CONTEXT FREE GRAMMARS AND NATURAL LANGUAGE DISCOURSES

Héctor Jiménez-Salazar
Guillermo Morales-Luna

Computer Science Section
CINVESTAV-IPN
e-mail: gmorales@cs.cinvestav.mx

April 1995

Abstract

In this paper we propose the use of classical Pattern Recognition techniques in order to build a context free grammar (CFG) from a given discourse in natural language. Our approach is based on similarities between strings over an alphabet. Thus we may treat in the same manner the recognition of words and the recognition of sentence structures, provided, in this last case, that the corpus has been "marked" previously. In spite that our word classification may be translated into a set of CFG rules, the obtained grammar may not be "congruent" with any formal grammar of natural language. The intended purpose of our approach is to collect the regularities of a set of strings into a set of CFG rules.

1 Introduction

Given a set of strings over a finite alphabet it is possible to "measure" similarities between strings through natural distance functions. This property allows the application of common clustering techniques to form goups of similar words. The characteristics of the formed groups are given by sets of common substrings, let us say *concepts*, of

the words appearing in the groups. Any such concept is synthetized by a CFG rule: a proper variable symbol is introduced with the rule that asserts that this symbol transforms into the given concept. The procedure is applied in a recursive way in order to obtain the greatest synthesis of the given set of strings as a set of CFG rules.

If the alphabet is the set of Roman letters then the above described procedures may be seen as an elementary morphological analysis of the words appearing in the text. The strings correspond to words in the discourse lexicon and are delimited by blank or punctuation signs.

On the other side, if the discourse is given as a marked corpus, where each word in the corpus has associated its grammar category, then any sentence in the corpus is translated quite directly into a string over the alphabet which is just the set of grammar categories. In this case the strings are sequences of grammar categories. The translation may be seen as the formation of sentence patterns.

Evidently, the CFG grammar obtained by our procedures need not be compatible with any formal grammar of the natural language. However, since grammar categories in a formalization of natural language may have some invariant particles we may expect that our classification may be consistent, to some extent, with formal grammars of natural language.

In Section 2. we present some basic techniques that ignore any previous knowledge of the syntactic structure of the given corpus and proceed canonically to the formation of clusters based on similarities.

In Section 3. we introduce some heuristics to guide the cluster formation in order to be more consistent with formal grammars of natural language.

As an extension of this paper we look toward procedures to construct a CFG grammar, let us say CG, based on similarities in such a way that given a formal grammar, NG, of natural language then the language generated by CG is contained in the language generated by NG. In other words the constructed grammar shall recognize just well formed sentences from the point of view of natural language.

2 String pattern recognition

2.1 Distances and similarities

Let A be a finite alphabet, and let A^* denote its dictionary:

$$A^* = \bigcup_{n \geq 0} A^n.$$

For any $\mathbf{a} \in A^*$, its *length* is

$$\text{length}(\mathbf{a}) = \text{Min}\{n | \mathbf{a} \in A^n\}.$$

Let us consider the relations "$\preceq_{str}$" and "$\preceq_{mon}$" in A^* defined as follows: $\forall \mathbf{a} = a_1 \cdots a_n, \mathbf{b} = b_1 \cdots b_m$, where $n = \text{length}(\mathbf{a})$ and $m = \text{length}(\mathbf{b})$,

$$\begin{aligned} \mathbf{a} \preceq_{str} \mathbf{b} &\Leftrightarrow \exists \mathbf{c}, \mathbf{d} \in A^* : \mathbf{b} = \mathbf{cad} \\ &\Leftrightarrow \mathbf{a} \text{ appears as a substring in } \mathbf{b}. \end{aligned}$$

$$\begin{aligned} \mathbf{a} \preceq_{mon} \mathbf{b} &\Leftrightarrow \exists f : [1,n] \to [1,m] \text{ increasingly monotone:} \\ &\quad \forall i \leq n : [a_i = b_{f(i)}] \\ &\Leftrightarrow \text{each symbol of } \mathbf{a} \text{ appears in } \mathbf{b}. \end{aligned}$$

Remarks:
a) Both relations "$\preceq_{str}$" and "$\preceq_{mon}$" are orderings in A^*.
b) "$\preceq_{mon}$" is an extension of "$\preceq_{str}$":

$$\forall \mathbf{a}, \mathbf{b} \in A^* : \mathbf{a} \preceq_{str} \mathbf{b} \Rightarrow \mathbf{a} \preceq_{mon} \mathbf{b}.$$

For any relation "$\preceq$" $\in \{$"$\preceq_{str}$","$\preceq_{mon}$"$\}$, and any $\mathbf{a}, \mathbf{b} \in A^*$ a "$\preceq$"-*meet* of $\{\mathbf{a}, \mathbf{b}\}$ is an element of maximal length in the set of "$\preceq$"-lower_bounds of the pair $\{\mathbf{a}, \mathbf{b}\}$. A pair of words may have more than one "$\preceq$"-meet.

In other words, a "$\preceq_{str}$"-meet of $(\mathbf{a}, \mathbf{b})$ is a common substring of maximal length; a "$\preceq_{mon}$"-meet of $(\mathbf{a}, \mathbf{b})$ is an element of maximal length that can be (increasingly) mapped into both $\mathbf{a}$ and $\mathbf{b}$.

Any meet "operation" is idempotent, conmutative, associative, the empty string is the null element, i.e.

$$nil \text{ is the only "}\preceq\text{"-meet of } nil \text{ itself and } \mathbf{a}, \forall \mathbf{a},$$

and it is consistent with the ordering:

$$(\mathbf{a} \text{ is the only "}\preceq\text{"-meet of } \mathbf{a} \text{ and } \mathbf{b}) \Leftrightarrow (\mathbf{a} \preceq \mathbf{b}).$$

Any meet operation in A^* has no unit because A^* is not upperly bounded.

Proposition: $\forall \mathbf{a}, \mathbf{b} \in A^*$ if $n = \text{length}(\mathbf{a})$ and $m = \text{length}(\mathbf{b})$ then a $\preceq_{str}$-meet can be computed in $O(nm)$ symbol scannings over $\mathbf{a}$ and $\mathbf{b}$.

Proof: Let's slide **a** over **b**. At each new position we check along **a** for symbol equality to find the longest common substring of **a** and **b**. We keep track of these longest common substrings in order to choose, at the end, the greatest of them all.

The pseudocode to compute the "$\preceq_{str}$"-meet of $(\mathbf{a}, \mathbf{b})$ is the following:

{ **for** $i = 1, \ldots, m$ **do**
{let s_i be the longest chain of the form
$s_i = [b_{i+j_i+k-2}]_{1 \leq k \leq k_i} = [a_{j_i+k-1}]_{1 \leq k \leq k_i}$
for some j_i and k_i } ;
let i_0 be such that $\text{length}(s_{i_0}) = \text{Max}_{i \leq m} \text{length}(s_i)$;
return s_{i_0}
}

The computation of a "$\preceq_{mon}$"-meet of two given strings $\mathbf{a}, \mathbf{b}$ is more complex. In fact, this problem can be posed as an equivalent problem in Graph Theory. Indeed, let $G_{\mathbf{a},\mathbf{b}} = (V, E, w)$ be the weighted digraph whose sets of vertexes and edges are specified as follows:

V has $n + m$ vertexes, one for each appearance of any symbol in **a** and **b**,

E has two kinds of edges: if $c_i c_{i+1}$ is a pair of consecutive symbols in **a** or **b**, then $\overline{c_i c_{i+1}}$ is an edge, of weight 1; if a symbol c appears in both **a** and **b** then there is an edge $\overline{cc}$ going from the c in **a** to the c in **b**, of weight $n + m$, and there is also a back edge $\overline{cc}$ going from the c in **b** to the c in **a**, of weight $n + m$ as well.

Initial vertexes Correspond to the first symbols in **a** and **b**.

End vertexes Correspond to the end symbols in **a** and **b**.

Extremal path Is a path going from an initial vertex to an end vertex **without** visiting twice any vertex.

Any "$\preceq_{mon}$"-meet of $\mathbf{a}, \mathbf{b}$ corresponds naturally to a extremal path of maximmum weight in $G_{\mathbf{a},\mathbf{b}}$.

Thus the problem to find a "$\preceq_{mon}$"-meet is equivalent to find a maximmum extremal path.

Let "$\preceq$" be an ordering in A^*.

If $\mathbf{a}, \mathbf{b} \in A^*$ are two strings such that $\mathbf{a} \preceq \mathbf{b}$ let us denote by $\mathbf{b} - \mathbf{a}$ the word obtained from $\mathbf{b}$ after deleting the image of $\mathbf{a}$ into $\mathbf{b}$.

Let us consider the function

$$\begin{array}{rcl} d_{\preceq} : A^* \times A^* & \rightarrow & R \\ (\mathbf{a}, \mathbf{b}) & \mapsto & d_{\preceq}(\mathbf{a}, \mathbf{b}) = \text{length}(\mathbf{a} - \mathbf{c}) + \text{length}(\mathbf{b} - \mathbf{c}) \end{array}$$

where $\mathbf{c}$ is a "$\preceq$"-meet of $\mathbf{a}$ and $\mathbf{b}$.

Clearly, $d_{\preceq}$ is a *distance* in A^*.

Remark: $d_{\preceq mon}$ is a refinement of $d_{\preceq str}$ in the following equivalent senses:

1. $\forall \mathbf{a}, \mathbf{b} \in A^* : d_{\preceq mon}(\mathbf{a}, \mathbf{b}) \leq d_{\preceq str}(\mathbf{a}, \mathbf{b})$.
2. The topology induced by $d_{\preceq mon}$ is a refinement of the topology induced by $d_{\preceq str}$.
3. The identity map $(A^*, d_{\preceq mon}) \rightarrow (A^*, d_{\preceq str})$ is continuous.

Let $h : R^+ \rightarrow [0, 1[$ be a differentiable increasing homeomorphism, for instance, $x \mapsto \frac{x}{1+x}$. Then

$$\forall x, y \in R^+ : \ h(x + y) \leq h(x) + h(y).$$

The function

$$\hat{d}_{\preceq} = h \circ d_{\preceq}$$

is a distance. It is the *normalized distance* induced by $d_{\preceq}$.

Let $s_{\preceq} : A^* \times A^* \rightarrow]0, 1]$ be defined as

$$\forall \mathbf{a}, \mathbf{b} \in A^* : \ s_{\preceq}(\mathbf{a}, \mathbf{b}) = 1 - \hat{d}_{\preceq}(\mathbf{a}, \mathbf{b}).$$

$s_{\preceq}$ is a *similarity* function: the closer to 1 is the value $s_{\preceq}(\mathbf{a}, \mathbf{b})$, the strings $\mathbf{a}$ and $\mathbf{b}$ are more likely. The similarity properties are

Definiteness $\forall \mathbf{a}, \mathbf{b} \in A^* : 0 \leq s_{\preceq}(\mathbf{a}, \mathbf{b}) \leq 1$ and

$$s_{\preceq}(\mathbf{a}, \mathbf{b}) = 1 \ \Leftrightarrow \ \mathbf{a} = \mathbf{b}.$$

Symmetry $\forall \mathbf{a}, \mathbf{b} \in A^* : \ s_{\preceq}(\mathbf{a}, \mathbf{b}) = s_{\preceq}(\mathbf{b}, \mathbf{a})$.

"Triangle inequality" $\forall \mathbf{a}, \mathbf{b} \in A^* :$

$$\forall \mathbf{c} : \ s_{\preceq}(\mathbf{a}, \mathbf{b}) \geq \mathrm{Max}\{s_{\preceq}(\mathbf{a}, \mathbf{c}) - \hat{d}_{\preceq}(\mathbf{c}, \mathbf{b}), s_{\preceq}(\mathbf{c}, \mathbf{b}) - \hat{d}_{\preceq}(\mathbf{a}, \mathbf{c})\}.$$

Remark: The calculations of $d_{\preceq}$ and $s_{\preceq}$ have the same complexity as the calculation of "$\preceq$"-meets.

Conversely, we may construct a "pseudosimilarity" function before a corresponding distance. Namely, let "$\preceq$" be an ordering in A^* related to a sort of string inclusion. Given an increasing homeomorphism $h : [0, +\infty[\to [0, 1[$,

$$\begin{array}{ccc} h(0) & = & 0 \\ h(x) & \underset{x \nearrow +\infty}{\longrightarrow} & 1, \end{array}$$

let $s_{\preceq} : (\mathbf{a}, \mathbf{b}) \mapsto h \circ \mathrm{length}(\mathbf{c})$, where $\mathbf{c}$ is a "$\preceq$"-meet of $\mathbf{a}, \mathbf{b}$.

For instance, if we consider as h an homotecy composed with the hyperbolic cotangent

$$h_a : x \mapsto \coth(ax) = \frac{e^{ax} - e^{-ax}}{e^{ax} + e^{-ax}}, \ a > 0,$$

then, by modifying a, we can make as "pseudosimilar" as we like any two strings with a common substring of positive length.

$s_{\preceq}$ is not a similarity function because it doesn't happen that

$$s_{\preceq}(\mathbf{a}, \mathbf{a}) = 1.$$

In fact, we have

$$s_{\preceq}(\mathbf{a}, \mathbf{a}) \underset{\mathrm{length}(\mathbf{a}) \nearrow +\infty}{\longrightarrow} 1.$$

Thus a string is more pseudosimilar to itself as it is longer.

Nevertheless a pseudosimilarity function may be used to collect strings by considering the pairwise likelyhood of the strings.

2.2 Clustering analysis

Let A be a finite alphabet and let $d : A^* \times A^* \to [0, 1]$ be a normalized distance. Let $s : A^* \times A^* \to [0, 1]$, where $s = 1 - d$, be its corresponding similarity function.

The *Clustering Problem* is the following:

Instance: A set $W = \{\mathbf{w}_j\}_j \subset A^*$ of strings.
Goal: A partition $\mathcal{P}_W = \{P_i\}_i$ of W such that each P_i is a cluster which collects the more pairwise alike strings.

There has been some remarkable methods in the literature to solve the Clustering Problem. We will review two of them. In spite that we will present the methods considering just distances between strings, they can be reconstructed to consider either similarity functions or pseudosimilarity functions.

2.2.1 Adaptive sample set construction

This method classifies a set of strings by clustering in a cover with no previously known number of classes. In the first part, it takes a set of the given strings as a *learning set* to recognize the clusters. After that, any new string is classified in the cluster in which it fits better.

It is very important to select a representative element in the cluster, which is intended to be an *average* element in the class, and to measure distances between clusters.

Let us go into the preliminaries of the method.

Let $P \in A^*$ be a finite set of m strings. For each $\mathbf{w} \in P$ let

$$d_P(\mathbf{w}) = \frac{1}{m-1} \sum_{\mathbf{v} \in P - \{\mathbf{w}\}} d(\mathbf{v}, \mathbf{w})$$

be the average distance between $\mathbf{w}$ and its companion elements in P.

Let the *intraset* distance in P be the average of those average distances:

$$\mathit{Intra}(P) = \frac{1}{m} \sum_{\mathbf{w} \in P} d_P(\mathbf{w}).$$

An *average* element in P is an element $\mathbf{w}_0 \in P$ that "realizes" the intraset distance in the set, i. e. it is such that

$$|d_P(\mathbf{w}_0) - \mathit{Intra}(P)| = \mathrm{Min}\{|d_P(\mathbf{w}) - \mathit{Intra}(P)| : \mathbf{w} \in P\}.$$

Given two sets of strings $P_1, P_2 \in A^*$, with m_1 elements and m_2 elements respectively, the *interset* distance between P_1 and P_2 is

$$\mathit{Intra}(P_1, P_2) = \frac{1}{m_1 m_2} \sum_{(\mathbf{w}_1, \mathbf{w}_2) \in P_1 \times P_2} d(\mathbf{w}_1, \mathbf{w}_2).$$

Learning process Suppose given a set of learning sample strings $W_0 = \{\mathbf{w}_j\}_j \subset A^*$ and a fixed ratio ρ to bound the intraset distance in the clusters to be formed.

Algorithm:

1. Let P consist of the first string $P = \{\mathbf{w}_1\}$, let $\mathbf{w}_P = \mathbf{w}_1$ be the average element in P and let the current partition be $\mathcal{P} = \{P\}$.

2. Repeat the following cycle till no further modifications in $\mathcal{P}$ were done.

   (a) For each word $\mathbf{w} \in W_0$ do the following:

   - If there is $P \in \mathcal{P}$ such that $d(\mathbf{w}, \mathbf{w}_P) < \rho$ then
     - put $\mathbf{w}$ into P, and
     - recalculate the average element $\mathbf{w}_P$.

     otherwise
     - define a new cluster P consisting of just $\mathbf{w}$ with average element $\mathbf{w}_P = \mathbf{w}$,
     - enlarge $\mathcal{P}$ by putting into it the new cluster P.

   (b) Repeat the cycle.

Classification process After the recognition of the clusters, any new string is classified into the nearest cluster.

2.2.2 "Maximin" distance

This algorithm builts also the classifying classes and it puts into them the given strings.

Suppose given a set of sample strings $W = \{\mathbf{w}_j\}_j \subset A^*$. The first cluster is fixed by the first string. Later on for any non-classified string $\mathbf{w}$ let us order the representative elements of the formed class according to their distances to $\mathbf{w}$. Let us take the shortest distance. Then we take the greates of such distances, varying $\mathbf{w}$ and we compare the obtained value with the average interset distance of the formed classes class in order to decide whether we define a new cluster or we incorporate $\mathbf{w}$ into an already formed class. More precisely, let us proceed as follows:

Algorithm:

1. Let $\mathbf{z}_1 = \mathbf{w}_1$. Mark $\mathbf{w}_1$ as classified.

2. Let j_1 be such that $d(\mathbf{w}_{j_1}, \mathbf{z}_1) = \text{Max}\{d(\mathbf{w}, \mathbf{z}_1) | \mathbf{w} \in W\}$.

3. Let $\mathbf{z}_2 = \mathbf{w}_{j_1}$. Mark $\mathbf{w}_{j_1}$ as classified. $\mathbf{z}_1, \mathbf{z}_2$ are the centers of two clusters P_1, P_2 whose ratios coincide with $\frac{1}{2}d(\mathbf{z}_1, \mathbf{z}_2)$. Let $\mathcal{P} := \{P_1, P_2\}$ and $m_{\mathcal{P}} := 2$.

4. While there remain any non-marked elements in W do the following:

   (a) Let ρ be the average distance between any two $\mathbf{z}$'s:

   $$\rho := \frac{1}{(m_{\mathcal{P}} - 1)m_{\mathcal{P}}} \sum_{1 \le i_1 < i_2 \le m_{\mathcal{P}}} d(\mathbf{z}_{i_1}, \mathbf{z}_{i_2}).$$

   (b) For any non-marked element $\mathbf{w}$ in W,

   - let $i(\mathbf{w})$ be such that

   $$d(\mathbf{w}, \mathbf{z}_{i(\mathbf{w})}) = \text{Min}\{d(\mathbf{w}, \mathbf{z}_i) | i \in [1, m_{\mathcal{P}}].\}$$

   (c) Let j_1 be such that

   $$d(\mathbf{w}_{j_1}, \mathbf{z}_{i(\mathbf{w}_{j_1})}) = \text{Max}\{d(\mathbf{w}, \mathbf{z}_{i(\mathbf{w})}) | \mathbf{w} \in W \text{ has not been marked}\}.$$

   - If $d(\mathbf{w}_{j_1}, \mathbf{z}_{i(\mathbf{w}_{j_1})}) > \rho$ then
     - Let $m_{\mathcal{P}} := m_{\mathcal{P}} + 1$, and $\mathbf{z}_{m_{\mathcal{P}}} := \mathbf{w}_{j_1}$,
     - define a new cluster with center $\mathbf{z}_{m_{\mathcal{P}}}$, ratio ρ and
     - mark $\mathbf{w}_{j_1}$ as classified,

     otherwise
     - put $\mathbf{w}_{j_1}$ into the cluster centered at $\mathbf{z}_{i(\mathbf{w}_{j_1})}$,
     - mark $\mathbf{w}_{j_1}$ as classified.
   - Look up for another non-marked element in W.

   (d) Repeat the cycle.

2.2.3 String clusters and CFG's

In this section we sketch a procedure to build a context free grammar (CFG) to generate a set of strings that have been classified into several clusters.

Let $d : A^* \times A^* \to R$ be a distance in the dictionary of A, let's assume that d is defined by the "$\preceq_{str}$" ordering in A^*.

Let $W = \{\mathbf{w}_j\}_j \subset A^*$ be a given language.

Let us assume that $\mathcal{P} = \{P_i\}_i$ is a partition of W by clusters:

$$\begin{aligned} \forall \mathbf{w} \in W \exists! P \in \mathcal{P} \quad &: \quad \mathbf{w} \in P \\ \forall \mathbf{w}, \mathbf{v} \in W, P \in \mathcal{P} \quad &: \quad \mathbf{w} \in P, \mathbf{v} \notin P \Rightarrow \operatorname*{Max}_{\mathbf{u} \in P} d(\mathbf{w}, \mathbf{u}) < d(\mathbf{w}, \mathbf{v}) \end{aligned}$$

Inner-cluster word generation

Let $P = \{\mathbf{w}_1, \ldots, \mathbf{w}_k\}$ be a cluster. For any two strings $\mathbf{w}_i, \mathbf{w}_j \in P$ let $\mathbf{u}_{ij}$ be their meet. Since $\mathbf{w}_i, \mathbf{w}_j$ are in P we may expect that any $\mathbf{u}_{ij}$ is as longest as possible.

We are going to construct a CFG G_P such that

$$\forall \mathbf{w} : \ \mathbf{w} \in P \ \Rightarrow \ \mathbf{w} \in L(G_P).$$

A pairwise method: Let L be the list consisting of the set of $\binom{k}{2} = \frac{k(k-1)}{2}$ pairs of elements in P ordered increasingly by the size of their "$\preceq_{str}$"-meet. We proceed as follows:

1. Initially, the set Va of grammar variables and the set Pr of grammar productions are empty.
2. For each pair $\{\mathbf{w}, \mathbf{v}\}$ in L do the following:
   (a) Let $\mathbf{u}$ be the meet of $\{\mathbf{w}, \mathbf{v}\}$.
   (b) Add a new grammar variable symbol V, put it into Va and add the production $V \to \mathbf{u}$ into Pr.
   (c) Substitute all the appearances of the string $\mathbf{u}$ in strings in P by V. (Note that it is not necessary to recalculate distances: they may disminish by a uniform number.)
   (d) Supress in L the pair $\{\mathbf{w}, \mathbf{v}\}$.
   (e) Repeat the cycle.

3. Add a new grammar symbol V_P, which will be considered as the *initial* symbol in P. Put V_P into Va.

4. For any $\mathbf{w}$ in P (which has been rewritten by the above process and may contain grammar variables) add the production $V_P \to \mathbf{w}$ into Pr.

Evidently $G_P = (Va, A, Pr, V_P)$ is a CFG that generates the strings in P.

In this construction we may associate to each variable symbol $V \in Va$ a *level value*:

- If V derives directly only strings in A^* then $Level(V) = 1$.
- If V derives directly a string that involves a symbol of level $n-1$ then $Level(V) = n$.

A centered method: Let us order the strings in P by their average distances to their companion elements in P:

$$\forall \mathbf{w} \in P : \; d_P(\mathbf{w}) = \frac{1}{m-1} \sum_{\mathbf{v} \in P - \{\mathbf{w}\}} d(\mathbf{v}, \mathbf{w}).$$

In this method we will compare the average elements with all other elements to find the grammar productions. More precisely we will proceed as follows:

1. Let P' be a copy of P. Let Q' be the list of processed strings in P. Initially, Q' is empty, the set Va of grammar variables and the set Pr of grammar productions are empty as well.

2. While P' has more than one string do the following:

   (a) Let $\mathbf{w}_0 \in P'$ realize the average of the average intraset distances in P', i.e. if
   $$\mu_{P'} = \frac{1}{\mathrm{card}(P')} \sum_{\mathbf{w} \in P'} d_{P'}(\mathbf{w})$$
   then $\mathbf{w}_0$ is the average element satisfying
   $$|d_{P'}(\mathbf{w}_0) - \mu_{P'}| = \mathrm{Min}\{|d_{P'}(\mathbf{w}) - \mu_{P'}| : \mathbf{w} \in P'\}.$$

   (b) Pop out $\mathbf{w}_0$ from P' and push it into Q'.

(c) For each $\mathbf{w} \in P'$ do the following:

- Let $\mathbf{u}$ be the meet of $\{\mathbf{w}, \mathbf{w}_0\}$.
- Add a new grammar variable symbol V, put it into Va and add the production $V \to \mathbf{u}$ into Pr.
- Substitute all the appearances of the string $\mathbf{u}$ in strings in P' by V.
- Repeat the cycle.

(d) Check the size of P' to decide whether to repeat this process.

3. Add a new grammar symbol V_P, which will be considered as the *initial* symbol for cluster P. Put V_P into Va.

4. For any $\mathbf{w}$ in Q' add the production $V_P \to \mathbf{w}$ into Pr.

Global word generation Now let us consider the whole classification of a given set of strings W into a clustering $\mathcal{P}$.

We proceed iteratively to built a global CFG $G_{W,\mathcal{P}}$ that generates the set of given examples W:

1. For each $P \in \mathcal{P}$, compute the "local" CFG G_P as has been explained before.

2. Sustitute the strings that are consequences of productions in G_P by their correspondent variable symbol in all the strings that belong in any other cluster $Q \in \mathcal{P}$. Proceed according to a cluster ordering and the level values of variables in G_P.

3. Collect the obtained set of words by the above transformations, into a new set of clusters.

4. Repeat the process:

$$\begin{pmatrix}\text{Construction of local}\\ \text{grammars.}\end{pmatrix} \to \begin{pmatrix}\text{"Inverse" sustitutions}\\ \text{of productions in a lo-}\\ \text{cal grammar in the}\\ \text{other clusters.}\end{pmatrix} \to \begin{pmatrix}\text{Construction of a new}\\ \text{clustering.}\end{pmatrix}$$

till any of the following conditions are satisfied:

- The clusters in the new clustering are all monads.
- The new clustering does not "refine" the last clustering.
- The new clustering consist of just one cluster.

5. Finally, add a new grammar symbol V_G, which will be considered as the *initial* symbol of the whole grammar G. Add the productions $V_G \to V_P$ for each cluster P in the last clustering obtained.

The obtained CFG $G_{W,\mathcal{P}}$ may have many "unit productions". They shall be eliminated in an equivalent way.

2.3 Heuristics for string cluster analysis

2.3.1 Heuristics based on substring positioning

Let us follow the next criteria to distinguish some string sets:

1. Let $l_0 \in N$ be a threshold. The set of l_0-*long* strings is defined as follows:

$$\forall \mathbf{a} \in A* : \text{length}(\mathbf{a}) > l_0 \Rightarrow \mathbf{a} \in \text{Long}(l_0).$$

2. For any substring $\mathbf{a}_0 \in A*$ let us consider the sets

$$\begin{aligned} \text{Prfix}(\mathbf{a}_0) &= \{\mathbf{a} \in A* | \exists \mathbf{b} : \mathbf{a} = \mathbf{a}_0\mathbf{b}\} \\ \text{Infix}(\mathbf{a}_0) &= \{\mathbf{a} \in A* | \exists \mathbf{b}, \mathbf{c} : \mathbf{a} = \mathbf{b}\mathbf{a}_0\mathbf{c}\} \\ \text{Sufix}(\mathbf{a}_0) &= \{\mathbf{a} \in A* | \exists \mathbf{b} : \mathbf{a} = \mathbf{b}\mathbf{a}_0\} \end{aligned}$$

3. Given any partition $\mathcal{P}$ in $A*$ in each set $P \in \mathcal{P}$ we may consider a specific recognition processing.

We will consider specific processings in each subset constructed above.

Specific processings Let us modify the algorithm "MaxiMin" in order that it distinguishes appearences of substrings in specific relative positions of any two words. We will apply this modification in sets of the form $\text{Long}(l_0)$, for a threshold $l_0 \in N$. Later on, we may repeat this procedure with an smaller $l_0 \in N$.

Input: 1. A set $W = (\mathbf{w}_i)_i$ of example strings.
2. A flag $\alpha \in \{0, 1, 2\}$ that indicates whether we are looking for prefixes, infixes or sufixes in pairs of strings.
3. A threshold $l_0 \in N$.

Output: A classification of the words in W whose length exceeds l_0.

In fact, the modification consists of "MaxiMin" applied to the set

$$W_{l_0} = W \cap \mathrm{Long}(l_0),$$

with distance

$$d_\alpha : (\mathbf{w}_1, \mathbf{w}_2) \mapsto \begin{cases} \mathrm{length}(\mathbf{w}_1 - \mathbf{c}_0) + \mathrm{length}(\mathbf{w}_2 - \mathbf{c}_0) & \text{if } \alpha = 0 \\ d_{\preceq}(\mathbf{w}_1, \mathbf{w}_2) & \text{if } \alpha = 1 \\ \mathrm{length}(\mathbf{w}_1 - \mathbf{c}_2) + \mathrm{length}(\mathbf{w}_2 - \mathbf{c}_2) & \text{if } \alpha = 2 \end{cases}$$

where

$\mathbf{c}_0$: is the longest common prefix of $\mathbf{w}_1$ and $\mathbf{w}_2$, and
$\mathbf{c}_2$: is the longest common sufix of $\mathbf{w}_1$ and $\mathbf{w}_2$.

Remarks:

1. According to the value of the parameter α the above procedure introduces in W_{l_0} a partition which is a refinement of the cover in W_{l_0} induced by the covers

$$\mathrm{Prfix}(\mathbf{a}_{0,j})_{j \in J}$$
$$\mathrm{Infix}(\mathbf{a}_{0,j})_{j \in J}$$
$$\mathrm{Sufix}(\mathbf{a}_{0,j})_{j \in J}$$

   where $(\mathbf{a}_{0,j})_{j \in J}$ is a finite set of substrings.

2. It is important to distinguish relative positions in common substrings because relative positioning is congruent with some semantic features of substrings coming fron natural languages discourses. Indeed,

   - in the case of words in a natural language common particles may determine
     - (common) grammar categories of word pairs,
     - the inflexion type that is affecting a word, and

  - phonological characteristics,
- and in the case of sentences in natural language
  - the common prefixes may be identified with new non-terminal grammar variables,
  - by clustering the set of sentences with respect to common prefixes the construction of a context-free grammar will emphasize rightmost derivations,
  - the synthesis of productions may be done by renaming variables, i. e. by introducing unit productions,
  - the synthesis of productions can be done in a more efficient way with some "partial learning": Some sentence classes are previously defined, correponding to sentencial forms, then the grammar rules are built in order to recognize the sentence examples and, at the same time, to preserve the sentential forms.

3 Induction of grammars based on pattern recognition

3.1 Substring comparison

Let A be a finite alphabet.

3.1.1 String codings

We will introduce the notion of *string matching* which will allow us to recognize frequent appearances of substrings in a sample set of strings.

A *matching* is a pair $(a, \mathbf{b}) \in A \times A^+$. The matching $(a, \mathbf{b})$ will be written also as $a \to \mathbf{b}$.

Given a subset $M \subset A \times A^+$ of matchings, the notion between words "$\mathbf{x}$ *matches* $\mathbf{y}$" in A^* is defined as follows:

1. If $a \to \mathbf{b}$ is a matching then $\mathbf{b}$ *matches* a.
2. If $\mathbf{b}_i$ matches a_i, $i = 1, \ldots, n$, then $\mathbf{b}_1 \cdots \mathbf{b}_n$ *matches* $a_1 \cdots a_n$.
3. If $a \to \mathbf{b}$ is a matching and $\mathbf{c}$ matches $\mathbf{b}$ then $\mathbf{c}$ *matches* a.

String	Code	Matchings
11011	$1a_1 0 a_2$	$a_1 \to 1$ $a_2 \to 1a_1$
111111	$1a_1 a_2 a_3$	$a_1 \to 1$ $a_2 \to 1a_1$ $a_3 \to a_2$
$ABCOABCVAB$	$ABCOa_1 V a_2$	$a_1 \to ABC$ $a_2 \to AB$
$ABCOABCVAC$	$ABCOa_1 V a_2 a_3$	$a_1 \to ABC$ $a_2 \to A$ $a_3 \to C$

Figure 1: Codings of strings: In the examples, the variable symbols are numbered in their order of appearance.

Let us assume that A' is a finite alphabet that extends A. Let us consider $A'' = A' - A$ as a set of *symbol variables*. An (A', A)-*matching* is a matching $a \to \mathbf{b}$ in A' such that $a \in A''$.

Let $\mathcal{F}(A') = \{\mathbf{x} \in (A')^* | i \neq j \Rightarrow x_i \neq x_j\}$ be the set of strings in $(A')^*$ without symbol repetitions.

A *coding* is a pair $(\mathbf{x}, \mathbf{y}) \in A^* \times \mathcal{F}(A')$ together with a set of (A', A)-matchings $M = \{a \to \mathbf{b}\}$ such that $\mathbf{x}$ matches $\mathbf{y}$, with respect to the given set of matchings, and A' is an extension of A.

A coding may be built directly for any given word $\mathbf{x}$: At each symbol x_i in $\mathbf{x}$ we check whether at the i-th position starts a substring that has been already been processed in $\mathbf{x}$, in this case, a new variable symbol is created, say a, and the longest string $\mathbf{b}$ starting at x_i that has appeared repeated in $\mathbf{x}$ is selected. Append in the code sequence the symbol a, add the matching $a \to \mathbf{b}$ and substitute any further appearance of $\mathbf{b}$ by a in $\mathbf{x}$. Repeat this process to the end of $\mathbf{x}$.

Examples: In Fig. 1 we present some examples of the proposed coding.

A more elaborated example is the following:

Let a sequence of matchings be defined iteratively,

$$\begin{array}{rcll} & a_0 \to 1 & & \text{and} \\ \forall k \geq 0 & : & a_{k+1} \to 1 a_1 \cdots a_k & \end{array}$$

Then $1^{(2^k)}$ matches with a_k, for all $k \geq 0$.

For any $n \geq 1$ let $n = [1n_{k-1} \cdots n_1 n_0]_2$ be its binary representation. The sequence $1^{(n)}$ consisting of exactly n 1's is coded by the sequence

$$1a_1 \cdots a_{k-2} a_{k-1} b_k b_{k+1} \cdots b_{2k-1}$$

considering that, for the variables b's, we have put the matchings

$$b_{k+j} \rightarrow \begin{cases} a_{k-j-1} & \text{if } n_{k-j-1} = 1 \\ nil & \text{if } n_{k-j-1} = 0 \end{cases}, j = 0, \ldots, k-1.$$

3.1.2 Grammar construction

For a given set of words $W \subset A^*$ we will build a CFG G_W such that $W \subset L(G_W)$, i.e. W is generated by the grammar G_W, as follows:

1. For each $\mathbf{w} \in W$ let $\mathbf{w}_c$ be its coding and let $R_{\mathbf{w}}$ be the corresponding set of matchings.

2. Rename the variables appearing in $\bigcup_{\mathbf{w} \in W} R_{\mathbf{w}}$ in such a way that any two matchings with the same consequence have equal antecedents:

$$\forall (a_1 \rightarrow \mathbf{b}_1), (a_2 \rightarrow \mathbf{b}_2) \in \bigcup_{\mathbf{w} \in W} R_{\mathbf{w}} : [\mathbf{b}_1 \ matches \ \mathbf{b}_2] \Rightarrow [a_1 = a_2].$$

3. Incorporate all the matchings as productions in G_W.

4. Apply, in an inverse way, the current rules in G_W into all the sample words, i.e. $\forall (a \rightarrow \mathbf{b}) \in \{\text{productions in } G_W\}, \forall \mathbf{v} \in W$: substitute any substring in $\mathbf{v}$ that matches with $\mathbf{b}$ by the symbol a.

5. Repeat the following till no new productions are produced:

   (a) For any pair of different words $\mathbf{w}, \mathbf{v} \in W$ do the following:

      i. Calculate the greatest substring $\mathbf{u}$ common to both $\mathbf{w}$ and $\mathbf{v}$.

      ii. Create a new variable symbol a, incorporate the production $a \rightarrow \mathbf{u}$ into G_W, and substitute by a any substring in any word of W that matches with $\mathbf{u}$.

iii. Continue the cycle with another pair of words.

(b) Check whether G_W has been modified to repeat the cycle above.

6. Create an initial variable symbol a_0 and for any word $\mathbf{w}$ in W (which has been rewritten by the above process), add the production $a_0 \to \mathbf{w}$ to G_W.

7. "G_W is the sought grammar."

Later, the grammar G_W may be simplified by eliminating unit productions. Some characteristics of our construction are:

- As we have renamed some variable symbols it may happen that recurrent productions had arouse.
- The construction of G_W is strongly dependent with the order in which the words in W are processed.
- In the above pseudocode, we begin with an "empty" set of productions in the grammar G_W. Actually we may begin the process with some "heuristic" rules already incorporated into G_W. In this way we can incorporate effectively some heuristics into the grammar G_W.

4 Conclusions

The procedures shown in this paper do not require any previous knowledge of the natural language structure. Thus, the formal grammar that they build may not necessarily be consistent with a formal grammar of natural language that follow the classical linguistic approach. Our procedures follow instead classical approaches in Pattern Recognition. We look toward procedures to conciliate these two different approaches.

The proposed methods are very procedural and effective. They allow to build a CFG that parses the sentences in the given corpus or the words in the given lexicon. However they give rise to unnatural grammar categories since they may form a concept that in spite that it embbeds string similarities it does not correspond to any "natural" category, in fact it may collects similar two strings in different "natural" categories.

5 Bibliography

[B] Bow: "Pattern Recognition: Applications to Large Data Set Problems", Marcel Dekker, New York, 1984

[HN] Hammond, Noonan: "Theoretical Morphology: Approaches to Modern Linguistics", Academic Press, 1988