Partial Learning Using Link Grammars Data

Erwan Moreau

Partial Learning Using Link Grammars Data

Erwan Moreau

LINA - FRE CNRS 2729 - Université de Nantes
2 rue de la Houssinière - BP 92208 - 44322 Nantes cedex 3
Erwan.Moreau@lina.univ-nantes.fr

Abstract. Kanazawa has shown that several non-trivial classes of categorial grammars are learnable in Gold's model. We propose in this article to adapt this kind of symbolic learning to natural languages. In order to compensate the combinatorial explosion of the learning algorithm, we suppose that a small part of the grammar to be learned is given as input. That is why we need some initial data to test the feasibility of the approach: link grammars are closely related to categorial grammars, and we use the English lexicon which exists in this formalism.

1 Introduction

In [1], Buszkowski has proposed an algorithm that learns rigid categorial grammars from structures. Kanazawa extended this result, and showed that several non-trivial classes of categorial grammars are learnable in Gold's model of identification in the limit [2]. Since these works, several results have been obtained, concerning learnability or unlearnability (in Gold's model) of some classes of categorial grammars (e.g. [3]). This kind of learning is symbolic, and relies on some properties of categorial grammars, in particular their total lexicalization.

In this article we propose to apply this learning method to natural language data. Such an application faces the problem of efficiency, due to the obviously big complexity of any natural language: actually, the fact that a class of grammars is learnable in Gold's model does not implie that learning can be done in a reasonable time. That is why we propose to learn in the framework of *partial learning*, where it is supposed that a part of the grammar to be learned is already known. This permits both to learn in a quite reasonable time, and without needing other information (like structures).

This idea is rather close from the one presented in [4], where semantic types are used, instead of structures, to learn efficiently. In a different framework, Hockenmaier has obtained good results in acquiring a natural language grammar from structures, using the Penn treebank [5].

Link grammars, introduced in [6], are close to categorial grammars. Thus it is possible to adapt learning algorithms to this formalism, as it has been shown in [7]. Furthermore, the authors Sleator and Temperley have built a link grammars lexicon which covers an important subset of the English language. This data will be very useful, since we need an *initial grammar* in our framework.

It should be emphasized that the problem is not simple, because the constraints imposed by symbolic learning and the fact that existing algorithms are exponential make it hard to obtain good results with a natural language. For this reason, we only study in this article the efficiency of the partial learning method, which is only one part of the problem. We will also discuss several problems appearing when using symbolic learning with complex data.

2 Link Grammars

Link grammars are defined by Sleator and Temperley in [6]. This is a rather simple formalism which is able to represent in a reliable way natural languages. This can be seen in the modelization that the authors provided for English in this sytem: their grammar deals with most of the linguistic phenomena in English, as it can be verified using their link grammar parser [8]. Their work is very interesting for our objectives for the following reasons:

- There exists a close relation between link grammars and categorial grammars: both are totally lexicalized and based on some kind of dependency notion.
- It is possible to express the link grammars model as a small set of rules using unification. This point is essential because several categorial grammars formalisms share this property, and it seems that this point plays an important role in learning algorithms based on Buszkowski's one.

Informally, a sentence is correct in link grammars if it is possible to *link* all words according to the links needed by each word, defined in the lexicon. Links represent syntactic relations between words.

A link grammar G is a tuple $\langle \Sigma, \mathcal{C}, \triangleright \rangle$, where Σ is the set of words, $\mathcal{C}$ is the set of *connectors*. A *disjunct* is a pair of lists denoted $\mathtt{d}(L,R)$, where L and R are lists of connectors, and $\triangleright$ is the relation assigning disjuncts to words: $w \triangleright d$ means that the disjunct d can be used with the word w.

Given a sentence $w_1, .., w_n$, a *linkage* is a set of links drawn above the sentence, each link connecting two words and being labelled with a connector. A linkage is valid if it satisfies the following conditions:

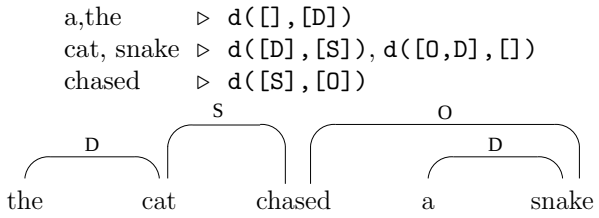
- *Planarity:* all links can be drawn above the words without crossing.
- *Connectivity:* given any couple of words (w_i, w_j) there exists a path of links between them.
- *Ordering:* for each word w_i there exists a disjunct d_i such that $w_i \triangleright d_i$ and d_i is satisfied. A disjunct $d_i = \mathtt{d}([L_1,..,L_l],\ [R_1,..,R_r])$ is satisfied if for each connector L_j (resp. R_j) there is a link labelled with the name of the connector coming to w_i from the left (resp. the right), and for any couple of connectors L_j, L_k (resp. R_j, R_k) with $j < k$, the words $w_{j'}, w_{k'}$ respectively connected to them verify $j' > k'$ (resp. $j' < k'$)[1].

[1] In other words when $j < k$ the word $w_{j'}$ is closer to w_i than $w_{k'}$. Remark: in the original definition the right list is written $[R_r, \ldots, R_1]$. We choose the reverse order to be coherent with the `cons(c,L)/nil` notation presented in section 2.1.

– *Exclusion:* No two links may connect the same pair of words.

These conditions are called the *meta-rules* of link grammars. It is shown in [6] that link grammars are equivalent to context-free grammars.

Example 1. With the following lexicon, the linkage below is valid:



2.1 Categorial Link Grammars

Link grammars are closely related to categorial grammars: both formalisms are totally lexicalized, and share the property to link words or constituents through a binary relation of dependency. However this dependency relation is different in the two systems. In link grammars, dependencies are not directed and apply only to words, contrary to categorial grammars where dependencies are directed and may apply to constituents of the sentence.

Classical categorial grammars. AB grammars (or classical categorial grammars) are the most simple formalism in categorial grammars. An AB grammar G is a tuple $\langle \Sigma, Pr, \triangleright \rangle$ where Σ is the set of words and Pr the set of primitive types. The set of types Tp is defined as the smallest set such that $Pr \subseteq Tp$ and $A/B, A\backslash B \in Tp$ for all $A, B \in Tp$. The relation $\triangleright \subseteq \Sigma \times Tp$ assigns one or several types to each word. The relation $\rightarrow$ is defined with the following universal rules:

$$A/B,\ B \rightarrow A \text{ (for any } A, B \in Tp)$$
$$B,\ B\backslash A \rightarrow A \text{ (for any } A, B \in Tp)$$

Let $\Rightarrow$ be the relation defined by $\alpha t_1 t_2 \beta \Rightarrow \alpha t_0 \beta$ if and only if $t_1 t_2 \rightarrow t_0$, and let $\Rightarrow^*$ be the reflexive and transitive closure of $\Rightarrow$. A sentence $x = w_1 w_2 ... w_n$ is correct for G (denoted $x \in L(G)$) if there is a sequence of types $\langle t_1 t_2 ... t_n \rangle$ such that $w_i \triangleright t_i$ for all i and $t_1 t_2 ... t_n \Rightarrow^* s$.

CLG: Definition. In order to emphasize the correspondence between link grammars and categorial grammars, we will use an intermediate formalism called *categorial link grammars* (CLG), where dependencies are directed and apply to constituents. This will permit to apply learning methods used with categorial grammars [2] to link grammars in a straightforward way. Actually, the CLG formalism is simply a different interpretation of (usual) link grammars rules. It is shown in [9] that CLG and basic link grammars rules are equivalent under the following restriction : each valid sentence must have a cycle-free linkage.

The CLG formalism takes exactly the same definition for grammars as the one presented above. But the rules governing derivations of sentences are expressed as the rewriting of a sequence of terms into another, like in classical categorial grammars. In order to emphasize that disjuncts are simple terms, we reformulate their definition in the following way: the set of connectors lists CL is defined as the smallest set such that `nil` $\in CL$ and for all `L'` $\in CL$ and `C` $\in \mathcal{C}$, `cons(C, L')` $\in CL$. The set of disjuncts, now called *types*, is defined as $Tp = \{\texttt{d(L,R)} \mid \texttt{L,R} \in CL\}$.[2]

Derivations with CLG. The two reduction rules between two types are

`d(L, cons(c, R))`, `d(cons(c, nil), nil)` $\rightarrow$ `d(L,R)` (with `c` $\in \mathcal{C}$ and `L,R` $\in CL$)
`d(nil, cons(c, nil))`, `d(cons(c, L), R)` $\rightarrow$ `d(L,R)` (with `c` $\in \mathcal{C}$ and `L,R` $\in CL$)

Let $\Rightarrow$ be the relation defined by $\alpha t_1 t_2 \beta \Rightarrow \alpha t_0 \beta$ if and only if $t_1 t_2 \rightarrow t_0$, with $\alpha, \beta \in Tp^*$ and $t_1, t_2, t_0 \in Tp$. $\Rightarrow^*$ is defined as the reflexive and transitive closure of $\Rightarrow$. Let $G = \langle \Sigma, \mathcal{C}, \triangleright \rangle$ be a grammar. A sentence $x = w_1 w_2 ... w_n$ is correct for G (denoted $x \in L(G)$) if there is a sequence of types $\langle t_1, t_2, ..., t_n \rangle$ such that $w_i \triangleright t_i$ for all i and $t_1 t_2 ... t_n \Rightarrow^*$ `d(nil, nil)`.

Example 2. With the lexicon defined in example 1, the sentence "The cat chased a snake" can be derived in the following way[3] :

	the	cat	chased	a	snake
	`d([],[D])`,	`d([D],[S])`,	`d([S],[O])`,	`d([],[D])`,	`d([D,O],[])`
$\Rightarrow$	`d([],[S])`,		`d([S],[O])`,	`d([],[D])`,	`d([D,O],[])`
$\Rightarrow$	`d([],[S])`,		`d([S],[O])`,		`d([O],[])`
$\Rightarrow$	`d([],[S])`,			`d([S],[])`	
$\Rightarrow$			`d([],[])`		

One can notice that a given linkage in basic link grammars can have several equivalent derivation trees in CLG. These ambiguities appear because links are not directed, and no order is required between connectors. The relation between CLG and AB grammars is detailed in [9].

2.2 The Link Grammars English Lexicon

One of the main reasons why link grammars are interesting is that the authors Sleator and Temperley have built such a grammar for English. The lexicon they provide contains approximately 59000 words, distributed into 1350 entries, each entry corresponding to a set of types. The grammar deals with an important number of linguistic phenomena in English, *"indicating that the approach may have practical uses as well as linguistic significance"* [6]. Furthermore, the link parser provided by the authors includes a file containing 928 sentences, labelled as correct or incorrect.

[2] One can see that this definition is coherent with the first one.

[3] *Remark:* in the following we will keep the notation `[`c_1`,`c_2`,..,`c_n`]` for connectors lists, easier to read than `cons(`c_1`,cons(`c_2`,..cons(`c_n`,nil)..))`. Nevertheless it is important to notice that there is no associativity in these terms.

In this section we outline how the link grammars lexicon for English is converted into the formalism of CLG. This transformation must take into account several complex features added to the basis of link grammars by their authors.

Multi-connectors. Iterations are represented using *multi-connectors*: Any connector `C` in a connectors list may be preceded by the operator `@`, meaning that this connector `@C` can be connected to several links `C`. For example, nouns can be given the type `d([@A,D],[S])`, allowing them to be preceded by any number of adjectives. This feature is included in the same way in CLG, by replacing the "basic rules" with these new ones (where `[@]` means that `@` is optionnal):

```
    d(L, cons([@]c, R)) , d(cons([@]c, nil), nil) → d(L,R)
    d(nil, cons([@]c, nil)) , d(cons([@]c, L), R) → d(L,R)
d(L, cons(@c, R)) , d(cons([@]c, nil), nil) → d(L,cons(@c, R))
d(nil, cons([@]c, nil)) , d(cons(@c, L), R) → d(cons(@c, L),R)
```

Subscripts. Subscripts are used to specialize connectors, in a similar way than feature structures in unification grammars. They are used to make the grammar easier to read and understand. For example, the subscripts `s` and `p` are assigned to nouns (and pronouns, determiners, etc.) to distinguish between singular and plural ones. In order to maintain the simplicity of the reduction rules, subscripts are converted into types that do not contain subscripts. This is achieved through a program that classifies all existing connectors (with their subscripts) in such a way that all connectors in a same class match the same set of connectors. When it is possible two classes are merged together, in order to minimize their number. Finally a new type is created for each such class.

The CLG parser. The correctness of the conversion has been tested by comparing results of parsing the example file using the converted grammar to the ones obtained using the original link parser. The CLG parser is a standard CYK-like parser running in $o(n^3)$, applying the binary reduction rules defined above. The file `4.0.batch` provided with the original link parser contains 928 sentences, in which 572 are correct and 356 are not. We extensively used this set of examples as a benchmark to test the CLG parser soundness and completness.

The CLG parser does not handle some "high-level" features of the original parser. Our objective being to provide a simple system based on a little set of rules, we did not translate these abilities into the CLG parser, and consequently modified the set of examples. Therefore sentences needing these features (conjonctions, post-processing rules, cost system and unknown words) have been removed from the example file : after this process, a set of 771 sentences is obtained, in which 221 are incorrect. Another important simplification concerns the problem of cycles. In the experiments detailed in this article, we consider a cycle-free version of the grammar which is simpler than the original one, but does not correctly handle all incorrect sentences : 38 incorrect sentences from the example file are parsed as correct, corresponding to an error rate of 4.9%. More details about the problem of cycles can be found in [9].

Example 3. Here are several sentences taken from the example file (incorrect ones are preceded with an asterisk):

What did John say he thought you should do
**What did John say did he think you should do*
To pretend that our program is usable in its current form would be silly
**Is that our program will be accepted likely*
The man there was an attempt to kill died

3 Partial Learning

3.1 Background

Buszkowski has proposed in [1] an algorithm, called RG, that learns *rigid* classical categorial grammars from *functor-argument structures* in Gold's model. A grammar is said *rigid* if every word in its lexicon is defined by only one type : this is of course a strong constraint on learnable languages. In particular, any slightly complex subset of any natural language does not fit into this definition. A *functor-argument structure* for a given sentence is some kind of parse tree for this sentence, where nodes are only labelled with FA or BA (identifiers for the two universal rules). Thus the nodes indicate which branch (i.e. constituant) should be used as functor (e.g. the left one, A/B, in the FA case) and which should be used as argument (e.g. the right one in the FA case). The fact that RG needs fa-structures as input is also an important drawback, because such structures are very precise, and therefore hardly available for applications.

Kanazawa has explored several extensions of this algorithm in [2], and has proved learnability in Gold's model of various classes of grammars obtained from these extensions. In particular, Kanazawa proposed an algorithm that learns k-valued[4] AB grammars from strings, and proved its convergence. Clearly, this extension avoids the two main constraints of the original algorithm. However, the problem of learning grammars (even rigid) without structures, as well as the one of learning k-valued grammars (even with structures) are shown by Costa-Florêncio to be NP-hard [10].

3.2 Partial Learning : Idea and Interest

As a consequence, it seems hard to apply this learning method to natural languages: the first algorithm is efficient but needs a lot of information as input and learns only a small class of grammars, and the other one is more general but unefficient. As a compromise between these two cases, We propose what we call *partial learning*: the algorithm takes flat strings as input, but it is also supposed that a part of the grammar to be learned, called the *initial grammar*, is already known (i.e. one knows the types assigned to some words in the lexicon). This information is intended to help building some part of the parse tree for

[4] A grammar is k-valued if each word in the lexicon is defined by at most k types.

the sentence given as input, thus replacing (to a certain extent) the information previously given in the structure. This information is important to avoid that too many possible trees make the algorithm unefficient.

Several arguments tend to make this hypothese rather plausible and well suited for natural languages applications:

- The total lexicalization of categorial grammars (or link grammars) was already an advantage for this kind of learning algorithms. Furthermore, it permits to consider that some words are known and other are unknown without any difficulty, since there can be no other kind of grammar rules.
- In the viewpoint of applications, the hypothese that a certain subset of the lexicon can be defined by advance seems more realistic than the hypothese that some complex information, like fa-structures, will be available with each sentence given to the algorithm.
- Finally, the efficiency (and the accuracy) of the algorithm in this framework mainly depends on the initial grammar: if there is a large number of known words in the examples, there are less possible parse trees and therefore the process goes faster. Applied to natural languages, Zipf's law says that, among the set of all words in a text, a little number of words covers a big part of the text (counting the number of occurences). Partial learning can benefit from this property in the following way: one has to build the initial grammar with this little set of frequent words, corresponding more or less to grammatical words (determiners, prepositions, etc.). This task is feasible because these words are not too numerous, and above all their number is bounded: for example, there may be new nouns or adjectives appearing in a natural language, but not a new determiner. Thanks to Zipf's law, these words are frequent, so knowing types for these words may be sufficiant to bound the number of possible parse trees for the sentences provided as input to the learning algorithm.

3.3 Algorithm

The fact that AB grammars and link grammars (when expressed in the CLG formalism) depend only on a small set of rules, and that these rules are based on unification is crucial in learning algorithms based on RG. This point is emphasized in the algorithm that we present here. The naive partial learning algorithm would consist in computing all possible parse trees for a sentence, and then see what among them are compatible with the types provided by the known words. Instead, the algorithm can benefit from the unification process by behaving like a parser: in the initialization step, distinct variables are assigned to unknown words. Then the incremental (CYK-like) parser computes, as usual, all possible types for each constituent, taking care of applying all substitutions that allow a reduction to the variables. At the end of the process, analyzing the set of applied substitutions provides all possible types for the unknown words.

The PL (Partial Learning) algorithm takes as input a sentence $w_1, .., w_n$ and an initial grammar G_0. This grammar contains rules of the form $w \triangleright t$, where t is

a type that can be used with word w. The algorithm returns the set of *general form grammars* : this means that any other grammar that accept the sentence according to the initial grammar G_0 is an instance of one of these grammars.

PL(G_0, $[w_1, w_2, .., w_n]$)
  $Lex \leftarrow \{(W, T) \mid (W \triangleright T) \in G_0\}$
  create an empty matrix $M[1..n, 1..n]$ % Initialization
  for $i \leftarrow 1$ to n do
    if $\exists T$ such that $(w_i \triangleright T) \in G_0$ then
      $M[i, i] \leftarrow \{(T, Id) \mid (w_i \triangleright T) \in G_0\}$
    else
      create a fresh variable V
      $Lex \leftarrow Lex \cup \{(w_i, V)\}$
      $M[i, i] \leftarrow \{(V, Id)\}$
    end if
  end for
  for $i \leftarrow 2$ to n do % Partial learning process (incremental parsing)
    for $j \leftarrow i - 1$ to 1 do
      for $k \leftarrow j$ to $i - 1$ do
        for each $(T_l, \sigma_l) \in M[j, k]$ do
          for each $(T_r, \sigma_r) \in M[k + 1, i]$ do
            let $\sigma_u = mgu(\sigma_l, \sigma_r)$
            for each rule $R \in \mathcal{R}$ do
              let $R = A_1 A_2 \rightarrow A_0$ % (where variables in all A_i are fresh)
              if $\exists \sigma_R = mgu(\{\{\sigma_u(T_l), A_1\}, \{\sigma_u(T_r), A_2\}\})$ then
                $M[j, i] \leftarrow M[j, i] \cup \{(\sigma_R(A_0), \sigma_R \circ \sigma_u \circ \sigma_l\}$
              end if
            end for
          end for
        end for
      end for
    end for
  end for
  $Res \leftarrow \emptyset$ % Apply possible substitutions
  for each $(T, \sigma) \in M[1, n]$ do
    if $\exists \tau$ such that $\tau(T) = S$ then
      $Res \leftarrow Res \cup \{(\tau \circ \sigma)(Lex)\}$
    end if
  end for
  return Res
End PL

- Id is the identity substitution. $\circ$ is the composition of two substitutions.
- A unifier for a family of sets $\mathcal{A}$ is a substitution σ such that for every $\mathcal{A}_i \in \mathcal{A}$ and every couple of types $t, t' \in \mathcal{A}_i$: $\sigma(t) = \sigma(t')$. The *most general unifier (mgu)* is the (unique) unifier σ_u such that for any other unifier σ there exists a substitution τ such that $\sigma = \tau \circ \sigma_u$.
- Since σ_u is defined as the MGU of σ_l and σ_r, we have $(\sigma_u \circ \sigma_l) = (\sigma_u \circ \sigma_r)$.

– $\mathcal{R}$ is the set of universal rules: in the case of AB grammars, one will obtain for example $A/B = A_1, B = A_2$ and $A = A_0$ for the first rule, FA (variables are A and B). In the case of CLG, one would have `d(`L`, cons(`c`, `R`))` $= A_1$, `d(cons(`c`, nil), nil)`$= A_2$ and `d(`L`,`R`)` $= A_0$ for the first rule (with variables L, R, c). This algorithm works for both formalisms.

The "CYK-like" form of this algorithm should not hide that the algorithm remains exponential in the general case. This algorithm describes the process only for one sentence: the question to know what should be done with the whole set of *general form grammars* is discussed in section 4.3.

4 Experiments and Discussion

In this section we explore feasibility of partial learning using link grammars data. A prototype has been implemented (coded in SWI Prolog), and some tests have been realized. The only part of the problem explored here is the possibility to "parse with variables" in a reasonable time.

4.1 The CLG Partial Learning Prototype: Parsing with Variables

The CLG partial learning prototype is based on the CLG parser presented in 2.2. Clearly, the Prolog language is well suited to deal with terms containing variables during the parsing: actually, substitutions are not stored in a data structure like this is presented in the algorithm above, but simply obtained by Prolog unification. The process remains deterministic however: types are copied whenever necesary in order to build lists of types in the matrix, and thus avoiding backtrack through the matrix.

Apart from some non essential optimization, the program has an important feature that avoids a large part of the combinatorial explosion. This feature consists in factorizing types that are "structurally equivalent": two types t_1 and t_2 are structurally equivalent if there exists two substitution σ_1 and σ_2 such that $\sigma_1(t_1) = t_2$ and $\sigma_2(t_2) = t_1$ (i.e. there is only a renaming of variables between the two types). The case where several structurally equivalent types belong to the same cell of the matrix is frequent, so this saves a lot of time and space. But this mechanism also requires that the composition of substitutions be no longer computed along the whole process, otherwise the next level would have to re-develop the types. That is why only the substitution used to transform a type at one level (one reduction) is stored: in order to obtain the composition (which gives what types are possible for unknown words), the program makes a second pass after the incremental parsing.

Some tests have been done using the example file provided with the link grammar parser [8]. First the set of examples has been filtered in order to keep only correct sentences. The tests consist in removing a certain set of words from the lexicon, so that these words will be considered as unknown.

4.2 Results

The sample used contains 537 sentences and 4765 words (size of the sample), but only 1064 different words (size of the lexicon). The most frequent words are *the* (270), *I* (155), *is* (122), *to* (121). In these tests we compare times taken for partial learning for different rates of unknown words in the lexicon and sample.

In this first test[5] we remove randomly a set of words from the lexicon. Results are shown in table 1: one can see that the rate of unknown words in the sample is similar to the rate of unknown (removed) words in the lexicon.

"UW" stands for "unknown words": the second column (lexicon) indicates the number of UW within the set of words, whereas the third one (sample) indicates the number of occurencies of UW in the sentences.

UW (lexicon)	UW (sample)	Total time	UW per sentence (min ; avg ; max)	time per sentence (min ; avg ; max) (sec)
0 (0%)	0 (0%)	27 min	0 ; 0 ; 0	0.1 ; 3.1 ; 14.7
106 (10%)	407 (8.5%)	75 min	0 ; 0.7 ; 4	0.1 ; 8.3 ; 333
211 (20%)	870 (18.2%)	3.5 h	0 ; 1.5 ; 6	0.1 ; 24 ; 1985
264 (25%)	1106 (23.2%)	3.8 h	0 ; 1.8 ; 7	0.1 ; 24.9 ; 1115
317 (30%)	1481 (31.1%)	Failure (out of memory)		

Table 1.

In order to simulate the idea that the initial grammar should be built using a small set of frequent words, in this second test we remove only the less frequent words from the lexicon. Therefore, Zipf's law permits to remove a large part of the lexicon without having too many unknown words in the sample (table 2).

UW (lexicon)	UW (sample)	Total time	UW per sentence (min ; avg ; max)	time per sentence (min ; avg ; max) (sec)
0 (0%)	0 (0%)	27 min	0 ; 0 ; 0	0.1 ; 3.1 ; 14.7
211 (20%)	211 (4.4%)	45 min	0 ; 0.3 ; 4	0.1 ; 5.0 ; 173
422 (40%)	422 (8.8%)	78 min	0 ; 0.6 ; 5	0.1 ; 8.7 ; 620
634 (60%)	727 (15.2%)	2.6 h	0 ; 1.1 ; 7	0.1 ; 17.3 ; 1844
845 (80%)	1302 (27.3%)	5.1 h	0 ; 2.0 ; 9	0.1 ; 34.1 ; 1893

Table 2.

These tests show that the partial learning process can run in an almost reasonable time (and without overloading memory) with up to 25 to 30% unknown words in the sample. It is interesting to see that we only need around 20% of the words defined in the lexicon to obtain such a rate, thanks to Zipf's law.

Of course, the data is a bit too small to consider that these tests guarantee that the algorithm works in any situation with a certain rate of unknown words.

[5] These tests were done on a PIII 1GHz.

However, link grammars data for English are a good approximation of a natural language, and were not intended to serve for this kind of application. So these tests tend to show that the partial learning method can be a realistic approach to apply symbolic learning to natural languages.

4.3 Problems and Future Work

Although the partial learning part seems to work quite well, we do not have yet "real" learning results to present in this paper. Actually, several problems make hard the adaptation of symbolic learning algorithms to "almost real" applications to natural languages. In a practical viewpoint, the main question is: what do we want to obtain as a result ? Indeed, the algorithm can return a set of grammars, but this set may be too big in complex cases (and therefore take too much time to be computed). It would also be possible to use a different representation for the types, like in the original link grammar where types are logical formulas using some operators (*and*, *or*, etc.). But symbolic learning requires that types be in a normal form, in order to be able to recognize two identical types.

Another (more theoretical) important problem concerns the constraints in Gold's model. Actually, Gold's model require that only one hypothese (grammar) be proposed for any set of examples. Kanazawa has explored this problem for categorial grammars, in a rather theoretical perspective [2]. In the case of AB grammars, as soon as one is interested in learning k-valued grammars with $k > 1$ or in learning without full fa-structures there are (generally) more than one grammar solution. Kanazawa shows that it is possible to compute a minimal grammar, and this is why the classes observed are learnable in Gold's model. But again the algorithm is exponential and unusable in any complex case. In the perspective of application to natural language, one can not use this solution. That is why the algorithm we propose does not really *learn*, in the sense of Gold. However, the reason of this constraint in Gold's model is that it would be too easy for an algorithm to *learn* if it was allowed to make an infinitely growing number of hypotheses.

We propose the following solution to bound the number of solutions without restricting too much the generative power of the learnable classes of grammars in this framework. Following the hypothese that it is rather easy to build an initial grammar containing most possible types but only a small set of words, it is possible to consider that a set of *syntactic classes* is defined: each class contains a set of fixed types, and the role of the learning algorithm would be to correctly assign a class to each word. This method is already used in part-of-speech taggers (like [11]). As the link grammar lexicon is made of 1350 entries, each containing a set of types corresponding to a set of words, one can hope that this method can be adapted in the CLG framework.

5 Conclusion

In conclusion, several important problems must be solved before considering obtaining useful results for a natural language: the form of the data, the formalism

itself, the size of the initial grammar are some very important parameters. Also, solutions should be found for some simplifications made in this experiment, for the problem of cycles (due to link grammars formalism) and the one of conjonctions.

Actually, nothing ensures that this learning method will be able to provide valuable results. Nevertheless, this first experiment shows that this hypothese could hold: it seems that a "good" initial grammar, together with Zipf's law, can compensate for the lack of efficiency of existing theoretical algorithms. Furthermore, the fact that this algorithm does not need important information (like fa-structures) in input makes it more suited to real applications. But the main interest in symbolic learning is the accuracy: probably this kind of learning can not be very efficient, but the fact that the correctness of the answer always hold can be an interesting feature for some applications.

References

1. Buszkowski, W., Penn, G.: Categorial grammars determined from linguistic data by unification. Technical Report TR-89-05, Department of Computer Science, University of Chicago (1989)
2. Kanazawa, M.: Learnable classes of categorial grammars. Cambridge University Press (1998)
3. Bonato, R., Retoré, C.: Learning rigid lambek grammars and minimalist grammars from structured sentences. In Popelínský, L., Nepil, M., eds.: Proceedings of the 3d Workshop on Learning Language in Logic, Strasbourg, France (2001) 23–34
4. Sofronie, D.D., Tellier, I., Tommasi, M.: A tool for language learning based on categorial grammars and semantic information. In Adriaans, P., Fernau, H., van Zaanen, M., eds.: Grammatical Inference: Algorithms and Applications; 6th International Colloquium, ICGI 2002. Volume 2484 of LNCS/LNAI., Springer (2002) 303–305
5. Hockenmaier, J.: Data and models for statistical parsing with Combinatory Categorial Grammar. PhD thesis, School of Informatics, The University of Edinburgh (2003)
6. Sleator, D.D.K., Temperley, D.: Parsing english with a link grammar. Technical Report CMU-CS-TR-91-126, Carnegie Mellon University, Pittsburgh, PA (1991)
7. Béchet, D.: k-valued link grammars are learnable from strings. In: Proceedings Formal Grammars 2003. (2003) 9–18
8. Temperley, D., Sleator, D., Lafferty, J.: Link grammar. http://hyper.link.cs.cmu.edu/link/ (1991)
9. Moreau, E.: From link grammars to categorial grammars. In: Proceedings of Categorial Grammars 2004, Montpellier, France. (2004) 31–45
10. Costa Florêncio, C.: Consistent Identification in the Limit of Rigid Grammars from Strings is NP-hard. In Adriaans, P., Fernau, H., van Zaanen, M., eds.: Grammatical Inference: Algorithms and Applications 6th International Colloquium: ICGI 2002. Volume 2484 of Lecture Notes in Artificial Intelligence., Springer-Verlag (2002) 49–62
11. Brill, E.: A Corpus-Based Approach to Language Learning. PhD thesis, Computer and Information Science, University of Pennsylvania (1993)