OPTIMIZING THE COMPUTATIONAL LEXICALIZATION OF LARGE GRAMMARS

Christian JACQUEMIN
Institut de Recherche en Informatique de Nantes (IRIN)
IUT de Nantes – 3, rue du Maréchal Joffre
F–44041 NANTES Cedex 01 – FRANCE
e–mail : jacquemin@irin.iut–nantes.univ–nantes.fr

Abstract

The computational lexicalization of a grammar is the optimization of the links between lexicalized rules and lexical items in order to improve the quality of the bottom-up filtering during parsing. This problem is **NP**-complete and untractable on large grammars. An approximation algorithm is presented. The quality of the suboptimal solution is evaluated on real-world grammars as well as on randomly generated ones.

Introduction

Lexicalized grammar formalisms and more specifically Lexicalized Tree Adjoining Grammars (*LTAGs*) give a lexical account of phenomena which cannot be considered as purely syntactic (Schabes *et al*, 1990). A formalism is said to be lexicalized if it is composed of structures or rules associated with each lexical item and operations to derive new structures from these elementary ones. The choice of the lexical anchor of a rule is supposed to be determined on purely linguistic grounds. This is the *linguistic* side of lexicalization which links to each lexical head a set of minimal and complete structures. But lexicalization also has a *computational* aspect because parsing algorithms for lexicalized grammars can take advantage of lexical links through a two-step strategy (Schabes and Joshi, 1990). The first step is the selection of the set of rules or elementary structures associated with the lexical items in the input sentence[1]. In the second step, the parser uses the rules filtered by the first step.

The two kinds of anchors corresponding to these two aspects of lexicalization can be considered separately :

- The linguistic anchors are used to access the grammar, update the data, gather together items with similar structures, organize the grammar into a hierarchy...
- The computational anchors are used to select the relevant rules during the first step of parsing and to improve computational and conceptual tractability of the parsing algorithm.

Unlike linguistic lexicalization, computational anchoring concerns any of the lexical items found in a rule and is only motivated by the quality of the induced filtering. For example, the systematic linguistic anchoring of the rules describing *"N_{metal} alloy"* to their head noun *"alloy"* should be avoided and replaced by a more distributed lexicalization. Then, only a few rules *"N_{metal} alloy"* will be activated when encountering the word *"alloy"* in the input.

In this paper, we investigate the problem of the optimization of computational lexicalization. We study how to choose the computational anchors of a lexicalized grammar so that the distribution of the rules on to the lexical items is the most uniform possible

[1] The computational anchor of a rule should not be optional (viz included in a disjunction) to make sure that it will be encountered in any string derived from this rule.

with respect to rule weights. Although introduced with reference to *LTAGs*, this optimization concerns any portion of a grammar where rules include one or more potential lexical anchors such as *Head Driven Phrase Structure Grammar* (Pollard and Sag, 1987) or *Lexicalized Context-Free Grammar* (Schabes and Waters, 1993).

This algorithm is currently used to good effect in *FASTR* a unification-based parser for terminology extraction from large corpora (Jacquemin, 1994). In this framework, terms are represented by rules in a lexicalized constraint-based formalism. Due to the large size of the grammar, the quality of the lexicalization is a determining factor for the computational tractability of the application. *FASTR* is applied to automatic indexing on industrial data and lays a strong emphasis on the handling of term variations (Jacquemin and Royauté, 1994).

The remainder of this paper is organized as follows. In the following part, we prove that the problem of the Lexicalization of a Grammar is **NP**-complete and hence that there is no better algorithm known to solve it than an exponential exhaustive search. As this solution is untractable on large data, an approximation algorithm is presented which has a computational-time complexity proportional to the cubic size of the grammar. In the last part, an evaluation of this algorithm on real-world grammars of 6,622 and 71,623 rules as well as on randomly generated ones confirms its computational tractability and the quality of the lexicalization.

The Problem of the Lexicalization of a Grammar

Given a lexicalized grammar, this part describes the problem of the optimization of the computational lexicalization. The solution to this problem is a lexicalization function (henceforth a lexicalization) which associates to each grammar rule one of the lexical items it includes (its lexical anchor). A lexicalization is optimized to our sense if it induces an optimal preprocessing of the grammar. Preprocessing is intended to activate the rules whose lexical anchors are in the input and make all the possible filtering of these rules before the proper parsing algorithm. Mainly, preprocessing discards the rules selected through lexicalization including at least one lexical item which is not found in the input.

The first step of the optimization of the lexicalization is to assign a *weight* to each rule. The weight is assumed to represent the cost of the corresponding rule during the preprocessing. For a given lexicalization, the *weight of a lexical item* is the sum of the weights of the rules linked to it. The weights are chosen so that a uniform distribution of the rules on to the lexical items ensures an optimal preprocessing. Thus, the problem is to find an anchoring which achieves such a uniform distribution.

The weights depend on the physical constraints of the system. For example, the weight is the number of nodes if the memory size is the critical point. In this case, a uniform distribution ensures that the rules linked to an item will not require more than a given memory space. The weight is the number of terminal or non-terminal nodes if the computational cost has to be minimized. Experimental measures can be performed on a test set of rules in order to determine the most accurate weight assignment.

Two simplifying assumptions are made :

- The weight of a rule does not depend on the lexical item to which it is anchored.
- The weight of a rule does not depend on the other rules simultaneously activated.

The second assumption is essential for settling a tractable problem. The first assumption can be avoided at the cost of a more complex representation. In this case, instead of having a unique weight, a rule must have as many weights as potential lexical anchors. Apart from this modification, the algorithm that will be presented in the next part remains much the same than in the case of a single weight. If the first assumption is removed, data about the frequency of the items in corpora can be accounted for. Assigning smaller weights to rules when they are anchored to rare items will

make the algorithm favor the anchoring to these items. Thus, due to their rareness, the corresponding rules will be rarely selected.

Illustration Terms, compounds and more generally idioms require a lexicalized syntactic representation such as *LTAGs* to account for the syntax of these lexical entries (Abeillé and Schabes, 1989). The grammars chosen to illustrate the problem of the optimization of the lexicalization and to evaluate the algorithm consist of idiom rules such as $\mathcal{G}$:

$$\mathcal{G} = \{\textit{from time to time, high time, high grade, high grade steel}\}$$

Each rule is represented by a pair (w_i, A_i) where w_i is the weight and A_i the set of potential anchors. If we choose the total number of words in an idiom as its weight and its non-empty words as its potential anchors, $\mathcal{G}$ is represented by the following *grammar* :

$$G_1 = \{a = (4, \{time\}), b = (2, \{high,time\}), c = (2, \{grade,high\}), d = (3, \{grade,high,steel\})\}$$

We call *vocabulary*, the union V of all the sets of potential anchors A_i. Here, $V = \{grade, high, steel, time\}$. A *lexicalization* is a function λ associating a lexical anchor to each rule.

Given a *threshold* θ, the membership problem called the Lexicalization of a Grammar (*LG*) is to find a lexicalization so that the weight of any lexical item in V is less than or equal to θ. If $\theta \geq 4$ in the preceding example, *LG* has a solution λ :

$$\lambda(a) = time,\ \lambda(b) = \lambda(c) = high,\ \lambda(d) = steel$$

If $\theta \leq 3$, *LG* has no solution.

Definition of the *LG* Problem

$G = \{(w_i, A_i)\}$ $(w_i \in \mathbb{Q}^+, A_i$ finite sets)

$V = \{v_i\} = \cup A_i$; $\theta \in \mathbb{Q}^+$

(1) $LG \doteq \{ (V, G, \theta, \lambda) \mid$ where $\lambda : G \rightarrow V$ is a total function anchoring the rules so that $(\forall (w, A) \in G)\ \lambda((w, A)) \in A$ and $(\forall v \in V) \sum_{\lambda((w,A)) = v} w \leq \theta \}$

The associated optimization problem is to determine the lowest value θ_{opt} of the threshold θ so that there exists a solution $(V, G, \theta_{opt}, \lambda)$ to *LG*. The solution of the optimization problem for the preceding example is $\theta_{opt} = 4$.

Lemma *LG is in NP.*

It is evident that checking whether a given lexicalization is indeed a solution to *LG* can be done in polynomial time. The relation R defined by (2) is polynomially decidable :

(2) $R(V, G, \theta, \lambda) \doteq$ [**if** λ: $V \rightarrow G$ and $(\forall v \in V) \sum_{\lambda((w,A)) = v} w \leq \theta$ **then** true **else** false]

The weights of the items can be computed through matrix products : a matrix for the grammar and a matrix for the lexicalization. The size of any lexicalization λ is linear in the size of the grammar. As $(V, G, \theta, \lambda) \in LG$ if and only if [R(V, G, θ, λ)] is true, *LG* is in **NP**. ■

Theorem *LG is NP-complete.*

Bin Packing (*BP*) which is **NP**-complete is polynomial-time Karp reducible to *LG*. *BP* (Baase, 1986) is the problem defined by (3) :

(3) $BP \doteq \{ (R, \{R_1, \ldots, R_k\}) \mid$ where $R = \{r_1, \ldots, r_n\}$ is a set of n positive rational numbers less than or equal to 1 and $\{R_1, \ldots, R_k\}$ is a partition of R (k bins in which the r_js are packed) such that $(\forall i \in \{1, \ldots, k\}) \sum_{r \in R_i} r \leq 1$.

First, any instance of *BP* can be represented as an instance of *LG*. Let $(R, \{R_1, \ldots, R_k\})$ be an instance of *BP* it is transformed into the instance (V, G, θ, λ) of *LG* as follows :

(4) $V = \{v_1, \ldots, v_k\}$ a set of k symbols, $\theta = 1$, $G = \{(r_1, V), \ldots, (r_n, V)\}$ and $(\forall i \in \{1, \ldots, k\})\ (\forall j \in \{1, \ldots, n\})$ $\lambda((r_j, v)) = v_i \Leftrightarrow r_j \in R_i$

For all $i \in \{1, \ldots, k\}$ and $j \in \{1, \ldots, n\}$, we consider the assignment of r_j to the bin R_i of *BP* as the anchoring of the rule (r_j, V) to the item v_i of *LG*. If $(R, \{R_1, \ldots, R_k\}) \in BP$ then :

$$(5)\quad (\forall i \in \{1, \ldots, k\}) \sum_{r \in R_i} r \leq 1$$
$$\Leftrightarrow (\forall i \in \{1, \ldots, k\}) \sum_{\lambda((r, v)) = v_i} r \leq 1$$

Thus $(V, G, 1, \lambda) \in LG$. Conversely, given a solution $(V, G, 1, \lambda)$ of LG, let $R_i \doteq \{r_j \in R \mid \lambda((r_j, V)) = v_i\}$ for all $i \in \{1, \ldots, k\}$. Clearly $\{R_1, \ldots, R_k\}$ is a partition of R because the lexicalization is a total function and the preceding formula ensures that each bin is correctly loaded. Thus $(R, \{R_1, \ldots, R_k\}) \in BP$. It is also simple to verify that the transformation from BP to LG can be performed in polynomial time. ■

The optimization of an **NP**-complete problem is **NP**-complete (Sommerhalder and van Westrhenen, 1988), then the optimization version of LG is **NP**-complete.

An Approximation Algorithm for LG

This part presents and evaluates an n^3-time approximation algorithm for the LG problem which yields a suboptimal solution close to the optimal one. The first step is the 'easy' anchoring of rules including at least one rare lexical item to one of these items. The second step handles the 'hard' lexicalization of the remaining rules including only common items found in several other rules and for which the decision is not straightforward. The discrimination between these two kinds of items is made on the basis of their *global weight GW* (6) which is the sum of the weights of the rules which are not yet anchored and which have this lemma as potential anchor. V_λ and G_λ are subsets of V and G which denote the items and the rules not yet anchored. The ws and θ are assumed to be integers by multiplying them by their lowest common denominator if necessary.

$$(6)\quad (\forall v \in V_\lambda)\ GW(v) = \sum_{(w, A) \in G_\lambda,\ v \in A} w$$

Step 1 : 'Easy' Lexicalization of Rare Items This first step of the optimization algorithm is also the first step of the exhaustive search. The value of the minimal threshold θ_{min} given by (7) is computed by dividing the sum of the rule weights by the number of lemmas ($\lceil x \rceil$ stands for the smallest integer greater than or equal to x and $|V_\lambda|$ stands for the size of the set V_λ) :

$$(7)\quad \theta_{min} = \left\lceil \frac{\sum_{(w, A) \in G_\lambda} w}{|V_\lambda|} \right\rceil \text{ where } |V_\lambda| \neq 0$$

All the rules which include a lemma with a global weight less than or equal to θ_{min} are anchored to this lemma. When this linking is achieved in a non-deterministic manner, θ_{min} is recomputed. The algorithm loops on this lexicalization, starting it from scratch every time, until θ_{min} remains unchanged or until all the rules are anchored. The output value of θ_{min} is the minimal threshold such that LG has a solution and therefore is less than or equal to θ_{opt}. After Step 1, either each rule is anchored or all the remaining items in V_λ have a global weight strictly greater than θ_{min}. The algorithm is shown in Figure 1.

Step 2 : 'Hard' Lexicalization of Common Items During this step, the algorithm repeatedly removes an item from the remaining vocabulary and yields the anchoring of this item. The item with the lowest global weight is handled first because it has the smallest combination of anchorings and hence the probability of making a wrong choice for the lexicalization is low. Given an item, the candidate rules with this item as potential anchor are ranked according to :

1 The highest priority is given to the rules whose set of potential anchors only includes the current item as non-anchored item.
2 The remaining candidate rules taken first are the ones whose potential anchors have the highest global weights (items found in several other non-anchored rules).

The algorithm is shown in Figure 2. The output of Step 2 is the suboptimal computational lexicalization λ of the whole grammar and the associated threshold θ_{subopt}.

Both steps can be optimized. Useless computation is avoided by watching the *capital*

of weight C defined by (8) with θ - θ_{min} during Step 1 and θ - θ_{subopt} during Step 2 :

$$(8)\quad C = \theta . |V_\lambda| - \sum_{(w, A) \in G_\lambda} w$$

C corresponds to the weight which can be lost by giving a weight $W(\varpi)$ which is strictly less than the current threshold θ. Every time an anchoring to a unit ϖ is completed, C is reduced from θ - $W(\varpi)$. If C becomes negative in either of both steps, the algorithm will fail to make the lexicalization of the grammar and must be started again from Step 1 with a higher value for θ.

Input V, G

Output $\theta_{min}, V_\lambda, G_\lambda, \lambda : (G - G_\lambda) \rightarrow (V - V_\lambda)$

$$\theta_{min} \leftarrow \left\lceil \frac{\sum_{(w, A) \in G} w}{|V|} \right\rceil ;$$

Step1 **repeat**

$G_\lambda \leftarrow G$; $V_\lambda \leftarrow V$;

for each $v \in V$ such as $GW(v) \leq \theta_{min}$ **do**

for each $(w, A) \in G$ such as $v \in A$ and $\lambda((w, A))$ not yet defined **do**

$\lambda((w, A)) \leftarrow v$;

$G_\lambda \leftarrow G_\lambda - \{(w, A)\}$;

update $GW(v)$;

end

$V_\lambda \leftarrow V_\lambda - \{v\}$;

end

$$\theta'_{min} \leftarrow \left\lceil \frac{\sum_{(w, A) \in G_\lambda} w}{|V_\lambda|} \right\rceil ;$$

if(($\theta'_{min} \leq \theta_{min}$ and ($(\forall v \in V_\lambda)\ GW(v) > \theta_{min}$)) or $G_\lambda = \emptyset$)

then exit repeat ;

$\theta_{min} \leftarrow \theta'_{min}$;

until(false) ;

Figure 1: Step 1 of the approximation algorithm.

Input $\theta_{min}, V, G, V_\lambda, G_\lambda, \lambda : (G - G_\lambda) \rightarrow (V - V_\lambda)$

Output $\theta_{subopt}, \lambda : G \rightarrow V$

Step2 $\theta_{subopt} \leftarrow \theta_{min}$;

repeat

;; anchoring the rules with only ϖ as
;; free potential anchor ($\varpi \in V_\lambda$ with
;; the lowest global weight)

$\varpi \leftarrow v_i$;

$G_{\varpi,1} \leftarrow \{ (w, A) \in G_\lambda \mid A \cap V_\lambda = \{\varpi\} \}$;

if ($\sum_{(w, A) \in G_{\varpi,1}} w < \theta_{subopt}$)

then $\theta_{min} \leftarrow \theta_{min} + 1$; goto *Step1* ;

for each $(w, A) \in G_{\varpi,1}$ **do**

$\lambda((w, A)) \leftarrow \varpi$;

$G_\lambda \leftarrow G_\lambda - \{ (w, A) \}$;

end

$G_{\varpi,2} \leftarrow \{(w, A) \in G_\lambda ; A \cap V_\lambda \supset \{\varpi\}\}$;

$W(\varpi) \leftarrow \sum_{\lambda((w, A)) = \varpi} w$;

;; ranking[2] $G_{\varpi,2}$ and anchoring

for($i \leftarrow 1; i \leq |G_{\varpi,2}| ; i \leftarrow i + 1$) **do**

$(w, A) \leftarrow r^{-1}(i)$;; i^{th} ranked by r

if($W(\varpi) + w > \theta_{min}$)

then exit for ;

$W(\varpi) \leftarrow W(\varpi) + w$;

$\lambda((w, A)) \leftarrow \varpi$;

$G_\lambda \leftarrow G_\lambda - \{(w, A)\}$;

end

$V_\lambda \leftarrow V_\lambda - \{\varpi\}$;

until ($G_\lambda = \emptyset$) ;

Figure 2: Step 2 of the approximation algorithm.

[2] The ranking function r: $G_{\varpi,2} \rightarrow \{1, \ldots\ |G_{\varpi,2}|\}$ is such that $r((w, A)) > r((w', A'))$
$\Leftrightarrow \min_{v \in A \cap V_\lambda - \{\varpi\}} W(v) \geq \min_{v' \in A' \cap V_\lambda - \{\varpi\}} W(v')$

Example[3] The algorithm has been applied to a test grammar G_2 obtained from 41 terms with 11 potential anchors. The algorithm fails in making the lexicalization of G_2 with the minimal threshold $\theta_{min} = 12$, but achieves it with $\theta_{subopt} = 13$. This value of θ_{subopt} can be compared with the optimal one by running the exhaustive search. There are 2^{32} ($\cong 4\ 10^9$) possible lexicalizations among which 35,336 are optimal ones with a threshold of 13. This result shows that the approximation algorithm brings forth one of the optimal solutions which only represent a proportion of $8\ 10^{-6}$ of the possible lexicalizations. In this case the optimal and the suboptimal threshold coincide.

Time-Complexity of the Approximation Algorithm A grammar G on a vocabulary V can be represented by a $|G| \times |V|$-matrix of Boolean values for the set of potential anchors and a $1 \times |G|$-matrix for the weights. In order to evaluate the complexity of the algorithms as a function of the size of the grammar, we assume that $|V|$ and $|G|$ are of the same order of magnitude n. Step 1 of the algorithm corresponds to products and sums on the preceding matrixes and takes $O(n^3)$ time. The worst-case time-complexity for Step 2 of the algorithm is also $O(n^3)$ when using a naive $O(n^2)$ algorithm to sort the items and the rules by decreasing priority. In all, the time required by the approximation algorithm is proportional to the cubic size of the grammar.

This order of magnitude ensures that the algorithm can be applied to large real-world grammars such as terminological grammars. On a Sparc 2, the lexicalization of a terminological grammar composed of 6,622 rules and 3,256 words requires 3 seconds (real time) and the lexicalization of a very large terminological grammar of 71,623 rules and 38,536 single words takes 196 seconds. The two grammars used for these experiment were generated from two lists of terms provided by the documentation center INIST/CNRS.

[3] The exhausitve grammar and more details about this example and the computations of the following section are in (Jacquemin, 1991).

Evaluation of the Approximation Algorithm

Bench Marks on Artificial Grammars In order to check the quality of the lexicalization on different kinds of grammars, the algorithm has been tested on eight randomly generated grammars of 4,000 rules having from 2 to 10 potential anchors (Table 1). The lexicon of the first four grammars is 40 times smaller than the grammar while the lexicon of the last four ones is 4 times smaller than the grammar (this proportion is close to the one of the real-world grammar studied in the next subsection). The eight grammars differ in their distribution of the items on to the rules. The uniform distribution corresponds to a uniform random choice of the items which build the set of potential anchors while the Gaussian one corresponds to a choice taking more frequently some items. The higher the parameter s, the flatter the Gaussian distribution.

The last two columns of Table 1 give the minimal threshold θ_{min} after Step 1 and the suboptimal threshold θ_{subopt} found by the approximation algorithm. As mentioned when presenting Step 1, the optimal threshold θ_{opt} is necessarily greater than or equal to θ_{min} after Step 1. Table 1 reports that the suboptimal threshold θ_{subopt} is not over 2 units greater than θ_{min} after Step 1. The suboptimal threshold yielded by the approximation algorithm on these examples has a high quality because it is at worst 2 units greater than the optimal one.

A Comparison with Linguistic Lexicalization on a Real-World Grammar This evaluation consists in applying the algorithm to a natural language grammar composed of 6,622 rules (terms from the domain of metallurgy provided by INIST/CNRS) and a lexicon of 3,256 items. Figure 3 depicts the distribution of the weights with the natural linguistic lexicalization. The frequent head words such as *alloy* are heavily loaded because of the numerous terms in *N–alloy* with *N* being a name of metal. Conversely, in Figure 4 the distribution of the weights from the approximation algorithm is much more

uniform. The maximal weight of an item is 241 with the linguistic lexicalization while it is only 34 with the optimized lexicalization. The threshold after Step 1 being 34, the suboptimal threshold yielded by the approximation algorithm is equal to the optimal one.

Lexicon size	Distribution of the items on the rules	θ_{min} before Step 1	θ_{min} after Step 1	θ_{subopt} suboptimal threshold
100	uniform	143	143	143
100	Gaussian ($s = 30$)	141	143	144
100	Gaussian ($s = 20$)	141	260	261
100	Gaussian ($s = 10$)	141	466	468
1,000	uniform	15	15	16
1,000	Gaussian ($s = 30$)	14	117	118
1,000	Gaussian ($s = 20$)	15	237	238
1,000	Gaussian ($s = 10$)	14	466	467

Table 1: Bench marks of the approximation algorithm on eight randomly generated grammars.

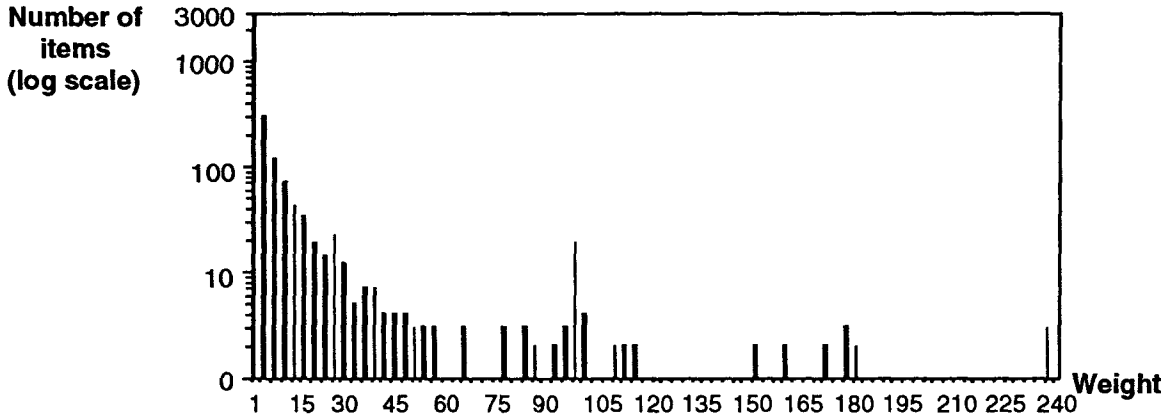

Figure 3: Distribution of the weights of the lexical items with the lexicalization on head words.

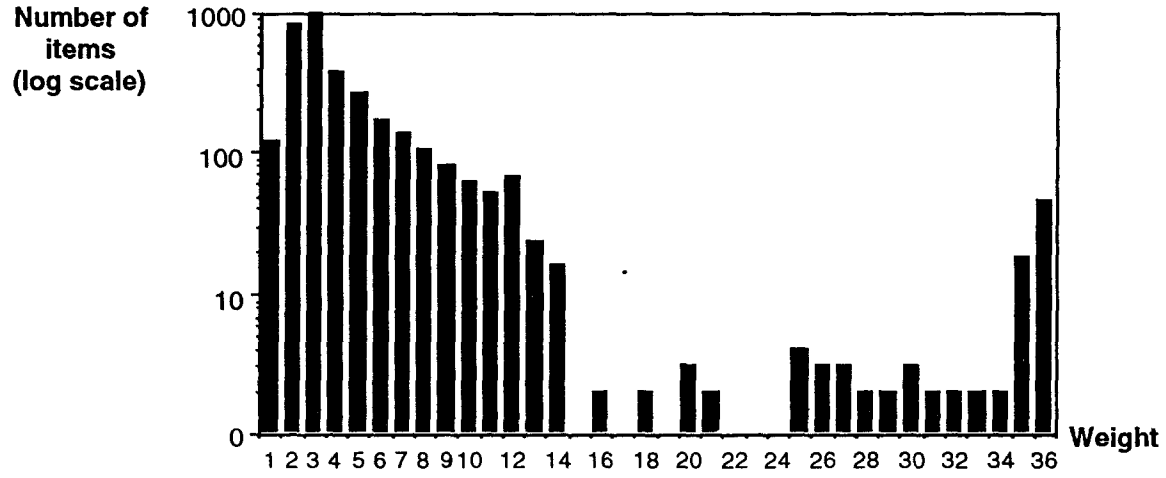

Figure 4: Distribution of the weights of the lexical items with the optimized lexicalization.

Conclusion

As mentioned in the introduction, the improvement of the lexicalization through an optimization algorithm is currently used in *FASTR* a parser for terminological extraction through NLP techniques where terms are represented by lexicalized rules. In this framework as in top-down parsing with *LTAGs* (Schabes and Joshi, 1990), the first phase of parsing is a filtering of the rules with their anchors in the input sentence. An unbalanced distribution of the rules on to the lexical items has the major computational drawback of selecting an excessive number of rules when the input sentence includes a common head word such as *"alloy"* (127 rules have *"alloy"* as head). The use of the optimized lexicalization allows us to filter 57% of the rules selected by the linguistic lexicalization. This reduction is comparable to the filtering induced by linguistic lexicalization which is around 85% (Schabes and Joshi, 1990). Correlatively the parsing speed is multiplied by 2.6 confirming the computational saving of the optimization reported in this study.

There are many directions in which this work could be refined and extended. In particular, an optimization of this optimization could be achieved by testing different weight assignments in correlation with the parsing algorithm. Thus, the computational lexicalization would fasten both the preprocessing and the parsing algorithm.

Acknowledgments

I would like to thank Alain Colmerauer for his valuable comments and a long discussion on a draft version of my PhD dissertation. I also gratefully acknowledge Chantal Enguehard and two anonymous reviewers for their remarks on earlier drafts. The experiments on industrial data were done with term lists from the documentation center INIST/CNRS.

REFERENCES

Abeillé, Anne, and Yves Schabes. 1989. Parsing Idioms in Tree Adjoining Grammars. In *Proceedings, 4th Conference of the European Chapter of the Association for Computational Linguistics (EACL'89)*, Manchester, UK.

Baase, Sara. 1978. *Computer Algorithms.* Addison Wesley, Reading, MA.

Jacquemin, Christian. 1991. *Transformations des noms composés.* PhD Thesis in Computer Science, Université of Paris 7. *Unpublished.*

Jacquemin, Christian. 1994. FASTR : A unification grammar and a parser for terminology extraction from large corpora. In *Proceedings, IA-94*, Paris, EC2, June 1994.

Jacquemin, Christian and Jean Royauté. 1994. Retrieving terms and their variants in a lexicalized unification-based framework. In *Proceedings, 17th Annual International ACM SIGIR Conference (SIGIR'94)*, Dublin, July 1994.

Pollard, Carl and Ivan Sag. 1987. *Information-Based Syntax and Semantics. Vol 1: Fundamentals.* CSLI, Stanford, CA.

Schabes, Yves, Anne Abeillé, and Aravind K. Joshi. 1988. Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammar. In *Proceedings, 12th International Conference on Computational Linguistics (COLING'88)*, Budapest, Hungary.

Schabes, Yves and Aravind K. Joshi. 1990. Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammar. In Masaru Tomita, editor, *Current Issues in Parsing Technologies.* Kluwer Academic Publishers, Dordrecht.

Schabes, Yves and Richard C. Waters. 1993. Lexicalized Context-Free Grammars. In *Proceedings, 31st Meeting of the Association for Computational Linguistics (ACL'93)*, Columbus, Ohio.

Sommerhalder, Rudolph and S. Christian van Westrhenen. 1988. *The Theory of Computability: Programs, Machines, Effectiveness and Feasibility.* Addison-Wesley, Reading, MA.