ACL)*, Madrid, Spain, 1997.

[35] F. Yvon. Pronouncing unknown words using multi-dimensional analogies. In *Proceeding of the European conference on Speech Application and Technology (Eurospeech)*, volume 1, pages 199–202, Budapest, Hungary, 1999.

[36] F. Yvon. Pronouncing unknown words using multi-dimensional analogies. In *Eurospeech*, volume 1, pages 199–202, Budapest, Hungary, 1999.

[37] F. Yvon. Finite-state transducers solving analogies on words. Technical Report D008, Ecole Nationale Supérieure des Télécommunications, Paris, France, 2003.

[38] F. Yvon, N. Stroppa, A. Delhay, and L. Miclet. Solving analogical equations on words. Technical Report ENST2004D005, École Nationale Supérieure des Télécommunications, August 2004.

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399