Semi-supervised Dependency Parsing using Lexical Affinities

Seyed Abolghasem Mirroshandel[†,⋆] **Alexis Nasr**[†] **Joseph Le Roux**[◇]
[†]Laboratoire d'Informatique Fondamentale de Marseille- CNRS - UMR 7279
Université Aix-Marseille, Marseille, France
[◇]LIPN, Université Paris Nord & CNRS,Villetaneuse, France
[⋆]Computer Engineering Department, Sharif university of Technology, Tehran, Iran
(ghasem.mirroshandel@lif.univ-mrs.fr, alexis.nasr@lif.univ-mrs.fr,
leroux@univ-paris13.fr)

Abstract

Treebanks are not large enough to reliably model precise lexical phenomena. This deficiency provokes attachment errors in the parsers trained on such data. We propose in this paper to compute lexical affinities, on large corpora, for specific lexico-syntactic configurations that are hard to disambiguate and introduce the new information in a parser. Experiments on the French Treebank showed a relative decrease of the error rate of 7.1% Labeled Accuracy Score yielding the best parsing results on this treebank.

1 Introduction

Probabilistic parsers are usually trained on treebanks composed of few thousands sentences. While this amount of data seems reasonable for learning syntactic phenomena and, to some extent, very frequent lexical phenomena involving closed parts of speech (POS), it proves inadequate when modeling lexical dependencies between open POS, such as nouns, verbs and adjectives. This fact was first recognized by (Bikel, 2004) who showed that bilexical dependencies were barely used in Michael Collins' parser.

The work reported in this paper aims at a better modeling of such phenomena by using a raw corpus that is several orders of magnitude larger than the treebank used for training the parser. The raw corpus is first parsed and the computed *lexical affinities* between lemmas, in specific lexico-syntactic configurations, are then injected back in the parser. Two outcomes are expected from this procedure, the first is, as mentioned above, a better modeling of bilexical dependencies and the second is a method to adapt a parser to new domains.

The paper is organized as follows. Section 2 reviews some work on the same topic and highlights their differences with ours. In section 3, we describe the parser that we use in our experiments and give a detailed description of the frequent attachment errors. Section 4 describes how lexical affinities between lemmas are calculated and their impact is then evaluated with respect to the attachment errors made by the parser. Section 5 describes three ways to integrate the lexical affinities in the parser and reports the results obtained with the three methods.

2 Previous Work

Coping with lexical sparsity of treebanks using raw corpora has been an active direction of research for many years.

One simple and effective way to tackle this problem is to put together words that share, in a large raw corpus, similar linear contexts, into word *clusters*. The word occurrences of the training treebank are then replaced by their cluster identifier and a new parser is trained on the transformed treebank. Using such techniques (Koo et al., 2008) report significative improvement on the Penn Treebank (Marcus et al., 1993) and so do (Candito and Seddah, 2010; Candito and Crabbé, 2009) on the French Treebank (Abeillé et al., 2003).

Another series of papers (Volk, 2001; Nakov and Hearst, 2005; Pitler et al., 2010; Zhou et al., 2011) directly model word co-occurrences. Co-occurrences of pairs of words are first collected in a

Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics, pages 777–785,
Jeju, Republic of Korea, 8-14 July 2012.

raw corpus or internet n-grams. Based on the counts produced, lexical affinity scores are computed. The detection of pairs of words co-occurrences is generally very simple, it is either based on the direct adjacency of the words in the string or their co-occurrence in a window of a few words. (Bansal and Klein, 2011; Nakov and Hearst, 2005) rely on the same sort of techniques but use more sophisticated patterns, based on simple paraphrase rules, for identifying co-occurrences.

Our work departs from those approaches by the fact that we do not extract the lexical information directly on a raw corpus, but we first parse it and then extract the co-occurrences on the parse trees, based on some predetermined lexico-syntactic patterns. The first reason for this choice is that the linguistic phenomena that we are interested in, such as as PP attachment, coordination, verb subject and object can range over long distances, beyond what is generally taken into account when working on limited windows. The second reason for this choice was to show that the performances that the NLP community has reached on parsing, combined with the use of confidence measures allow to use parsers to extract accurate lexico-syntactic information, beyond what can be found in limited annotated corpora.

Our work can also be compared with self training approaches to parsing (McClosky et al., 2006; Suzuki et al., 2009; Steedman et al., 2003; Sagae and Tsujii, 2007) where a parser is first trained on a treebank and then used to parse a large raw corpus. The parses produced are then added to the initial treebank and a new parser is trained. The main difference between these approaches and ours is that we do not directly add the output of the parser to the training corpus, but extract precise lexical information that is then re-injected in the parser. In the self training approach, (Chen et al., 2009) is quite close to our work: instead of adding new parses to the treebank, the occurrence of simple interesting subtrees are detected in the parses and introduced as new features in the parser.

The way we introduce lexical affinity measures in the parser, in 5.1, shares some ideas with (Anguiano and Candito, 2011), who modify some attachments in the parser output, based on lexical information. The main difference is that we only take attachments that appear in an n-best parse list into account, while they consider the first best parse and compute all potential alternative attachments, that may not actually occur in the n-best forests.

3 The Parser

The parser used in this work is the second order graph based parser (McDonald et al., 2005; Kübler et al., 2009) implementation of (Bohnet, 2010). The parser was trained on the French Treebank (Abeillé et al., 2003) which was transformed into dependency trees by (Candito et al., 2009). The size of the treebank and its decomposition into train, development and test sets is represented in table 1.

	nb of sentences	nb of words
FTB_TRAIN	9 881	278 083
FTB_DEV	1 239	36 508
FTB_TEST	1 235	36 340

Table 1: Size and decomposition of the French Treebank

The part of speech tagging was performed with the MELT tagger (Denis and Sagot, 2010) and lemmatized with the MACAON tool suite (Nasr et al., 2011). The parser gave state of the art results for parsing of French, reported in table 2.

	pred. POS tags		gold POS tags	
	punct	no punct	punct	no punct
LAS	88.02	90.24	88.88	91.12
UAS	90.02	92.50	90.71	93.20

Table 2: Labeled and unlabeled accuracy score for automatically predicted and gold POS tags with and without taking into account punctuation on FTB_TEST.

Figure 1 shows the distribution of the 100 most common error types made by the parser. In this figure, x axis shows the error types and y axis shows the error ratio of the related error type ($\frac{\text{number of errors of the specific type}}{\text{total number of errors}}$). We define an error type by the POS tag of the governor and the POS tag of the dependent. The figure presents a typical Zipfian distribution with a low number of frequent error types and a large number of unfrequent error types. The shape of the curve shows that concentrating on some specific frequent errors in order to increase the parser accuracy is a good strategy.

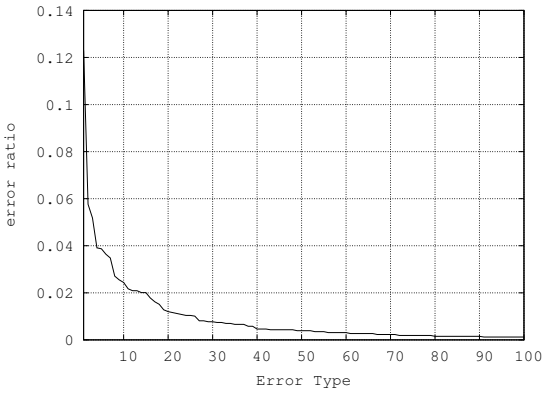

Figure 1: Distribution of the types of errors

Table 3 gives a finer description of the most common types of error made by the parser. Here we define more precise patterns for errors, where some lexical values are specified (for prepositions) and, in some cases, the nature of the dependency is taken into account. Every line of the table corresponds to one type of error. The first column describes the error type. The second column indicates the frequency of this type of dependency in the corpus. The third one displays the accuracy for this type of dependency (the number of dependencies of this type correctly analyzed by the parser divided by the total number of dependencies of this type). The fourth column shows the contribution of the errors made on this type of dependency to the global error rate. The last column associates a name with some of the error types that will prove useful in the remainder of the paper to refer to the error type.

Table 3 shows two different kinds of errors that impact the global error rate. The first one concerns very common dependencies that have a high accuracy but, due to their frequency, hurt the global error rate of the parser. The second one concerns low frequency, low accuracy dependency types. Lines 2 and 3, respectively attachment of the preposition *à* to a verb and the subject dependency illustrate such a contrast. They both impact the total error rate in the same way (2.53% of the errors). But the first one is a low frequency low accuracy type (respectively 0.88% and 69.11%) while the second is a high frequency high accuracy type (respectively 3.43% and 93.03%). We will see in 4.2.2 that our method behaves quite differently on these two types of error.

dependency	freq.	acc.	contrib.	name
N→N	1.50	72.23	2.91	
V → à	0.88	69.11	2.53	VaN
V—suj → N	3.43	93.03	2.53	SBJ
N → CC	0.77	69.78	2.05	NcN
N → de	3.70	92.07	2.05	NdeN
V → de	0.66	74.68	1.62	VdeN
V—obj → N	2.74	90.43	1.60	OBJ
V → en	0.66	81.20	1.24	
V → pour	0.46	67.78	1.10	
N → ADJ	6.18	96.60	0.96	ADJ
N → à	0.29	70.64	0.72	NaN
N → pour	0.12	38.64	0.67	
N → en	0.15	47.69	0.57	

Table 3: The 13 most common error types

4 Creating the Lexical Resource

The lexical resource is a collection of tuples $\langle C, g, d, s\rangle$ where C is a lexico-syntactic configuration, g is a lemma, called the governor of the configuration, d is another lemma called the dependent and s is a numerical value between 0 and 1, called the lexical affinity score, which accounts for the strength of the association between g and d in the context C. For example the tuple $\langle (V, g) \stackrel{obj}{\rightarrow} (N, d), eat, oyster, 0.23\rangle$ defines a simple configuration $(V, g) \stackrel{obj}{\rightarrow} (N, d)$ that is an object dependency between verb g and noun d. When replacing variables g and d in C respectively with eat and $oyster$, we obtain the fully specified lexico syntactic pattern$(V, eat) \stackrel{obj}{\rightarrow} (N, oyster)$, that we call an instantiated configuration. The numerical value 0.23 accounts for how much *eat* and *oyster* like to co-occur in the verb-object configuration. Configurations can be of arbitrary complexity but they have to be generic enough in order to occur frequently in a corpus yet be specific enough to model a precise lexico syntactic phenomenon. The context $(*, g) \stackrel{*}{\rightarrow} (*, d)$, for example is very generic but does not model a precise linguistic phenomenon, as selectional preferences of a verb, for example. Moreover, configurations need to be error-prone. In the perspective of increasing a parser performances, there is no point in computing lexical affinity scores between words that appear in a configuration for which

the parser never makes mistakes.

The creation of the lexical resource is a three stage process. The first step is the definition of configurations, the second one is the collection of raw counts from the machine parsed corpora and the third one is the computation of lexical affinities based on the raw counts. The three steps are described in the following subsection while the evaluation of the created resource is reported in subsection 4.2.

4.1 Computing Lexical Affinities

A set of 9 configurations have been defined. Their selection is a manual process based on the analysis of the errors made by the parser, described in section 3, as well as on the linguistic phenomena they model. The list of the 9 configurations is described in Table 4. As one can see on this table, configurations are usually simple, made up of one or two dependencies. Linguistically, configurations OBJ and SBJ concern subject and object attachments, configuration ADJ is related to attachments of adjectives to nouns and configurations NdeN, VdeN, VaN, and NaN indicate prepositional attachments. We have restricted ourselves here to two common French prepositions *à* and *de*. Configurations NcN and VcV deal respectively with noun and verb coordination.

Name	Description		
OBJ	(V, g)	$\overset{obj}{\rightarrow}$	(N, d)
SBJ	(V, g)	$\overset{subj}{\rightarrow}$	(N, d)
ADJ	(N, g)	$\rightarrow$	ADJ
NdeN	(N, g)	$\rightarrow (P, de) \rightarrow$	(N, d)
VdeN	(V, g)	$\rightarrow (P, de) \rightarrow$	(N, d)
NaN	(N, g)	$\rightarrow (P, \grave{a}) \rightarrow$	(N, d)
VaN	(V, g)	$\rightarrow (P, \grave{a}) \rightarrow$	(N, d)
NcN	(N, g)	$\rightarrow (CC, *) \rightarrow$	(N, d)
VcV	(V, g)	$\rightarrow (CC, *) \rightarrow$	(V, d)

Table 4: List of the 9 configurations.

The computation of the number of occurrences of an instantiated configuration in the corpus is quite straightforward, it consists in traversing the dependency trees produced by the parser and detect the occurrences of this configuration.

At the end of the counts collection, we have gathered for every lemma l its number of occurrences as governor (resp. dependent) of configuration C in the corpus, noted $\mathcal{C}(C, l, *)$ (resp. $\mathcal{C}(C, *, l)$), as well as the number of occurrences of configuration C with lemma l_g as a governor and lemma l_d as a dependent, noted $\mathcal{C}(C, l_g, l_d)$. We are now in a position to compute the score $s(C, l_g, l_d)$. This score should reflect the tendency of l_g and l_d to appear together in configuration C. It should be maximal if whenever l_g occurs as the governor of configuration C, the dependent position is occupied by l_d and, symmetrically, if whenever l_d occurs as the dependent of configuration C, the governor position is occupied by l_g. A function that conforms such a behavior is the following:

$$s(C, l_g, l_d) = \frac{1}{2}\left(\frac{\mathcal{C}(C, l_g, l_d)}{\mathcal{C}(C, l_g, *)} + \frac{\mathcal{C}(C, l_g, l_d)}{\mathcal{C}(C, *, l_d)}\right)$$

it takes its values between 0 (l_g and l_d never co-occur) and 1 (g and d always co-occur). This function is close to pointwise mutual information (Church and Hanks, 1990) but takes its values between 0 and 1.

CORPUS	Sent. nb.	Tokens nb.
AFP	1 024 797	31 486 618
EST REP	1 103 630	19 635 985
WIKI	1 592 035	33 821 460
TOTAL	3 720 462	84 944 063

Table 5: sizes of the corpora used to gather lexical counts

4.2 Evaluation

Lexical affinities were computed on three corpora of slightly different genres. The first one, is a collection of news report of the French press agency *Agence France Presse*, the second is a collection of newspaper articles from a local French newspaper : *l'Est Républicain*. The third one is a collection of articles from the French Wikipedia. The size of the different corpora are detailed in table 5. The corpus was first POS tagged, lemmatized and parsed in order to get the 50 best parses for every sentence. Then the lexical resource was built, based on the 9 configurations described in table 4.

The lexical resource has been evaluated on FTB_DEV with respect to two measures: coverage

and correction rate, described in the next two sections.

4.2.1 Coverage

Coverage measures the instantiated configurations present in the evaluation corpus that are in the resource. The results are presented in table 6. Every line represents a configuration, the second column indicates the number of different instantiations of this configuration in the evaluation corpus, the third one indicates the number of instantiated configurations that were actually found in the lexical resource and the fourth column shows the coverage for this configuration, which is the ratio third column over the second. Last column represents the coverage of the training corpus (the lexical resource is extracted on the training corpus) and the last line represents the same quantities computed on all configurations.

Table 6 shows two interesting results: firstly the high variability of coverage with respect to configurations, and secondly the low coverage when the lexical resource is computed on the training corpus, this fact being consistent with the conclusions of (Bikel, 2004). A parser trained on a treebank cannot be expected to reliably select the correct governor in lexically sensitive cases.

Conf.	occ.	pres.	cov.	T cov.
OBJ	1017	709	0.70	0.21
SBJ	1210	825	0.68	0.24
ADJ	1791	1239	0.69	0.33
NdeN	1909	1287	0.67	0.31
VdeN	189	107	0.57	0.16
NaN	123	61	0.50	0.20
VaN	422	273	0.65	0.23
NcN	220	55	0.25	0.10
VcV	165	93	0.56	0.04
Σ	7046	4649	0.66	0.27

Table 6: Coverage of the lexical resource over FTB_DEV.

4.2.2 Correction Rate

While coverage measures how many instantiated configurations that occur in the treebank are actually present in the lexical resource, it does not measure if the information present in the lexical resource can actually help correcting the errors made by the parser.

We define *Correction Rate* (CR) as a way to approximate the usefulness of the data. Given a word d present in a sentence S and a configuration C, the set of all potential governors of d in configuration C, in all the n-best parses produced by the parser is computed. This set is noted $\mathcal{G} = \{g_1, \ldots, g_j\}$. Let us note G_L the element of $\mathcal{G}$ that maximizes the lexical affinity score. When the lexical resource gives no score to any of the elements of $\mathcal{G}$, G_L is left unspecified.

Ideally, $\mathcal{G}$ should not be the set of governors in the n-best parses but the set of all possible governors for d in sentence S. Since we have no simple way to compute the latter, we will content ourselves with the former as an approximation of the latter.

Let us note G_H the governor of d in the (first) best parse produced and G_R the governor of d in the correct parse. CR measures the effect of replacing G_H with G_L.

We have represented in table 7 the different scenarios that can happen when comparing G_H, G_R and G_L.

$G_H = G_R$	$G_L = G_R$ or G_L unspec.	CC
	$G_L \neq G_R$	CE
$G_H \neq G_R$	$G_L = G_R$	EC
	$G_L \neq G_R$ or G_L unspec.	EE
	$G_R \notin \mathcal{G}$	NA

Table 7: Five possible scenarios when comparing the governor of a word produced by the parser (G_H), in the reference parse (G_R) and according to the lexical resource (G_L).

In scenarios CC and CE, the parser did not make a mistake (the first letter, C, stands for correct). In scenario CC, the lexical affinity score was compatible with the choice of the parser or the lexical resource did not select any candidate. In scenario CE, the lexical resource introduced an error. In scenarios EC and EE, the parser made an error. In EC, the error was corrected by the lexical resource while in EE, it wasn't. Either because the lexical resource candidate was not the correct governor or it was unspecified. The last case, NA, indicates that the correct governor does not appear in any of the n-best parses. Technically this case could be integrated in EE (an error made by the parser was not corrected by the lexical resource) but we chose to keep it apart

since it represents a case where the right solution could not be found in the n-best parse list (the correct governor is not a member of set $\mathcal{G}$).

Let's note n_S the number of occurrences of scenario S for a given configuration. We compute CR for this configuration in the following way:

$$\begin{aligned} CR &= \frac{\text{old error number - new error number}}{\text{old error number}} \\ &= \frac{n_{EC} - n_{CE}}{n_{EE} + n_{EC} + n_{NA}} \end{aligned}$$

When CR is equal to 0, the correction did not have any impact on the error rate. When CR > 0, the error rate is reduced and if CR < 0 it is increased[1].

CR for each configuration is reported in table 8. The counts of the different scenarios have also been reported.

Conf.	n_{CC}	n_{CE}	n_{EC}	n_{EE}	n_{NA}	CR
OBJ	992	30	51	5	17	0.29
SBJ	1131	35	61	16	34	0.23
ADJ	2220	42	16	20	6	-0.62
NdeN	2083	93	42	44	21	-0.48
VdeN	150	2	49	1	13	0.75
NaN	89	5	21	10	2	0.48
VaN	273	19	132	8	11	0.75
NcN	165	17	12	31	12	-0.09
VcN	120	21	14	11	5	-0.23
Σ	7223	264	398	146	121	0.20

Table 8: Correction Rate of the lexical resource with respect to FTB_DEV.

Table 8 shows very different results among configurations. Results for PP attachments VdeN, VaN and NaN are quite good (a CR of 75% for a given configuration, as VdeN indicates that the number of errors on such a configuration is decreased by 25%). It is interesting to note that the parser behaves quite badly on these attachments: their accuracy (as reported in table 3) is, respectively 74.68, 69.1 and 70.64. Lexical affinity helps in such cases. On the other hand, some attachments like configuration ADJ and NdeN, for which the parser showed very good accuracy (96.6 and 92.2) show very poor performances. In such cases, taking into account lexical affinity creates new errors.

[1]One can note, that contrary to coverage, CR does not measure a characteristic of the lexical resource alone, but the lexical resource combined with a parser.

On average, using the lexical resource with this simple strategy of systematically replacing G_H with G_L allows to decrease by 20% the errors made on our 9 configurations and by 2.5% the global error rate of the parser.

4.3 Filtering Data with Ambiguity Threshold

The data used to extract counts is noisy: it contains errors made by the parser. Ideally, we would like to take into account only non ambiguous sentences, for which the parser outputs a single parse hypothesis, hopefully the good one. Such an approach is obviously doomed to fail since almost every sentence will be associated to several parses. Another solution would be to select sentences for which the parser has a high confidence, using confidence measures as proposed in (Sánchez-Sáez et al., 2009; Hwa, 2004). But since we are only interested in some parts of sentences (usually one attachment), we don't need high confidence for the whole sentence. We have instead used a parameter, defined on single dependencies, called the *ambiguity measure*.

Given the n best parses of a sentence and a dependency δ, present in at least one of the n best parses, let us note $\mathcal{C}(\delta)$ the number of occurrences of δ in the n best parse set. We note $AM(\delta)$ the ambiguity measure associated to δ. It is computed as follows:

$$AM(\delta) = 1 - \frac{\mathcal{C}(\delta)}{n}$$

An ambiguity measure of 0 indicates that δ is non ambiguous in the set of the n best parses (the word that constitutes the dependent in δ is attached to the word that constitutes the governor in δ in all the n-best analyses). When n gets large enough this measure approximates the non ambiguity of a dependency in a given sentence.

Ambiguity measure is used to filter the data when counting the number of occurrences of a configuration: only occurrences that are made of dependencies δ such that $AM(\delta) \leq \tau$ are taken into account. τ is called the ambiguity threshold.

The results of coverage and CR given above were computed for τ equal to 1, which means that, when collecting counts, all the dependencies are taken into account whatever their ambiguity is. Table 9 shows coverage and CR for different values of τ. As expected, coverage decreases with τ. But, interest-

ingly, decreasing τ, from 1 down to 0.2 has a positive influence on CR. Ambiguity threshold plays the role we expected: it allows to reduce noise in the data, and corrects more errors.

	$\tau = 1.0$	$\tau = 0.4$	$\tau = 0.2$	$\tau = 0.0$
	cov/CR	cov/CR	cov/CR	cov/CR
OBJ	0.70/0.29	0.58/0.36	0.52/0.36	0.35/0.38
SBJ	0.68/0.23	0.64/0.23	0.62/0.23	0.52/0.23
ADJ	0.69/-0.62	0.61/-0.52	0.56/-0.52	0.43/-0.38
NdeN	0.67/-0.48	0.58/-0.53	0.52/-0.52	0.38/-0.41
VdeN	0.57/0.75	0.44/0.73	0.36/0.73	0.20/0.30
NaN	0.50/0.48	0.34/0.42	0.28/0.45	0.15/0.48
VaN	0.65/0.75	0.50/0.8	0.41/0.80	0.26/0.48
NcN	0.25/-0.09	0.19/0	0.16/0.02	0.07/0.13
VcV	0.56/-0.23	0.42/-0.07	0.28/0.03	0.08/0.07
Avg	0.66/0.2	0.57/0.23	0.51/0.24	0.38/0.17

Table 9: Coverage and Correction Rate on FTB_DEV for several values of ambiguity threshold.

5 Integrating Lexical Affinity in the Parser

We have devised three methods for taking into account lexical affinity scores in the parser. The first two are post-processing methods, that take as input the n-best parses produced by the parser and modify some attachments with respect to the information given by the lexical resource. The third method introduces the lexical affinity scores as new features in the parsing model. The three methods are described in 5.1, 5.2 and 5.3. They are evaluated in 5.4.

5.1 Post Processing Method

The post processing method is quite simple. It is very close to the method that was used to compute the Correction Rate of the lexical resource, in 4.2.2: it takes as input the n-best parses produced by the parser and, for every configuration occurrence C found in the first best parse, the set ($\mathcal{G}$) of all potential governors of C, in the n-best parses, is computed and among them, the word that maximizes the lexical affinity score (G_L) is identified.

Once G_L is identified, one can replace the choice of the parser (G_H) with G_L. This method is quite crude since it does not take into account the confidence the parser has in the solution proposed. We observed, in 4.2.2 that CR was very low for configurations for which the parser achieves good accuracy. In order to introduce the parser confidence in the final choice of a governor, we compute $\mathcal{C}(G_H)$ and $\mathcal{C}(G_L)$ which respectively represent the number of times G_H and G_L appear as the governor of configuration C. The choice of the final governor, noted $\hat{G}$, depends on the ratio of $\mathcal{C}(G_H)$ and $\mathcal{C}(G_L)$. The complete selection strategy is the following:

1. if $G_H = G_L$ or G_L is unspecified, $\hat{G} = G_H$.
2. if $G_H \neq G_L$, $\hat{G}$ is determined as follows:

$$\hat{G} = \begin{cases} G_H & \text{if } \frac{\mathcal{C}(G_H)}{\mathcal{C}(G_L)} > \alpha \\ G_L & \text{otherwise} \end{cases}$$

where α is a coefficient that is optimized on the development data set.

We have reported, in table 10 the values of CR, for the 9 different features, using this strategy, for $\tau = 1$. We do not report the values of CR for other values of τ since they are very close to each other. The table shows several noticeable facts. First, the new strategy performs much better than the former one (crudely replacing G_H by G_L), the value of CR increased from 0.2 to 0.4, which means that the errors made on the nine configurations are now decreased by 40%. Second, CR is now positive for every configuration: the number of errors is decreased for every configuration.

Conf.	OBJ	SUJ	ADJ	NdeN	VdeN
CR	0.45	0.46	0.14	0.05	0.73
Conf.	NaN	VaN	NcN	VcV	Σ
CR	0.12	0.8	0.12	0.1	0.4

Table 10: Correction Rate on FTB_DEV when taking into account parser confidence.

5.2 Double Parsing Method

The post processing method performs better than the naive strategy that was used in 4.2.2. But it has an important drawback: it creates inconsistent parses. Recall that the parser we are using is based on a second order model, which means that the score of a dependency depends on some neighboring ones. Since with the post processing method only a subset of the dependencies are modified, the resulting parse is inconsistent: the score of some dependencies is computed on the basis of other dependencies that have been modified.

In order to compute a new optimal parse tree that preserves the modified dependencies, we have used a technique proposed in (Mirroshandel and Nasr, 2011) that modifies the scoring function of the parser in such a way that the dependencies that we want to keep in the parser output get better scores than all competing dependencies.

The double parsing method is therefore a three stage method. First, sentence S is parsed, producing the n-best parses. Then, the post processing method is used, modifying the first best parse. Let's note $\mathcal{D}$ the set of dependencies that were changed in this process. In the last stage, a new parse is produced, that preserves $\mathcal{D}$.

5.3 Feature Based Method

In the feature based method, new features are added to the parser that rely on lexical affinity scores. These features are of the following form: $\langle C, l_g, l_d, \delta_C(s)\rangle$, where C is a configuration number, s is the lexical affinity score ($s = s(C, l_g, l_d)$) and $\delta_c(\cdot)$ is a discretization function.

Discretization of the lexical affinity scores is necessary in order to fight against data sparseness. In this work, we have used Weka software (Hall et al., 2009) to discretize the scores with unsupervised binning. Binning is a simple process which divides the range of possible values a parameter can take into subranges called bins. Two methods are implemented in Weka to find the optimal number of bins: equal-frequency and equal-width. In equal-frequency binning, the range of possible values are divided into k bins, each of which holds the same number of instances. In equal-width binning, which is the method we have used, the range are divided into k subranges of the same size. The optimal number of bins is the one that minimizes the entropy of the data. Weka computes different number of bins for different configurations, ranging from 4 to 10. The number of new features added to the parser is equal to $\sum_C B(C)$ where C is a configuration and $B(C)$ is the number of bins for configuration C.

5.4 Evaluation

The three methods described above have been evaluated on FTB_TEST. Results are reported in table 11. The three methods outperformed the baseline (the state of the art parser for French which is a second order graph based method) (Bohnet, 2010). The best performances were obtained by the Double Parsing method that achieved a labeled relative error reduction of $7,1\%$ on predicted POS tags, yielding the best parsing results on the French Treebank. It performs better than the Post Processing method, which means that the second parsing stage corrects some inconsistencies introduced in the Post Processing method. The performances of the Feature Based method are disappointing, it achieves an error reduction of 1.4%. This result is not easy to interpret. It is probably due to the limited number of new features introduced in the parser. These new features probably have a hard time competing with the large number of other features in the training process.

		pred. POS tags		gold POS tags	
		punct	no punct	punct	no punct
BL	LAS	88.02	90.24	88.88	91.12
	UAS	90.02	92.50	90.71	93.20
PP	LAS	88.45	90.73	89.46	91.78
	UAS	90.61	93.20	91.44	93.86
DP	LAS	**88.87**	**91.10**	**89.72**	**91.90**
	UAS	**90.84**	**93.30**	**91.58**	**93.99**
FB	LAS	88.19	90.33	89.29	91.43
	UAS	90.22	92.62	91.09	93.46

Table 11: Parser accuracy on FTB_TEST using the standard parser (BL) the post processing method (PP), the double parsing method (DP) and the feature based method.

6 Conclusion

Computing lexical affinities, on large corpora, for specific lexico-syntactic configurations that are hard to disambiguate has shown to be an effective way to increase the performances of a parser. We have proposed in this paper one method to compute lexical affinity scores as well as three ways to introduce this new information in a parser. Experiments on a French corpus showed a relative decrease of the error rate of 7.1% Labeled Accuracy Score.

Acknowledgments

This work has been funded by the French Agence Nationale pour la Recherche, through the projects SEQUOIA (ANR-08-EMER-013) and EDYLEX (ANR-08-CORD-009).

References

A. Abeillé, L. Clément, and F. Toussenel. 2003. Building a treebank for french. In Anne Abeillé, editor, *Treebanks*. Kluwer, Dordrecht.

E.H. Anguiano and M. Candito. 2011. Parse correction with specialized models for difficult attachment types. In *Proceedings of EMNLP*.

M. Bansal and D. Klein. 2011. Web-scale features for full-scale parsing. In *Proceedings of ACL*, pages 693–702.

D. Bikel. 2004. Intricacies of Collins' parsing model. *Computational Linguistics*, 30(4):479–511.

B. Bohnet. 2010. Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of ACL*, pages 89–97.

M. Candito and B. Crabbé. 2009. Improving generative statistical parsing with semi-supervised word clustering. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 138–141.

M. Candito and D. Seddah. 2010. Parsing word clusters. In *Proceedings of the NAACL HLT Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 76–84.

M. Candito, B. Crabbé, P. Denis, and F. Guérin. 2009. Analyse syntaxique du français : des constituants aux dépendances. In *Proceedings of Traitement Automatique des Langues Naturelles*.

W. Chen, J. Kazama, K. Uchimoto, and K. Torisawa. 2009. Improving dependency parsing with subtrees from auto-parsed data. In *Proceedings of EMNLP*, pages 570–579.

K.W. Church and P. Hanks. 1990. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29.

P. Denis and B. Sagot. 2010. Exploitation d'une ressource lexicale pour la construction d'un étiqueteur morphosyntaxique état-de-l'art du français. In *Proceedings of Traitement Automatique des Langues Naturelles*.

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten. 2009. The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18.

R. Hwa. 2004. Sample selection for statistical parsing. *Computational Linguistics*, 30(3):253–276.

T. Koo, X. Carreras, and M. Collins. 2008. Simple semi-supervised dependency parsing. In *Proceedings of the ACL HLT*, pages 595–603.

S. Kübler, R. McDonald, and J. Nivre. 2009. Dependency parsing. *Synthesis Lectures on Human Language Technologies*, 1(1):1–127.

M.P. Marcus, M.A. Marcinkiewicz, and B. Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330.

D. McClosky, E. Charniak, and M. Johnson. 2006. Effective self-training for parsing. In *Proceedings of HLT NAACL*, pages 152–159.

R. McDonald, F. Pereira, K. Ribarov, and J. Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of HLT-EMNLP*, pages 523–530.

S.A. Mirroshandel and A. Nasr. 2011. Active learning for dependency parsing using partially annotated sentences. In *Proceedings of International Conference on Parsing Technologies*.

P. Nakov and M. Hearst. 2005. Using the web as an implicit training set: application to structural ambiguity resolution. In *Proceedings of HLT-EMNLP*, pages 835–842.

A. Nasr, F. Béchet, J-F. Rey, B. Favre, and Le Roux J. 2011. MACAON: An NLP tool suite for processing word lattices. In *Proceedings of ACL*.

E. Pitler, S. Bergsma, D. Lin, and K. Church. 2010. Using web-scale N-grams to improve base NP parsing performance. In *Proceedings of COLING*, pages 886–894.

K. Sagae and J. Tsujii. 2007. Dependency parsing and domain adaptation with lr models and parser ensembles. In *Proceedings of the CoNLL shared task session of EMNLP-CoNLL*, volume 7, pages 1044–1050.

R. Sánchez-Sáez, J.A. Sánchez, and J.M. Benedí. 2009. Statistical confidence measures for probabilistic parsing. In *Proceedings of RANLP*, pages 388–392.

M. Steedman, M. Osborne, A. Sarkar, S. Clark, R. Hwa, J. Hockenmaier, P. Ruhlen, S. Baker, and J. Crim. 2003. Bootstrapping statistical parsers from small datasets. In *Proceedings of EACL*, pages 331–338.

J. Suzuki, H. Isozaki, X. Carreras, and M. Collins. 2009. An empirical study of semi-supervised structured conditional models for dependency parsing. In *Proceedings of EMNLP*, pages 551–560.

M. Volk. 2001. Exploiting the WWW as a corpus to resolve PP attachment ambiguities. In *Proceedings of Corpus Linguistics*.

G. Zhou, J. Zhao, K. Liu, and L. Cai. 2011. Exploiting web-derived selectional preference to improve statistical dependency parsing. In *Proceedings of HLT-ACL*, pages 1556–1565.