Extraction of Syntactical Patterns from Parsing Trees

Jean-Gabriel Ganascia

Université Pierre et Marie Curie - LIP6 – 8, rue du Capitaine Scott – 75015 Paris – France

Abstract

This paper presents experiments with a new algorithm designed to detect recurrent syntactical patterns in natural language texts. It first describes the pattern extraction algorithm, which is based on an edit model generalized to SOT (Stratified Ordered Trees). Then it focuses on experiments with French classical literature of the 18th and 19th centuries. It goes on to evaluate efficiency before providing some examples of recurrent patterns that are typical of an 18th century author, Madame de Lafayette.

Keywords: edit distance, syntactical patterns, clustering, non-supervised learning

1. Introduction

Our aim is to extract syntactical figures from natural language texts, those figures corresponding to recurrent syntactical patterns, i.e. to patterns occurring many times in syntactical trees. The proposed algorithm takes a parsing tree as input, since our goal is not to generate or to refine grammars, but to detect general patterns with multiple approximate occurrences. The method described here is general and could be applied to any parsing tree, the only point is that it has to be expressed as a Stratified Ordered Tree (see section 2.1.2).

There are three reasons for doing this research.

The first is to characterize the personal style of authors, based on the hypothesis that the style of writing is embedded within their choice of syntactical structure and lexicon. In the future, this hypothesis will be tested on imitation. This is because many parodies exist where authors caricature great writers by bringing out certain salient features of their personal style and if our hypothesis is true, some of the authors' ways of writing will correspond to patterns which imitators mimic.

The second reason is educational. Our aim is to help school children, students or young writers to evaluate the richness and the diversity of their own style. It should also be possible to identify classical mistakes and propose corrections. For instance, people who learn foreign languages could take advantage of such an analysis to detect their own idiomatic expressions and mistakes. More precisely, it so happens that some constructions are correct from a grammatical point of view but unusual, which makes their frequent use strange or confusing for a native reader.

The third reason is academic. Computational linguistics could take advantage of such a study in order to distinguish different registers of language and to characterize them. Our claim is that the advances in natural language analysis coupled with artificial intelligence techniques may help to detect automatically such recurrent syntactical patterns, providing a new way of studying uses of language. In this way it would be possible to generate figures of speech and to automate the way classical rhetoric lists them.

In the proposed architecture, designed to discover such patterns, an unsupervised learning algorithm clusters similar patterns.

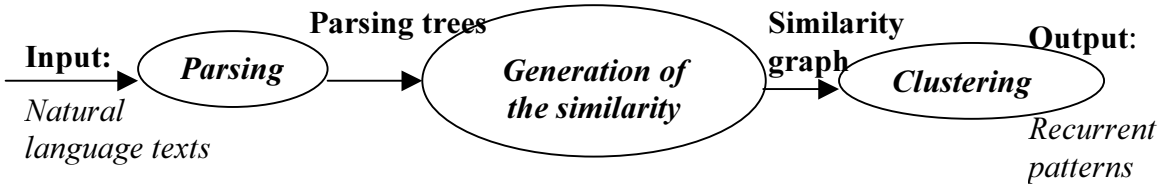
The first part of this paper (section 2) focuses on the learning algorithms involved. After a description of the whole chain (subsection 2.1), the paper describes the edit distance on which the overall clustering process is based (subsection 2.2) and then the generation algorithm (section 2.3) that builds the similarity graph. The following section (section 2.4) presents the "center star", designed to extract highly connected sub-graphs from the similarity graph and which correspond to clusters of similar patterns. The second part of the paper (section 3) describes some experiments. We are thus able to provide an evaluation of the proposed approach from both a computational (and quantitative) point of view, and a linguistic and qualitative one.

2. The Extraction Algorithm

This first section describes in detail the extraction algorithm that can be characterized as an unsupervised learning algorithm.

2.1. The Overall Chain

To summarize, the whole processing chain that transforms a natural language text into a set of frequent patterns is given below.



This subsection details the input of the process and the first step. As we shall see in the following, the algorithm is based on the use of the Stratified Ordered Tree (SOT) data structure, which will be described in subsection 2.1.2.

2.1.1. Natural Language Analysis

The main input of the system consists of a natural language text that is a sequence of sentences, each being affirmative, interrogative or exclamatory. A natural language analysis performs the first step, through parsing or categorization. It associates labels to words (noun, verb, etc.) and to groups of words (noun group, verb group, etc.). Since the paper is not focused on this natural language analysis, the parser and the categorization process used are not described in detail here. We shall just focus on the generality of the approach that has as input any natural language parsing tree with different grammar and different sets of labels. The only point is that the analysis transforms texts into trees or forests, i.e. into sequences of trees, which means that we exclude general graphs. Most of the time, trees are layered, i.e. depending on the level of the trees, labels belong to distinct classes. For instance, one level corresponds to non-recursive syntagmas, i.e. noun groups or verb groups; the second to word categories, i.e. articles or nouns; the third to attributes like gender or number; the next to lemma, i.e. canonical forms of verbs or nouns, and the last to words as they appear in sentences.

Note that our approach is not restricted to syntactical decomposition in non-recursive groups. The only limitation is that the result of the analysis has to be structured in a Stratified Ordered Tree (SOT) (see the following subsection). The experiments conducted here apply two different analyzers. The first uses Winbrill-0.3 trained at INALF-CNRS (Lecomte 1998). It just labels text words, without extracting groups. The second uses the Vergne-98 parser written by Jacques Vergne. (Cf. http://www.info.unicaen.fr/~jvergne and (Vergne 1999)) at GREYC (Groupe de REcherche en Informatique, Image, Instrumentation de Caen). In the future we plan to use "CORDIAL analyseur" from SYNAPSE, because it generates recursive parsing trees. For reasons of space, we shall report only the second experiment using the Vergne-98 parser.

2.1.2. Stratified Ordered Trees (SOT)

According to a classical definition, an ordered tree is a tree where left to right order between siblings is significant. In other words, it means that an ordered tree can be recursively defined as a labeled node followed by a forest, i.e. a sequence of trees.

All sequential data can obviously be represented with a depth-1 ordered tree. By adding levels to ordered trees, it is possible to organize data in a way that represents implicit background knowledge. For instance, a text, i.e. a sequence of characters, is a list of sentences, each of which is composed of words and punctuation marks. Therefore, it can be represented using a depth-3 tree that makes this structure explicit. Considering now parsing trees resulting from a syntactical analysis, there are two more levels corresponding to syntactical groups (i.e. noun or verb groups) and to word categorization.

As has previously been shown, ordered trees increase representation power and it is possible to detect similar sub-trees with respect to this data organization. It is also possible to extract general patterns that have multiple approximate occurrences. For instance, any specific sequence of syntactical groups which appears many times in the data may be detected without considering the corresponding words or their categories.

Nevertheless, due to the high number of potential pairs of sub-trees recurrent pattern detection is intractable. To make it manageable, nodes are categorized into *sorts* in such a way that two successfully matched nodes must be of the same sort. In other words, a match between two trees is valid if and only if the corresponding nodes in the matching are of the same sort.

In addition, we now suppose that there exists a total order on the set of sorts and that, with respect to this order, the sort of the son(s) is identical to, or immediately follows that of the father. This constraint defines the so-called stratification and the resulting structure is a SOT (Stratified Ordered Tree). More formally, by defining an ordered set, S (set of sorts), and a function *sort*(x) which associates a sort to each labeled node belonging to L, we can specify a SOT as an ordered tree where each node sort is the immediate successor of its father sort, except for the root which has no father.

In our favorite example, syntactical trees resulting from natural language text parsing, it means that the ordered set of sorts S contains five categories {Text, Sentence, Group, Category, Word} such that Text < Sentence < Group < Category < Word.

Given the SOT data structure, we shall first determine a similarity measure quantifying the approximate matching between SOTs (Cf. subsection 2.2). Then, it will be possible to efficiently generate a *similarity graph* recording the closest sub-trees of the input SOT (Cf.

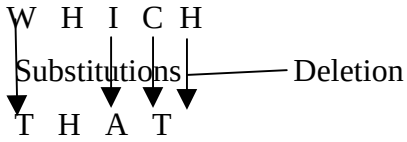
subsection 2.3). It is this similarity graph which serves as input of a clustering algorithm that builds groups of similar patterns.

2.2. Edit Distance

Much work has been done in molecular biology (e.g. Sagot et Viari 1996), music (Rolland et Ganascia 1999) or text analysis to compare sequences of characters. In the past, a lot of extremely good results have been obtained concerning the exact matching problems between strings, areas or binary trees (Karp et al. 1972; Landraud et al. 1989). Other approaches have dealt with approximate pattern matching. Some of them used dynamic programming techniques based on the notion of edit distance (Sankof et Kruskal 1983). A general overview of these techniques is to be found in (Crochemore & Rytter 1994). Let us just recall here some of the basic principles.

Definition: An *edition* is an operator that replaces one character or one sub-string of a string, or more generally one node of a tree, by another one. For instance, a substitution is an edition if it transforms one character of a string into another in the same position. An insertion (respectively deletion) which inserts (respectively deletes) one character in a string is also an edition.

Definition: An *edit distance* between two strings or two trees is based on the minimum number of editions that transform one string or one tree into another. For instance, here is an edit transformation from the string "WHICH" to the string "THAT":



It follows that the edit distance from "WHICH" to "THAT", i.e. $edit_{As}$(WHICH, THAT), is lower or equal than cost(substitution(W, T)) + cost(substitution(I, A)) + cost(substitution(C, T) + cost(deletion(H)). Let us note that in this case, it is equal, since there exist no cheaper set of edition which transforms WHICH in THAT.

2.2.1. Edit Distance Between Strings

Let us now consider that strings x and y are given as two tables of length n and m, i.e. as x[1..n] and y[1..m]. Then it is possible to build a matrix (n+1)×(m+1) called $EDIT_{As}$ where $EDIT_{As}(i, j)$ is filled with the value of $edit_{As}(x[1..i], y[1..j])$ for non null i and j, while EDIT(0, j) corresponds to the insertion of y[j], and EDIT(i, 0) to the deletion of x[i]. A simple formula summarizes the way the matrix elements are computed:

$$\text{edit}_{As}(x[1..i], y[1..j]) = \min \begin{cases} \text{edit}_{As}(x[1..i-1], y[1..j]) + deletion(x[i]) \\ \text{edit}_{As}(x[1..i-1], y[1..j-1]) + substitution(x[i], y[j]) \\ \text{edit}_{As}(x[1..i], y[1..j-1]) + insertion(y[j]) \end{cases}$$

where *deletion*(x[i]), *substitution*(x[i], y[j]) and *insertion*(y[j]) correspond to the cost of respectively the deletion of x[i], the substitution of x[i] by y[j] and the insertion of y[j].

2.2.2. Extension of Edit Model to Trees

The edit model can be extended to trees and forests, but its inherent complexity makes it intractable in general cases for pattern extraction algorithms. It has been proved that some efficient procedures exist (Zhang K 1993) under strict conditions, but the imposed restrictions preclude their use for practical problems.

Ganascia has shown (Cf. Ganascia (2001)) that, by restricting the structured input to SOT, i.e. to Stratified Ordered Trees, it is possible to build a new efficient pattern extraction algorithm. This algorithm uses as input a huge SOT containing 100,000 or more nodes, and generates clusters of small similar SOTs that appear to have multiple occurrences in the input SOT.

Since this paper is not concerned by all the algorithmic detail, we shall just provide an intuitive script of the way the edit distance between trees is computed.

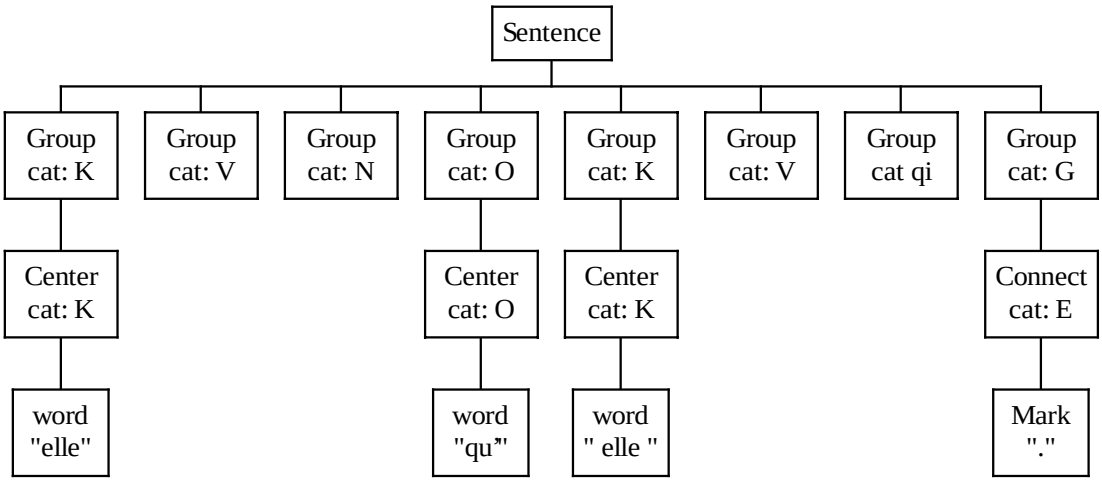
Before giving such a definition let us note that since trees are stratified, the node sorts refer directly to their level in trees. Therefore the comparison of sequences of nodes resulting from the left-hand exploration of two trees is equivalent to the comparison of those trees. Taking this remark into account, the edit distance between two SOTs is reduced to the edit distance between the sequences of nodes resulting from the left-hand exploration of those SOTs.

In a more formal way, by denoting lhe(T) the left-hand exploration of SOT T, the edit distance edit(T, T') between two SOTs T and T' can be expressed by $edit_{As}(lhe(T), lhe(T'))$.

2.3. Generation of the Similarity Graph

Using the edit distance, a labeled graph called the *similarity graph* is built which makes the distances between patterns explicit when they do not go beyond a fixed threshold. This similarity graph is of crucial importance; it constitutes the main input of the clustering module and includes all the patterns that generalize sub-trees of the input SOT.

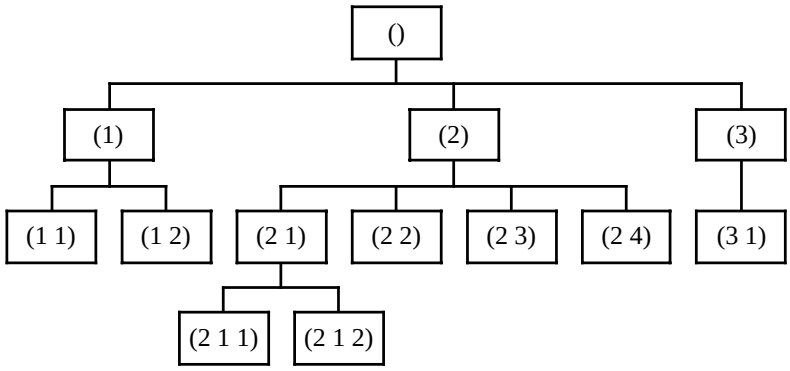
Let us note that this implicit generalization is a key-point in the overall algorithm, since it generates all general patterns including non-balanced ordered trees. In the case of natural language parsing trees, it means that generated patterns may look like the following:



More precisely, the similarity graph of a tree T contains all pairs of similar patterns of T with respect to the edit distance. In other words, its nodes correspond to sub-trees of T, and its arcs to pairs of such sub-trees, labeled with their edit-distance.

Therefore, building the similarity graph primarily requires listing all the patterns belonging to a tree T, including all their generalizations. For instance, considering the tree associated to the expression (3-1)×(2+5), this generation lists all the different generalizations corresponding to the expansion of different nodes of the tree, i.e. to x, x×y, x×(2+5), (3-1)×y, (3-z)×(x+y), (3-1), (3-x), etc. Since the maximum depth of ordered trees considered here is not very high, the number of such patterns is not so high. It is also possible to introduce some restrictions on the type of patterns, thus considerably limiting this number.

For the sake of clarity, let us consider the tree shown below. The listing of the sub-trees of T begins with sub-trees of depth-0, i.e. with nodes (), (1), (2), … (3 1).



Then it is possible to generate all the depth-1 sub-trees by expanding each node of the depth-0 sub-trees. Therefore, the first node () gives () (1) (2) (3), the second, i.e. (1), generates (1) (1 1) (1 2), the third (2), (2) (2 1) (2 2) (2 3) (2 4), etc.

The building of depth-2 sub-trees proceeds in the same way, by expanding depth-1 nodes in depth-1 sub-trees. For instance, the first depth-1 sub-tree () (1) (2) (3) will generate three depth-2 sub-trees:

1. () (1) (1 1) (1 2) (2) (3),
2. () (1) (2) (2 1) (2 2) (2 3) (2 4) (3),
3. () (1) (2) (3) (3 1)

From all three sub-trees, it is possible to expand other depth-1 nodes to generate all the depth-2 sub-trees. For instance, by expanding node 2 and node 3 in the first one, i.e. ()(1)(1 1)(1 2)(2)(3), we obtain ()(1)(1 1)(1 2)(2)(2 1)(2 2)(2 3)(2 4)(3) and ()(1)(1 1)(1 2)(2)(3)(3 1).

More generally, the generation algorithm starts from trees reduced to nodes and extends progressively each non-terminal node of those trees with all their sons, until no extension is possible. For the sake of clarity, we have introduced an intermediate function called "expansion". This function derives V_n that contains all the patterns extending at least one depth-n node from trees belonging to V_{n-1}. The algorithm just lists all the non-terminal depth-n nodes of each tree belonging to V_{n-1}, and then extends all the possible combinations of those nodes. Below is a pseudo-code formulation of the corresponding algorithm:

```
□□□□□□□□□□□□□□□□□□
    □□□□□ list V, integer n; □□□□□□ list VP
□□□□□□□ A ∈ V
        B ← all non terminal depth-n nodes of A
        PB ← set of non null parts of B
        □□□□□□□ E ∈ PB
                Q ← copy(A)
                □□□□□□□ N ∈ E
                        Extend node N in Q
                □□□□□□□ all
                Add Q to VP
        □□□□□□□ all
□□□□□□□ all
```

Once the expansion function is given, a simple algorithm generates all the patterns contained in any ordered tree T — let us call it S(T) —. Here is the corresponding pseudo-code:

```
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
    □□□□□ tree T; □□□□□□ list R
    □□□□□□□□□□□□□□ n ← 0 ; V ← nodes(T) ; R ← V ;
       □□□□□ V
               □□□□□□□□□(V, n, VP) ;
               R ← R ∪ VP ; V ← VP ; n ← n + 1 ;
       □□□ While
```

In conclusion, let us note that the graph does not contain all pairs of patterns, but only the closest ones (see (Ganascia 2001) for more details). By taking into consideration mathematical properties of the edit distance, it is possible to considerably reduce the computational complexity of similarity graph generation.

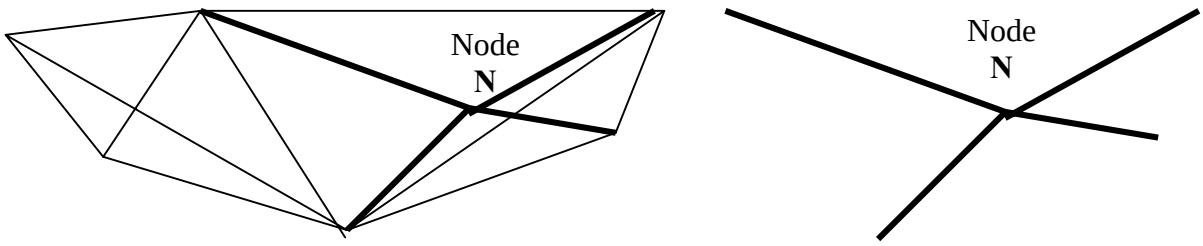
2.4. Clustering algorithm

The last step is the pattern extraction that is itself split into two logical operations: *categorization* which builds classes of similar patterns and the *description* of each cluster.

2.4.1. Categorization

Categorization groups patterns that are similar with respect to the edit distance. Since the similarity graph records all similarities between patterns, it is natural to extract clusters from this graph. However, there are many different ways to build such clusters.

We have chosen here a very efficient approach to pattern extraction, which is called the "center star" algorithm (Gusfield 1993). The basic principle can easily be described. Let us first define the notion of *centered star*.



Definition: A *star centered* on N is a graph of which all vertices contain node N. In other words, a star centered on N is composed of all nodes P such that the pair {N, P} is a vertex (see figure).

Using any similarity measure, the "center star" algorithm first computes all the star evaluations for all nodes of the similarity graph, then the best star is selected and the nodes belonging to it are discarded from the similarity graph before the process iterates.

2.4.2. Depiction

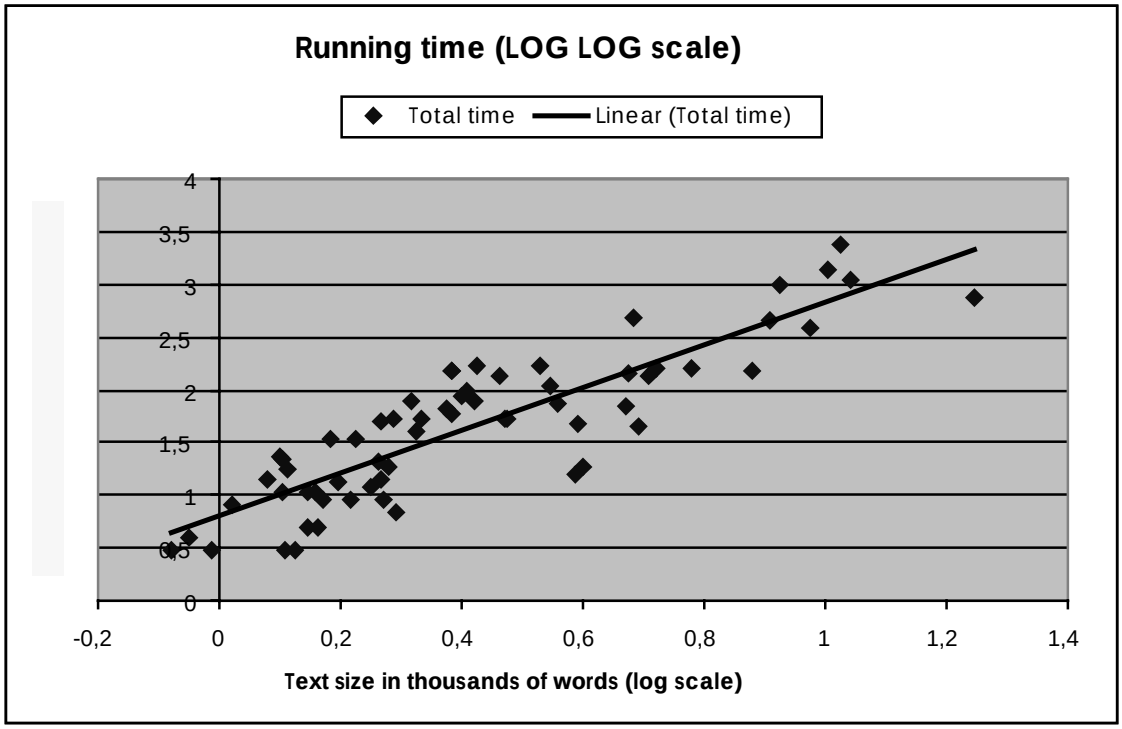
The final step in the clustering algorithm is related to the depiction of each cluster obtained. Our algorithm just chooses the pattern that maximizes the similarity with other members of the cluster and minimizes the similarity with members of other classes. To facilitate interpretation it is possible to show the original fragments of text that match the pattern. For the sake of illustration, let us consider again the pattern of section 1.3. It was extracted by running our program on a French 18th century text written by Madame de Lafayette and entitled *La comtesse de Tende*. It generalizes the syntactic tree of the following phrases: *"Elle exécuta ce qu'elle avait projeté :".* It also covers two sentences *"Il se joignit un nouveau tourment à ceux qu'elle avait déjà :"* and *"Elle vit toutes les raisons qu'il avait de l'aimer ;"* which means that the edit distance between this pattern and one of the generalizations of the syntactic trees derived from those two sentences is lower than a certain threshold. Under this condition, it can be considered as the description of the cluster containing those three sentences.

3. Experiments

The system was tested on more than 100 short stories and novels drawn from the 18th and 19th centuries, written by Madame de Lafayette, Guy de Maupassant, Alphonse Allais, Marcel Schwob, Alphonse Daudet, Eugène Mouton, Hégésippe Moreau and George Sand, among others. The texts were first parsed using the Vergnes-98 analyzer, and then the resulting sequence of syntactical trees was transformed into one SOT.

3.1. Efficiency

From a practical point of view, we studied the empirical complexity by relating execution time in seconds to input size in thousands of words, by reporting it on a log-log scale, and by applying a linear regression algorithm. It clearly appears (see figure) that the regression coefficient (i.e. the slope of the line) is equal to 2 which empirically shows that the temporal complexity is quadratic.



This first empirical result is highly satisfactory since the theoretical complexity of our algorithm is at least quadratic with the size of the input text. It comes from the way our algorithm computes the similarity graph, by exploring all the pairs of patterns. Because of the tree structure of texts, the number of sub-trees is linear with the number of sentences, so the global complexity cannot be any lower.

To avoid misunderstanding, it should be said that in the case of exact repetition (Karp et al. 1972) the procedure is clearly more efficient, but not in the case of approximate matching, as here.

The system has been implemented in C++ and tests are run on a Macintosh G3, with a 300 MHz processor. It takes a few seconds to extract patterns from short stories while, in the case of full novels, it may take one hour or more. It means that it is possible to apply our algorithm to extract patterns that are characteristic of full books, but not to deal with the lifetime work of an author. However, as we shall see in the next subsection, it can already be of great help.

3.2. Examples of extracted patterns

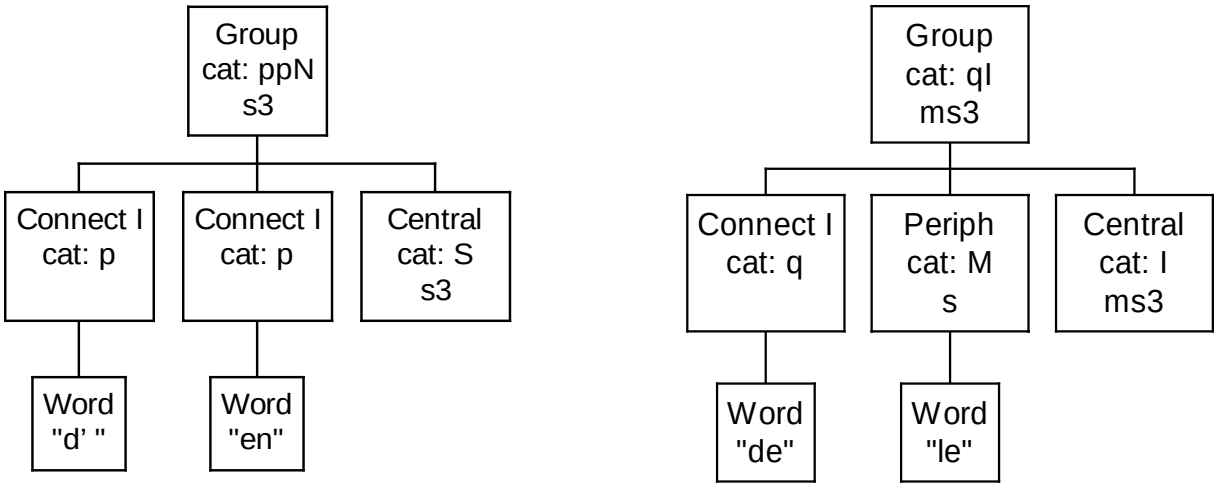
The pattern extraction program is completed by a discrimination procedure. Given two texts, this procedure detects the recurrent patterns covering multiple occurrences of some syntactical structure in the first text without detecting any occurrence of this structure in the second. This discrimination procedure has been employed to detect syntactical structures characteristic of one author, i.e. that distinguish this author from others. The author chosen was Madame de Lafayette, the two texts, a short story entitled *La comtesse de Tende* and a famous novel, *La princesse de Clèves*. Three 19th century authors were used by the discrimination procedure, Guy de Maupassant, Georges Sand and Marcel Schwob. More precisely, our corpus contains the following stories:

By Guy de Maupassant: *La peur (1882)* and *La peur (1884), La veillée, La rempailleuse, Pierrot, En mer, Un normand, Ce cochon de Morin, Les sabots*.

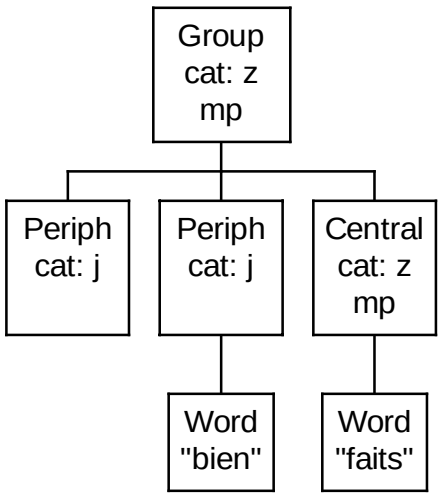
By Georges Sand: *La fée poussière, Le gnome des huîtres, Le marteau rouge, L'orgue du titan, La fée aux gros yeux.*

By Marcel Schwob: *Arachné, Béatrice, Sur les dents, Le Dom, L'homme double, Le fort, Gabelous, Parabole, Lilith, Conte des œufs, Les portes de l'opium.*

The following three patterns are present in the Lafayette texts without any occurrences in the other texts:



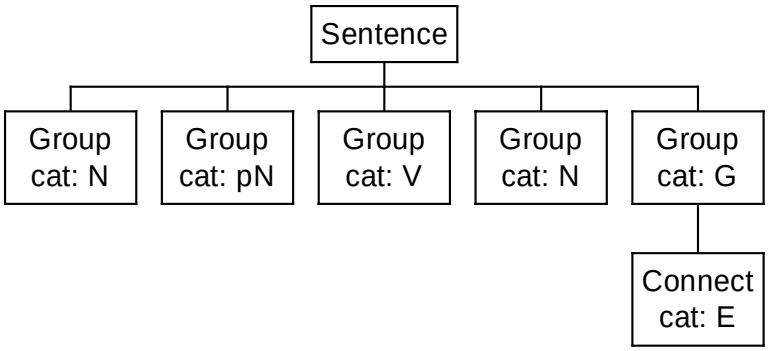
Among others, the first pattern covers the following French expression: "*de le supplier*", "*de l'éviter*", "*de l'aimer*", and others like "*de la tromper*", "*à le servir*", "*pour l'obliger*" etc. The second covers "*d'en avoir*", "*d'en attendre*", "*d'en garantir*", "*d'en faire*" etc. but also, "*sans en avoir*" and others which appear to have a very similar structure.



While the third covers the following three fragments "*admirablement bien faits*", "*parfaitement bien faits*", "*très bien fait*".

There are many others specific syntactical patterns characteristic of Madame de Lafayette. Among those, here is a syntactical structure that is frequently repeated:

It closely covers all those fragments (and others in Madame de Lafayette's work) whereas it is



virtually absent from the other authors: *"Le prince de Navarre prit la parole :*", "*La reine de Navarre avait ses favorites*", , "*Monsieur de Nemours prit la reine dauphine",* "*Madame de Clèvcs ne répondit rien*", "*Le comte de Tende aimait déjà le chevalier de Navarre* ;", "*La passion de la reine surmonta enfin toutes ses irrésolutions.*", etc.

There exists also many fragments less closely covered by this pattern. For instance: "*Madame de Chartes avait une opinion opposée*", "*Le comte de Tende sentit son procédé dans toute sa dureté ;*", "*La comtesse reçut ce billet avec joie*", "*L'humeur ambitieuse de la reine lui faisait trouver une grande douceur à régner*", etc.

Let us note that in most of those phrases, the word "*comte*" which means count in French and refers to a member of the aristocracy, is matched against other words like "*prince*", "*madame*" (i.e. madam), "*monsieur*" (i.e. sir), "*reine*" (i.e. queen) and "*comtesse*" (i.e. countess). Since no semantics has been given, this example shows how the syntax may convey semantics. This study is part of a more general project aiming to investigate relations between form and meaning in literature.

All those results were presented to experts of French literature. They recognize some of the pattern as characteristic of the 18th century style of writing, while others seemed to be more specific to Madame de Lafayette. We are now currently integrating our program to a reading environment. As the reader might will imagine, there are many other hypotheses that may be investigated using this procedure.

Note that there exists already many Computer-Assisted Research on Literature (CARL) known as stylometric analysis (Cf. (LoweD. Matthews R 1995), (Holmes D 1994) etc.) . However, those studies are based on the words, on their repetition, on their size, on their number or on their categories, not on the syntactical structure. How goal would be to compare and to combine our approach to those one, so the next step of our research would be a full statistical evaluation of our technique on an authorship attribution task.

4. Conclusion

In conclusion, it has been proved that the new text mining procedure described here is efficient and valuable. Literary critics could obviously take advantage of such kind of analysis, so it could be included in a reading environment for researchers or students. Moreover, the extracted patterns could be reused by a natural language synthesis procedure. Therefore, natural language processing would benefit from this research.

References

Crochemore M, Rytter W (1994), *Text Algorithms*, "Approximate pattern matching", pp. 237-251.

Ganascia J-G (2001) *Extraction of Recurrent Patterns from Stratified Ordered Trees*, 12th European Conference on Machine Learning, 3-7 September 2001, Freiburg, Germany, in proc. ed. Luc De Raedt & Peter Flach, Springer LNAI 2168, pp. 167-178.

Gusfield D. (1993) *Efficient methods for multiple sequence Alignment with Guaranteed Error Bounds*, Bull. Math. Biol., 55:141-154.

Holmes D (1994) Autorship Attribution, *Computers and the Humanities*, 28 pp. 87-106.

Karp R M., Miller R E., Rosenberg A L. (1972), *Rapid Identification of Repeated Patterns in Strings, Trees and Arrays*, in Proc. 4th Annu. ACM Symp. Theory of Computing, pp. 125-136.

Landraud A M., Avril J-F, Chrétienne P (1989) *An algorithm for Finding a Common Structure Shared by a Family of Strings*, IEEE transactions on Pattern Analysis and Machine Intelligence, 11 (8), pp. 890-895.

Lecomte J, (1998) BRILL14-JL5 / WINBRILL-0.3, user's manual available at INALF.

Lowe D, Matthews R (1995)*Shakespeare Vs. Fletcher: A Stylometric Analysis by Radial Basis Function*, Computer and the Humanities, 29 pp. 449-461.

Rolland, P-Y, Ganascia J-G, (1999) *Musical Pattern Extraction and Similarity Assessment*. In Miranda, E. (ed.). Readings in Music and Artificial Intelligence. Contemporary Music Studies - Vol 20. Harwood Academic Publishers.

Sagot A. , Viari A. (1996) A Double Combinatorial Approach to Discovering Patterns in Biological Sequences, *Combinatorial Pattern Matching*, Springer Verlag, LNCS 1075 pp. 168-208

Sankoff D., Kruskal J.B. (1983), *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison*, Addison-Wesley, Reading, Mass..

Vergne J., (1999) *Analyseur linéaire avec dictionnaire partiel, décembre 1999*, convention d'utilisation de l'analyseur de Jacques Vergne.

Zhang K. (1993) *Fast algorithms for the constrained editing distance between ordered labeled trees and related problems*, report N°361, Department of computer science, University of Western Ontario, London, Ontario, Canada.