A Framework for Unsupervised Dependency Parsing using a Soft-EM Algorithm and Bilexical Grammars

Martín Ariel Domínguez[1], Gabriel Infante-Lopez[1,2]

[1]Grupo de Procesamiento de Lenguaje Natural,
Universidad Nacional de Córdoba, Argentina
[2]Consejo Nacional de Investigaciones Científicas y Técnicas, Argentina
{mdoming, gabriel}@famaf.unc.edu.ar

Abstract. Unsupervised dependency parsing is acquiring great relevance in the area of Natural Language Processing due to the increasing number of utterances that become available on the Internet. Most current works are based on Dependency Model with Valence (DMV) [12] or Extended Valence Grammars (EVGs) [11], in both cases the dependencies between words are modeled by using a fixed structure of automata. We present a framework for unsupervised induction of dependency structures based on CYK parsing that uses a simple rewriting techniques of the training material. Our model is implemented by means of a k-best CYK parser, an inductor for Probabilistic Bilexical Grammars (PBGs) [8] and a simple technique that rewrites the treebank from k trees with their probabilities. An important contribution of our work is that the framework accepts any existing algorithm for automata induction making the automata structure fully modifiable. Our experiments showed that, it is the training size that influences parameterization in a predictable manner. Such flexibility produced good performance results in 8 different languages, in some cases comparable to the state-of-the-art ones.

Keywords: Unsupervised dependency parsing, bilexical grammars, soft-EM algorithm.

1 Introduction

In the last decade, unsupervised dependency parsing has acquired increasing relevance [3,4,9,11,13,17,18]. The special interest in unsupervised methods comes hand in hand with the growing number of natural languages available in different applications on the Internet. Unlike supervised and semi-supervised methods, unsupervised dependency parsing does not require training from hand-annotated corpora which is usually an expensive process. Therefore, unsupervised parsing becomes a solution for languages and domains with minimal hand-annotated resources, making it a low cost and high performance method of approaching the new challenges of natural language parsing.

Unsupervised dependency parsing looks for regularities in the languages by applying statistical methods to large quantities of data. The resulting linguistic patterns serve

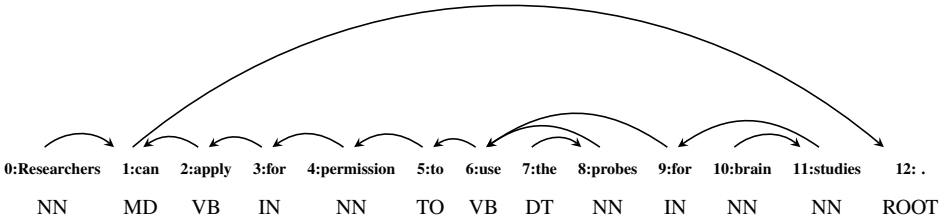

Fig. 1. An example of a dependency tree

as rules for inferring an underlying grammar. For example, the "dependency" pattern considers that the "dependent" of a preposition is usually a noun occurring to its right. This rule can explain -that is, parse- a number of different sentences that other candidate rules cannot. Then, a better method should prefer such a rule over competing alternatives and "discover" that grammatical rule. Usually, dependency relations are modeled as shown in Figure 1, where each arc is a relation between a head and its argument.

Currently, unsupervised dependency parsers exhibit a degree of complexity that can shy away newcomers to the field. We challenge such complexity and present a straightforward soft-EM based framework. We achieve results close to state-of-the-art ones, while making it simple to experiment with sub-components (see below).

Since the task is unsupervised, correct dependency structures are not available and our input consists only of sequences of parts of speech (POS) tags. Our dependency relations are modeled with Probabilistic Bilexical Grammars (PBGs) [8] for which we have implemented a novel learning/training algorithm. Our algorithm is a soft version of the EM-algorithm [5]. As shown in [16] an EM algorithm for inducing grammars can be described as an iteration between an E-step and an M-step. During the E-step a new treebank is computed, while during M-step a grammar together with its probabilities is read out from the treebank. Usual implementations of the EM do not actually compute the treebank; they compute the new grammar using inside-outside probabilities from the previous step.

We take a different approach. We present an algorithm based on the 4 different modules showed in Figure 2, that mainly computes new versions of a treebank. These 4 components are: a supervised PBGs INDUCTOR, (simulating the M-step), a k-BEST PBG PARSER, plus a TREEBANK REWRITER (together simulating the E-step), and an initial TREEBANK GENERATOR (in charge of building the initial seed). In the first step of the algorithm, the grammar is learned from an initial set of trees. Those trees are built based on constraints aimed to start the learning process from simple models. In each iterative step of the algorithm, it parses the set of sentences with the PBG and refines the grammar by contrasting the parsed trees of the input sentences. The quality of the grammar of each step is calculated by the logarithmic likelihood of the treebank obtained using that grammar to parse the set of input sentences.

The resulting soft-EM[1] algorithm is well defined for different PBG learning algorithms and for different initial treebanks. Consequently, these two components can be instantiated differently at almost no effort.

Thanks to the versatility offered by our framework, we are able to test three different ways to generate initial treebanks, and two different schemas for learning automata. Most of the recent work in this area, e.g., [4, 11, 18], has focused on variants of the Dependency Model with Valence (DMV) [12]. DMV was the first unsupervised dependency grammar induction algorithm to achieve accuracy above a right-branching baseline. With all its strengths, DMV is still limited in the type of dependencies it can model. The DMV model can be seen as a sort of PBG with the particularity that all of its automata have similar structures and that they only differ in the probabilities of their arcs. In contrast with our model, DMV and others in the literature are still in need of a well understood learning mechanism. By using a generalization of EM we can tap into a large body of learning expertise.

Our results show a very good performance in 5 languages. Particularly, for English these are very close to the state-of-the-art performance for sentences with a restricted length of up to 10 POS. For languages with enough available training material (German, Portuguese and Danish), we have state-of-the-art results or close to them such as for Swedish. For the rest of languages Turkish, Spanish and Bulgarian, our performance is considerably higher than the standard DMV performance.

The paper is organized as follows: Sections 2 and 3 present our framework and the algorithms for learning automata. Section 4 shows experimental results, Section 5 discusses related work, Section 6 explains possible research lines to continue this work and, finally, Section 7 concludes the paper.

2 Training Architecture

The training or learning framework (Figure 2) consists of 4 modules: the TREEBANK GENERATOR, the PBGS INDUCTOR, a k-BEST PBG PARSER, and a TREEBANK REWRITER. The learning algorithm starts by creating a treebank over a given set of sentences. The resulting treebank is used by the PBGS INDUCTOR module to induce a PBG. Once a grammar has been induced, it is used by the k-BEST PBG PARSER to parse all original sentences. The k-BEST PBG PARSER returns the k-best trees for each sentence with their corresponding probabilities. All these trees are used by the TREEBANK GENERATOR to create a new treebank that reflects the probabilities of all trees. Once the new treebank has been created, the algorithm cycles between the PBGS INDUCTOR, k-BEST PBG PARSER and TREEBANK REWRITER until the likelihood of the k-best trees hopefully converges. We will now describe each component.

[1] It is soft-EM because of the parameter k in the parser. If $k = \infty$ it will be a hard-EM.

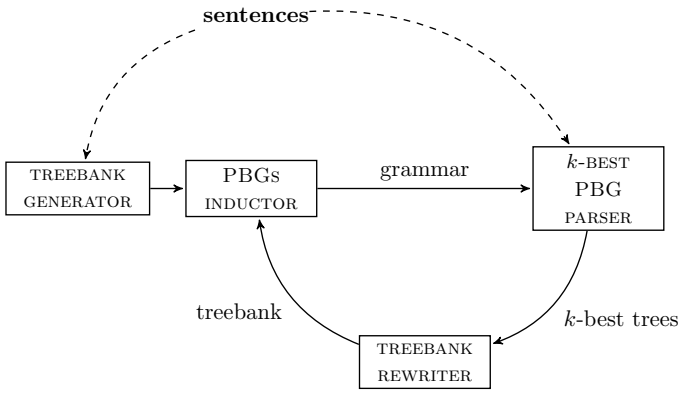

Fig. 2. Framework Schema

PBGs INDUCTOR. This module is one of the key components of our algorithm. Its task is to find a probabilistic bilexical grammar from a collection of dependency trees. From [8], recall that a *Bilexical Grammar* B is a 3-tuple $(W, \{r_w\}_{w \in W}, \{l_w\}_{w \in W},)$ where, W is a set of terminals, plus a distinguished symbol ROOT, and l_w, r_w with $w \in W$ are probabilistic automata with initial symbols S_l^w and S_r^w respectively. For this paper, it is enough to grasp the intuition behind them: the two automata for a word w accept a sub-language of W^* which models the arguments of w to its right and to its left. A Bilexical Grammar makes the strong assumption that the languages defined by the arguments of a word can be modeled with regular languages.

Learning a dependency grammar from a dependency treebank is simple if a learning algorithm for the induction of its automata is given. To induce a PBG from a dependency corpus, first we need to build the bags of strings that are to be used to learn the automata. In this sense, two bags of dependencies are built for each terminal in the set of terminals. These bags are given to the automata learning algorithm and it produces the two automata for that particular terminal. The bags of words are extracted from all trees in the dependency treebank. For example, using the tree in Figure 1 the corresponding left and right bags for POS `VB` are $M_{left}^{VB} = \{$`"VB #","VB #"`$\}$ and $M_{right}^{VB} = \{$`"VB IN #","VB NN IN #"`$\}$ respectively[2]. The symbol `#` marks the end of a string.

Once the process of collecting dependents has finished, there are two bags M_{left}^{w} and M_{right}^{w} for each POS w. These bags are used as training material to inducing automata l_w and r_w. We can now define the PBG $B = (\text{POS}, \{r_w\}_{w \in POS}, \{l_w\}_{w \in POS})$. In Section 3, we describe some algorithms capable of induce the automata l_w and r_w.

[2] Note that, for each POS, only the incoming arrows are considered as left or right dependents.

k-BEST PBG PARSER. Since PBGs are a particular case of probabilistic context free grammars, our parser for PBG is based on an implementation of a k-best CYK parser for Chomsky Normal Form PCFGs. The k-BEST PBG PARSER returns the k-best trees together with their probabilities.

TREEBANK REWRITER. Intuitively, this module uses the k-best trees for creating a new treebank that resembles the known probabilities of individual trees. Although we know the probabilities of the sentences, we need to replicate it because it allows us to use any automata inductor, for example MDI, which accepts only a set of sentences as a training material. Since the grammar inductor only takes a treebank as input, it is not aware of their probabilities. The TREEBANK REWRITER module replicates the k-best trees in such a way that the probability mass associated to each tree is proportional to the probability assigned by the parser. The TREEBANK REWRITER produces a new treebank that contains as many trees for each sentence as are required to reproduce the sentence probability. In order to mark the boundaries of the number of possible replicas, we introduce a constant M that states the maximum number of trees a sentence will have in the resulting treebank. Suppose that $C = \{c^1 \dots c^N\}$ of N sentences are the input senteces. Suppose also that $t_1^j \dots t_k^j$ are the k trees returned by the parser for the sentence c^j and let $p_1^j \dots p_k^j$ be their probabilities. t_i^j is replicated R_i^j times, where:$R_i^j = \text{round}\left(M * \frac{p_i^j}{\sum_{l=1}^{k} p_l^j}\right)$. Finally, the size of the resulting treebank is $\sum_{j=1}^{N}\sum_{i=1}^{k} R_i^j$. Note that, under this definition, if the probability of a tree is too small, it will not be a part of the new treebank at all. For computational reasons both k and M cannot be too large. In all our experiments, k and M are set to 130 and 150 respectively.

TREEBANK GENERATOR. The aim of the TREEBANK GENERATOR is to build the first treebank that is given to the grammar inductor. This module uses *meta-trees*. A *meta-tree* of size n is a tree with a particular fixed structure of arcs, similar to a dependency tree, but with n variable nodes, where any sentence of length n can be fitted. These meta-trees have the particularity that their leaves are all different and that they are grouped by sentence length. Given an initial sentence of length n, a small treebank for this sentence is built via a two step procedure. First, all meta-trees whose yield has length n are selected and, second, all terminal symbols in the meta-trees are replaced by those in the sentence. The TREEBANK GENERATOR produces a new treebank by joining individual treebanks for all sentences. As a consequence, the resulting treebank contains the same trees for all sentences with the same length independently of the sentence words.

To generate the meta-trees that correspond to all n-long sentences, a special sentence $w_0, \dots, w_n$, with all w_i different symbols, is processed by the parser and the tree rewriter modules. The sentence is parsed with an *ad-hoc* PBG that we built specifically for each possible length. The *ad-hoc* PBG is defined by describing the automata for each word in the sentence. If w_i is the i-th terminal then, its right automaton is like the one shown in Figure 3. That is, the automaton has three states, one is final and absorbing, one is initial and has only one outgoing transition that is labeled with label w_i and

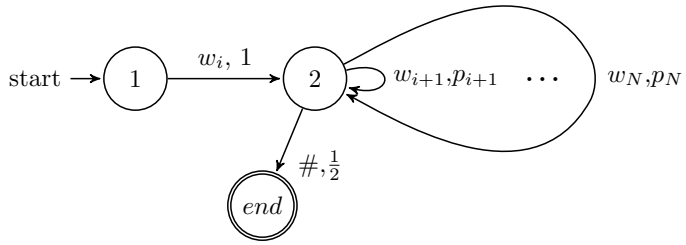

Fig. 3. Typical meta-tree building automaton

probability 1. The third state is in between the two previous ones. It is connected to the final state by means of an arc labeled with probability 0.5 and label $\#$. Moreover, it has $n - i$ loops with labels $w_{i+1} \dots w_n$ and probabilities p_j defined as: $p_j = \frac{\frac{1}{d_{ij}^s}}{2 * \sum_{d=1}^{n-1} \frac{1}{d^s}}$, where d_{ij} is the distance between w_i and w_j, and the exponent s is a parameter in our model that modifies the mass of probability that are assigned to long dependencies. The three initialization, namely init-1, init-2 and init-3, we report in Section 4 are obtained by setting s to 1, 2 and 3 respectively. All p_i are such that their sum is not equal to 1, and in order to correctly define a probabilistic automaton, a forth non-final and absorbing state is required to normalize the probability. The probability of going to this state is $1 - (0.5 + 0.5 * \sum_{l=i+1}^{m} p_l)$. This state is not shown in the picture. Intuitively, the probability of having many dependents and of having long distance dependents diminishes with an exponential factor s. The bigger the s the less likely are these two situations. The $2 * m$ automata, 2 per terminal symbol in the sentence, plus one automaton for the *root* symbol, are used to define a PBG G. Following the general schema, G is used to produce the k-best parsers of sentence $w_0, \dots, w_m$. These k trees are fed into the TREEBANK GENERATOR, and it produces the treebank of meta-trees.

3 Automata Learning Algorithms

We use two different algorithms for learning probabilistic automata. First, we propose the Minimum Discrimination Information (MDI) algorithm [24], which infers automata in a fully unsupervised fashion. It takes a bag of strings and returns a probabilistic deterministic automaton. Briefly, the MDI algorithm produces an automaton by first building a prefix tree that accepts only the training material. Moreover, the prefix tree contains one path for each string and it contains as many final states as there are different strings. All arcs in the prefix tree are marked with the number of times each arc is traversed while mapping strings into paths. These numbers are then transformed into probabilities which results in a probabilistic automaton that recognize exactly the training material. The algorithm proceeds to look for pairs of states that can be merged into one single state. Two states can be merged if the probability distribution defined

by the automata that results from the merge is not too far away[3] from the distribution defined by the prefix tree. The algorithm proceeds greedily until no further merges can be done. MDI has only one parameter, α, that can be used to control the maximum allowed value of distance between two distributions before a merge is performed. α is a real number between 0 and 1; when it is equal to 0, no merges are allowed while when equal to 1 all merges are allowed. The MDI algorithm receives a bag of strings together with a value for the parameter α, and it outputs a probabilistic automaton.

Second, we contribute an *ad hoc* algorithm that only learns the transitions probabilities of a given automaton backbone structure. A backbone structure is a deterministic finite automaton without probabilities. It receives a bag of strings and a automaton backbone structure and returns the automaton backbone structure plus their probabilities. The backbones we use are general enough to warranty that they accept all strings in the training material. In contrast, our second algorithm is not fully unsupervised because it receives along with a bag of strings, the backbone of the automaton it should produce. The backbone consists of the states and the arcs of the automaton, but it is missing the transition probabilities. It is the task of our Given Structure (GS) algorithm to find them.

As we see in Section 5, DMV and EVG define a particular skeleton to their automata and as is the GS, the skeleton is information that is given to the algorithm as prior knowledge. In this sense, MDI is fairer learner than GS given that it works with less information. In our experiments we show that even with less information, MDI works better than GS. We currently experiment with different skeletons, but all of them have similar structure: They have a unique absorbing final state, and N intermediate states $S_1 \dots S_N$. The skeleton has one arc between states S_i and S_{i+1} for each possible label, and one arc between states S_i and the final state, labeled with the end of production symbol $\#$. The GS algorithm uses the training material to estimate all arcs probabilities. Since the skeleton is both deterministic and expressive enough, there is a path in the automaton for each sequence in the training material. The GS algorithm maps each sentence to a path in the automaton, it records the number of times each arc has been used, and, finally, it transforms those counters into probabilities. GS-N refers to the GS algorithm when it uses a backbone with N states as describe above. We use these skeletons because they are the easiest structure that can be manually described without making strong assumptions about the underlying language. We experiment with two different skeletons both having 3 and 4 states respectively. The skeleton with 3 states pays special attention to the first dependent while the one with 4 to the first two dependents. Moreover, GS-3 automata generate dependents independently of their siblings. GS-4 automata can recall if a dependent is the first one or not. In general the GS-i can recall if there has been less that $i - 1$ dependents. Our GS algorithm is also a generalization over n-grams which can be seen as instances of GS-n where the skeleton has a particular shape. Moreover, Section 5 shows that they can be seen as instances of a

[3] The difference between the two probability distributions is computed by means of the Kullback-Leibler divergence

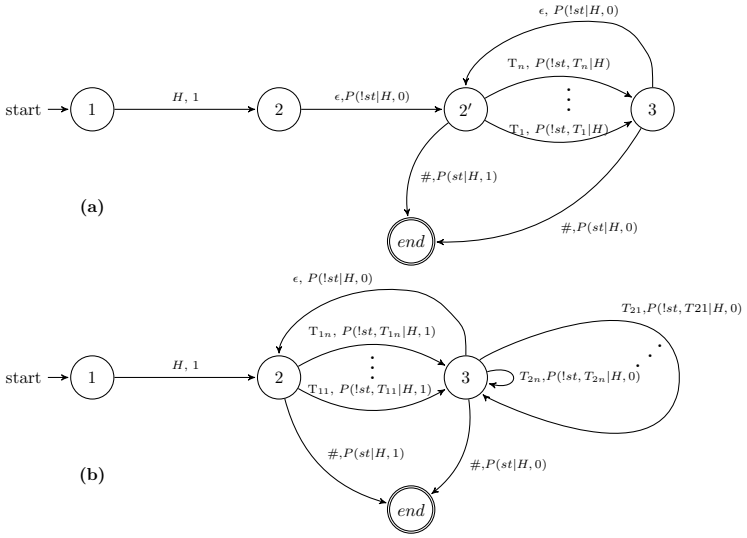

Fig. 4. DMV (a) and EVG (b) automata

Table 1. Size of the training corpus for each language and the number of differents POS tags

	English	German	Turkish	Swedish	Bulgarian	Spanish	Portuguese	Danish
#sen.	6007	12089	3203	3255	5713	595	2409	1757
#POS	36	51	28	40	40	23	19	24

version of GS that allows non-deterministic backbones.

4 Experimental Results

Our model was tested on sentences with a restricted length up to 10 POS. Due to computational costs, the current version of our parser cannot deal with sentences of any length. However, we have some experiments which use sentences with up to 20 POS with promising results.

We report results for English, German, Swedish, Turkish, Bulgarian, Spanish, Portuguese and Danish. We tested 3 different initializations in the TREEBANK GENERATOR module, and two different algorithms for learning automata in the PBGs INDUCTOR module. This showcases the flexibility of our framework. All of our experiments used syntactic categories —POS tags— instead of words. The English model was induced using sentences in the Penn treebank (PTB) [14] with at most ten words (usually called WSJ10 [11, 12, 17]). Sections 2 to 21, that is, 6,007 sentences in total, were used for training. Testing was done using the 398 sentences of Section 23. The metrics used for

Table 2. Directed accuracies on Section 23 of WSJ10 for several baselines and recent systems

model	accuracy
Attach-Right [13]	33.4
DMV-standard [13]	45.8
DMV-babysteps (@15) [18]	55.5
DMV-babysteps (@45) [18]	55.1
DMV-diriclet [3]	45.9
Best of [15]	53.5
Log-Normal Families [3]	59.4
Shared Log-Normals (tie-verb-noun) [4]	61.3
Bilingual Log-Normals (tie-verb-noun) [4]	62.0
EVG-Smoothed (skip-head) [11]	65.0
EVG-Smoothed (skip-val) [11]	62.1
Viterbi EM [23]	**65.3**
EVG-Smoothed (skip-head), Lexicalized [11]	68.8
Hypertext Markup [20]	**69.3**
LexTSG-DMV ($P_{lcfg}, P_{cfg}, P_{sh}$) [1]	67.7

model	accuracy
MDI, $\alpha = 0$, init-1	67.5
MDI, $\alpha = 0$, init-2	**69.1**
MDI, $\alpha = 0$, init-3	67.2
GS-3, init-1	50.9
GS-3, init-2	66.6
GS-3, init-3	67.1
GS-4, init-1	55.7
GS-4, init-2	66.7
GS-4, init-3	**67.6**

evaluation are the usual ones, directed and undirected accuracy[4]. Table 2 compares our English results with others in the literature. In this case we report only directed accuracy. From the table, it can be seen that our results are comparable with the state-of-the-art ones, even for lexicalized instances. Our best result is what we call MDI-0: MDI with $\alpha = 0$ using init-2.

Our best performing models are those that can model the language of dependents as finite languages. Moreover, an inspection on the resulting automata shows that all models tend to create very short sequence of dependents, mostly of length up to one. To better understand our results, it is important to think our learning algorithm as a two-step process. First, the parser and the rewriter define the training material that is going to be given to the automata learning algorithms. In a second phase, all the automata are learned. It is interesting to note that both the MDI and the GS-n algorithms generalize less over the training material as their respective parameters α and n go to zero and ∞, respectively. The MDI algorithm ends up building a tree like automata that recall all and only those strings in the training material. The probability assigned to each string is proportional to the number of times it occurs in the training material. In contrast, a GS-n automaton recalls the number of times each tag occurs as the i-th dependent. In this sense, MDI-0 generalizes less than any GS-n. In both cases, the resulting automaton accepts only finite languages.

From the experiments, it is clear that GS-4 improves over GS-3, and both EVG and DMV. It is surprising that our models obtain good results without resorting to smoothing which is usually applied in other models. As the experiments show, good results are obtained with initializations that penalize long distance dependencies and a high number of dependents. In other words, the models that work better are those that use initialization where words have fewer dependents and where dependents are close to

[4] Directed accuracy is the ratio of correctly predicted dependencies (including direction) over total amount of predicted dependencies. Undirected accuracy is much the same, but also considers a predicted dependency correct if the direction of the dependency is reversed [12].

their heads. If we compare the automata that results at the end of our algorithm, when init-2 and init-3 is used, the most noticeable feature is that, even for GS-3 and GS-4, the probabilities associated to cycling arcs are zero or very close to zero. When init-1 is used, cycles occur in the final automata of GS-3 but only a few in GS-4. Note that MDI-0 is stable across different initializations. If we look at the resulting automata, they all accept finite languages and moreover, all elements in the languages contain only a few symbols per string.

Since there are many parameters in our setup, it might be the case that our models are optimized for English. To validate our model, and unsupervised models in general, it is important to test their performance also in languages other than English. Table 3 compares our results for other languages. We compare them against standard baselines like right and left attachment, DMV model results, and the best results reported in [9]. According to [9], German and Turkish best results are obtained by one model while the score for English and Swedish by two other different models. The fourth row displays the highest score independently of the model used to obtain it. All corpora were part of the ConNLL-X special task on parsing [2]. We show results using treebanks for Swedish, German, Turkish, Bulgarian, Spanish, Portuguese and Danish. Trees that were non-projective or that had more than one root were discarded as well as all trees whose sentences were longer that 10 words. Except Turkish, where the best performing model is the right attach baseline, all instances of our algorithm improve over DMV and the baselines.

Figure 5 shows the evolution of the directed accuracy and logarithmic likelihood. Figure 5 (left) shows, for each language, the directed accuracy in the 30 first iterations measured against the gold trees from the training material. More specifically, while the X axe varies in the number of iteration, the Y axe plots, the directed accuracy of the trees that consists only of the most probably trees returned by the k-BEST PBG PARSER for each sentence in the training material [5]. For iteration number 0 we use the treebank returned by TREEBANK GENERATOR instead of k-BEST PBG PARSER.

Similarly, in Figure 5 (right) we plot the logarithmic likelihood for each treebank in the first 30 iterations. It is important to remark that the gold trees are used only for analysis purposes, and they are not used inside the algorithm.

Two variables must be taken into account to decide which parameterization of our system should be used for a given language: the number of sentences available for training and the number of POS tags.[6] Table 1 shows the variables for the languages used in this work. Table 3 shows that MDI-0 is very robust for those languages that have available a corpus with a significant number of sentences. This is the case of languages such as English, German and Bulgarian. For languages with a reduced number of sentences or a big number of POS, we should use GS-3 or GS-4. Intuitively, if we have

[5] This treebank is the same that produce as a result the 1-BEST PBG PARSER

[6] More tags means a larger number of automata to be induced and thus less training material for each automaton.

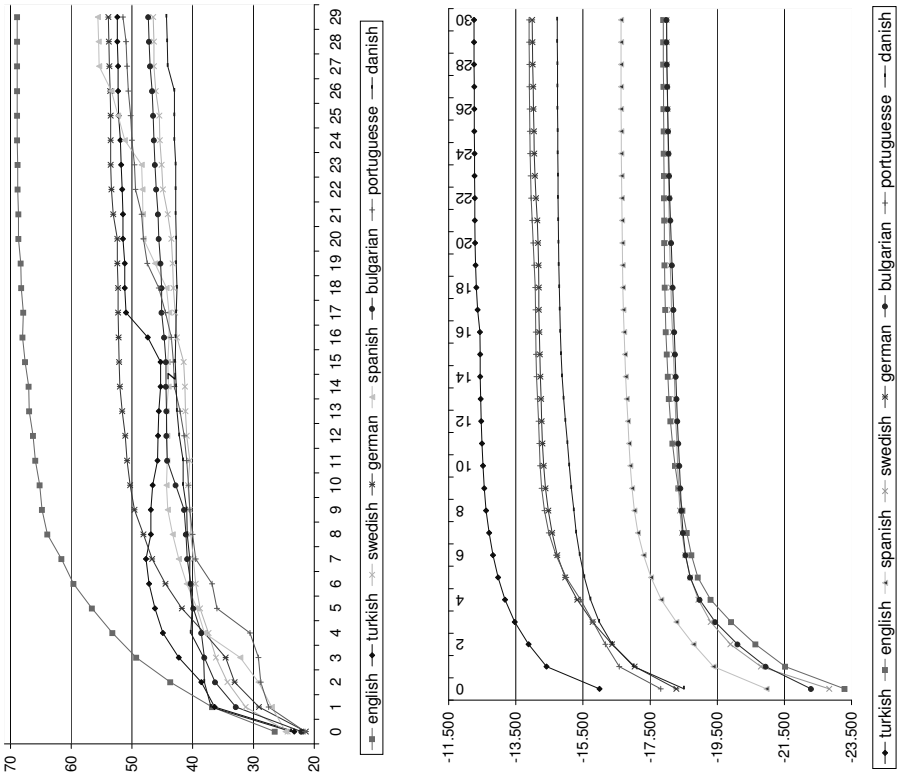

Fig. 5. (left):Directed Accuracy evaluated for each language over the first 30 iterations of the training phase of our framework (right):Evolution of logarithmic likelihood for each language

less training material, models like GS-3 or GS-4 perform better because they generalize over the training material better than MDI-0.

Table 3. Our results for a variety of languages compared with the baselines: right attach, left attach and standard DMV results. We also report the state-of-the-art results for these languages.

model	English	German	Turkish	Swedish	Bulgarian	Spanish	Portuguese	danish
Gillenwater et al. [9] DMV	45.8 / -	35.7 / -	46.8 / -	39.4 / -	37.8 / -	40.3 / -	35.7 / -	47.2 / -
left attach	24.1 / 54.6	25.9 / 53.2	5.1 / 54.8	28.5 / 56.3	40.5 / 59.9	29.0 / 55.2	34.1 / 61.7	43.7 / 60.1
right attach	33.4 / 56.3	29.0 / 52.1	**63.8 / 68.5**	28.5 / 55.5	20.2 / 56.3	29.4 / 55.2	27.9 / 55.5	17.2 / 57.5
Gillenwater et al. [9] best result	64.4 / -	47.4 / -	56.9 / -	**48.6 / -**	**59.8 / -**	**62.4 / -**	54.3 / -	46.6 / -
GS-3 Init-1	50.9 / 65.3	48.6 / 60.7	52.8 / 65.3	46.0 / 60.2	48.7 / 64.1	57.4 / 68.6	55.6 / 66.4	36.8 / 59.7
GS-3 Init-2	66.6 / 72.4	49.1 / 60.7	20.4 / 54.4	47.5 / 61.5	48.6 / 63.9	45.6 / 63.7	39.7 / 63.3	**47.9 / 66.4**
GS-3 Init-3	67.1 / 71.5	46.7 / 59.6	20.4 / 53.6	41.6 / 58.6	34.1 / 55.0	38.3 / 59.0	38.0 / 62.1	44.3 / 62.9
GS-4 Init-1	55.7 / 66.9	48.5 / 60.8	53.5 / 65.2	46.7 / 60.2	34.6 / 55.8	55.3 / 66.9	**55.7 / 66.8**	38.9 / 59.9
GS-4 Init-2	66.7 / 72.4	48.7 / 60.4	43.3 / 60.2	47.6 / 61.5	47.7 / 63.0	45.1 / 63.5	39.5 / 63.2	41.6 / 60.6
GS-4 Init-3	67.6 / 71.9	47.9 / 60.1	25.6 / 53.9	42.3 / 59.2	48.6 / 64.1	38.2 / 58.9	38.1 / 61.9	43.1 / 63.9
MDI-0 Init-1	67.5 / 72.5	47.7 / 60.1	52.5 / 64.9	45.4 / 59.5	35.9 / 55.6	51.1 / 62.6	49.5 / 63.6	35.5 / 58.1
MDI-0 Init-2	**69.1 / 73.3**	**54.1 / 63.4**	38.5 / 58.2	**48.1 / 61.4**	55.1 / 68.9	48.8 / 64.7	30.6 / 55.5	44.1 / 64.6
MDI-0 Init-3	67.2 / 72.6	53.9 / 63.5	24.6 / 53.0	46.2 / 60.7	38.1 / 56.5	46.0 / 64.0	30.8 / 55.8	44.7 / 65.0

5 Related Work

Most unsupervised approaches to unsupervised parsing are based on Dependency Model with Valence (DMV). DMV implements an EM algorithm that maximizes the likelihood of a particular grammar. This grammar can be seen as PBG where all its automata are like the one in Figure 4 (a). The probability between states 1 and 2 is the probability of generating a particular head. The one between states 2 and $2'$ is the probability of generating any dependent using a ϵ movement; the one between states 2 and end is the probability of not generating any dependent; the one between $2'$ and 3 is the probability to generate a particular dependent with its corresponding probability, the one between 3 and 2 is the probability of generating a new dependent, modeled again with an ϵ move, and finally, the probability between 3 and end is the probability of stop generating.

In general, is not possible to transform a non-deterministic automaton to a deterministic one [7]. But for this particular case, the automata can be transformed to one

without ϵ moves, but having in mind that some of the its arc probabilities are correlated and consequently can not be learned independently. Cohen et al. [3] derive a Variational Bayes EM for the DMV model. results were 59.3 of directed accuracy.

Spitkovsky et. al. [18] use the DMV model, but they introduce two interesting techniques. First, they use an incremental initialization that starts with sentences of length 1, and only later uses longer sentences. Second, they analyze the trade-off between complexity and quality in the training phase. They found that training with sentences up to length 15 performs better than training with longer sentences when testing in section 23 of WSJ10, WSJ20, WSJ30, WSJ100 and WSJ∞, among others.

Headden et al. [11] extend DMV by adding a parameter that distinguishes the probabilities of the first dependent from the probabilities of the subsequent ones. As in the DMV, even with the automata not being explicitly defined, they can be rebuilt from the definition of the model. Figure 4 (b) shows such an automaton. The probability between states 1 and 2 is the probability of generating a particular head, between 2 and 3, are the probabilities of generating a particular dependent as the first one, between 3 and 3 are the probabilities of generating a dependent that is not the first one anymore, the probability between 2 and end is the probability of not having any dependents, and finally the probability between 3 and end is the probability of stopping generating dependents. To maximize the likelihood of their model they use a Variational Bayes EM algorithm with a Dirichlet prior (similar to [3]) and they used a linearly smoothed model to deal with the data sparseness. As their initialization, they randomly sample some sets of trees and choose the best ones using some iterations of a Variational Bayes EM algorithm. They show that, by including smoothing, they improve over DMV obtaining the best result that is known for unsupervised parsing. The unsmoothed version of the EVG model corresponds exactly to our GS-3 model. The DMV model without the one-side-first parameter, is in between GS-2 and GS-3. It does not distinguish the probabilities for the dependent generation, as in GS-2, but the probability of stopping is distinguished like in GS-3. Gillenwater et al. [9] used a posterior regularization (PR) framework [10] instead of the traditional EM algorithm. They model their dependencies as in DMV, and as variants of the EVG model. They argue that the main problem with unsupervised parsing is data sparseness and their model deals with such problem adding constraint that control for long dependencies. They report substantial gains over the standard EM algorithm, but they are not stable across languages. Finally, our soft-EM corresponds to a hard EM algorithm [16] when a k-BEST PBG PARSER with $k = \infty$ is used. Hard EM is not feasible in our setup because a ∞-best parsing requires an exponential amount of space and time.

Spitkovsky et. al. [23] is in one sense, the work most similar to ours, as we are also estimating the probabilities of a model given the previous model, albeit using k-best parse trees. They obtain good scores 44.8% for English, in long sentences (all sentences in section 23 of PTB) .

Blunsom and Cohn [1] replace the simple underlying grammar commonly used by a

probabilistic tree substitution grammar. This formalism is capable of better representing complex linguistic structures because they can learn long dependencies. To limit the model's complexity they used a Bayesian non-parametric prior. They obtained state-of-the-art results for English in long sentences 55.7%.

Using standard DMV, Spitkovsky et al. [20] use Web mark-up for improving parsing up to 50.4% of directed accuracy.

6 Future Work

One of the most important aspects to continue our work is to extend our experiments to longer sentences. To do so, we should optimize our implementation of the parser to make it parallel. We performed some experiments with wsj15 and we obtained promising results, about 49% of directed accuracy, which is close to the state-of-the-art ones, about 53%.

Another experiment that we will perform is to use ideas explored in [6] to split the POS tags in a refined set. Fully lexicalized models may be costly and too sensitive to data sparseness, nevertheless we think unsupervised parsers can benefit of a more appropiate granularity of POS tag sets. This idea may be implemented by selecting a POS tag and splitting the words with this POS by using a chosen feature function. For example, by selecting the POS `VB`, and splitting it using the distance of the word to the root node. We think of applying the split of POS tags starting with the initial tree-bank, which is obtained as in section 2. This split should be recalculated in each step of our learning architecture, after the new set of dependency trees is calculated by using the k-best parser. We hope that this idea may help to obtain more accurate dependency trees, specially with longer sentences because it could distinguish more complex dependency relationships by using more automata specialized according to the dependency languages of each POS tag considered.

Finally, our model allows us to choose different kinds of automata structures. Another interesting idea to improve this parsing model is to benefit from this flexibility of our framework. An interesting experiment is to choose the automaton structure to be associated with a given POS tag according with the size of its training set. As the results obtained for different languages suggest, the automata structure[7] can be adapted to the size of the dependency tree-bank; our idea is to investigate potential relationships between the size of the training set associated with each POS tag.

7 Discussion and Conclusions

Over the last years NLP research has been focused in unsupervised dependency parsing, specially after the DMV parser [13]. Most of the recent parsers like [1, 3, 4, 9, 11, 15,

[7] Recall that we use the same automata structure for all POS tags.

21, 22] are essentially the DMV model which uses a different bias function in each step of its EM algorithm definition. This bias function penalizes long dependencies in utterances. This penalization is needed, as it is explained in [19], because DMV reserves too much probability mass for what might be unlikely productions at the beginning, and the classic EM is not good enough for redistributing such probability across all parse trees.

We chose to take a different approach. Our algorithm implements a soft-EM algorithm that instead of computing the probability distribution over the whole forest of trees, uses a tree-replicator module that builds tree-banks resembling the most likely part of the probability distribution. The resulting algorithm allows us to test different ways to model dependents and different ways to induce automata.

Instead of using a penalization in each iteration of the EM algorithm, in our model we use different biased tree-banks which perform the penalization of long dependencies. We show experimental results using three different initializations, two automata learning algorithms and eight different languages.

Our experiments showed that, for a given language, we have to choose a parameterization of our system that generalizes across different training sets depending on the size of training material available for this language. We show training size influences parameterization in a predictable manner.

References

1. Blunsom, P., Cohn, T.: Unsupervised induction of tree substitution grammars for dependency parsing. In: In Proceedings of EMNLP 2010 (2010)
2. Buchholz, S., Marsi, E.: Shared task on multilingual dependency parsing. In: CoNLL-X. SIGNLL (2006)
3. Cohen, S.B., Gimpel, K., Smith, N.A.: Logistic normal priors for unsupervised probabilistic grammar induction. In: Proceedings of Advances in Neural Information Processing Systems (NIPS) (2008)
4. Cohen, S.B., Smith., N.A.: Shared logistic normal distributions for soft parameter tying in unsupervised grammar induction. In: Proceedings of NAACL-HLT. (2009)
5. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the em algorithm. JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B 39(1), 1–38 (1977)
6. Domínguez, M.A., Infante-Lopez, G.: Searching for part of speech tags that improve parsing models. In: Proceedings of the 6th international conference on Advances in Natural Language Processing, GoTAL,Gotemburgo, Suecia. (2008)
7. Dupont, P., Denis, F., Esposito, Y.: Links between probabilistic automata and hidden Markov models: probability distributions, learning models and induction algorithms. Pattern Recognition 38(9), 1349–1371 (September 2005)
8. Eisner, J.: Three new probabilistic models for dependency parsing: An exploration. In: Proceedings of COLING-96. Copenhagen (”1996”)

9. Gillenwater, J., Ganchev, K., Graça, J.a., Pereira, F., Taskar, B.: Sparsity in dependency grammar induction. In: Proceedings of the ACL 2010 Conference Short Papers. pp. 194–199. Morristown, NJ, USA (2010)
10. Graça, K.G., Taskar, B.: Expectation maximization and posterior constraints. In: Proceedings of Advances in Neural Information Processing Systems (NIPS) (2007)
11. Headden, W.P., Johnson, M., McClosky, D.: Improving unsupervised dependency parsing with richer contexts and smoothing. In: Proceedings of NAACL-HLT. (2009)
12. Klein, D.: The Unsupervised Learning of Natural Language Structure. Ph.D. thesis, Stanford University (2005)
13. Klein, D., Manning, C.: Corpus-based induction of syntactic structure: Models of dependency and constituency. In: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics (2004)
14. Marcus, M., Santorini, B.: Building a large annotated corpus of English: The Penn treebank. Computational Linguistics 19, 313–330 (1993)
15. Pate, J.K., Goldwater, S.: Unsupervised syntactic chunking with acoustic cues: computational models for prosodic bootstrapping. In: Proceedings of CMCL '11. Association for Computational Linguistics, Stroudsburg, PA, USA (2011)
16. Prescher, D.: Inside-outside estimation meets dynamic EM. In: Proceedings of the 7th International Workshop on Parsing Technologies (IWPT) (2001)
17. Smith, N., Eisner, J.: Guiding unsupervised grammar induction using contrastive estimation. In: IJCAI, Workshop on Grammatical Inference Applications. pp. 73–82. Edinburgh (July 2005)
18. Spitkovsky, V., Alshawi, H., Jurafsky, D.: From baby steps to leapfrog: How less is more in unsupervised dependency parsing. In: Proceedings of NAACL-HLT. (2010)
19. Spitkovsky, V.I., Alshawi, H., Jurafsky, D.: Baby Steps: How "Less is More" in unsupervised dependency parsing. In: NIPS: Grammar Induction, Representation of Language and Language Learning (2009)
20. Spitkovsky, V.I., Alshawi, H., Jurafsky, D.: Profiting from mark-up: Hyper-text annotations for guided parsing. In: Proceedings of ACL-2010 (2010)
21. Spitkovsky, V.I., Alshawi, H., Jurafsky, D.: Punctuation: making a point in unsupervised dependency parsing. In: Proceedings of CoNLL '11. pp. 19–28. ACL (2011)
22. Spitkovsky, V.I., Alshawi, H., Jurafsky, D.: Bootstrapping dependency grammar inducers from incomplete sentence fragments via austere models. Journal of Machine Learning Research - Proceedings Track 21 (2012)
23. Spitkovsky, V.I., Alshawi, H., Jurafsky, D., Manning, C.D.: Viterbi training improves unsupervised dependency parsing. In: Proceedings of CoNLL '10g (2010)
24. Thollard, F., Dupont, P., de la Higuera, C.: Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In: Proceedings of ICML (2000)