Reducing the Size of the Representation for the uDOP-Estimate

Christoph Teichmann
Abteilung Automatische Sprachverarbeitung
Institute of Computerscience
University of Leipzig
Max Planck Institute for Human Cognitive and Brain Sciences
Leipzig
teichmann@informatik.uni-leipzig.de

Abstract

The unsupervised Data Oriented Parsing (uDOP) approach has been repeatedly reported to achieve state of the art performance in experiments on parsing of different corpora. At the same time the approach is demanding both in computation time and memory. This paper describes an approach which decreases these demands. First the problem is translated into the generation of probabilistic bottom up tree automata (pBTA). Then it is explained how solving two standard problems for these automata results in a reduction in the size of the grammar. The reduction of the grammar size by using efficient algorithms for pBTAs is the main contribution of this paper. Experiments suggest that this leads to a reduction in grammar size by a factor of 2. This paper also suggests some extensions of the original uDOP algorithm that are made possible or aided by the use of tree automata.

1 Introduction

The approaches to unsupervised parsing given by Bod (2006a,2006b,2007) are all based on using all possible subtrees over a training corpus. This means that a great number of subtrees has to be represented. For every sentence the number of binary trees that can be proposed for that sentence[1] is given by the Catalan number of the length of the sentence. The number of subtrees for a tree in this set is exponential with respect to the length of the sentence.

In Bod (2007) a packed representation for all subtrees was proposed that is based on a technique for supervised Data Oriented Parsing (*DOP*) given in Goodman (2003). This paper aims to relate the problem of representing an estimate for all subtrees over a corpus to the field of tree automata (Fülöp and Vogler, 2009). With this step it will be possible to reduce the size of the packed representation of the subtrees even further. This newly formulated approach will also consider working with partially bracketed corpora.

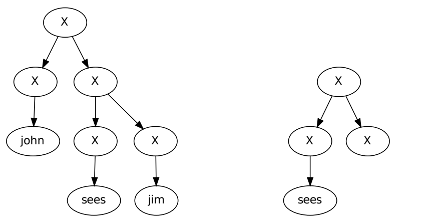
The next step in this paper will be a short discussion of uDOP. Then the necessary terminology is introduced. The reduction approach of this paper is given in section 4. In the final section it will be discussed how the step to tree automata creates additional possibilities to influence the final estimate produced by the uDOP estimator. In section 5 some evaluation results will be given for the decrease in grammar size that can be achieved by the techniques presented here.

2 A Short Discussion of uDOP

The unsupervised Data Oriented Parsing (uDOP) approach (Bod 2006a,2006b,2007) is defined by two steps. The first step is proposing every binary parse tree for each sentence in the corpus. This is followed by adding any subtree that occurs in the trees to the grammar as a possible derivation step. Since binary trees have more subtrees than nonbinary trees, the binary

Acknowledgments: The author would like to thank Amit Kirschbaum, Robert Remus, Anna Janska and the anonymus reviewers for their remarks.

[1]Only a single nonterminal X is used



(a) A possible tree proposed over a corpus sentence

(b) another possible subtree

Figure 1: an example for the uDOP approach

ones would always be the parses the approach prefers. Therefore the uDOP approach only uses the binary trees. The only nonterminal used is a new symbol usually refered to as 'X'.

The second step is estimation. For each subtree the number of occurrences in the proposed trees is counted. This number is divided by the sum of all occurrences of subtrees starting with the same nonterminal, which allows to derive a probability distribution over all trees.

If one takes the sentence 'john sees jim', one tree that can be proposed is shown in Figure 1(a). Then one possible subtree is shown in Figure 1(b). The subtree in Figure 1(b) would occur twice among the parses for the sentence 'jim sees the french guest', since there are two possible binary parses with the nonterminal 'X' for the substring 'the french guest'. One is given by

$$X(X(X(the)X(french))X(guest)) \tag{1}$$

the other is given by:

$$X(X(the)X(X(french)X(guest))) \tag{2}$$

In this paper a small extension of the original uDOP algorithm is considered. The idea is well known from Pereira and Schabes (1992). The extension is assuming that the corpus may consist of partial parses. The algorithm is changed so that for every partial tree all binary trees that are completions of the partial tree are proposed. Labels for the constituents in the partial tree are kept. Only a single nonterminal is used for the completions.

Take for example the sentence 'john sees the reporter'. If one is confident that 'the reporter' is a constituent of the type NP then the corpus entry would be:

$$X(X(john)X(sees)NP(X(the)X(reporter))) \tag{3}$$

This entry has two completions, the first one is given by:

$$X(X(X(john)X(sees))NP(X(the)X(reporter))) \tag{4}$$

The second one is given by:

$$X(X(john)X(X(sees)NP(X(the)X(reporter)))) \tag{5}$$

So making a parse complete means introducing additional brackets until the tree is binary. One may also consider not introducing brackets inside of existing brackets in order to allow for nonbinary branching.

These two parses contain subtrees starting and terminating with the nonterminal NP. This shows that such partial brackets and their class labels can create recursion on the introduced labels. These partial parses could come from other algorithms and reduce the final grammar size.

Approaches like the ones in Hänig (2010), Santamaria and Araujo (2010) and Ponvert et al. (2011) could be combined with the uDOP approach using this simple extension. All three approaches do not necessarily produce binary parse trees. This could be used to extend uDOP to nonbinary trees. Using the low level bracketings from the algorithms would reduce the size of an uDOP grammar estimated from them. Partial bracketing could also be approximated by using HTML annotation, punctuation and semantical annotation (Spitkovsky et al., 2010; Spitkovsky et al., 2011; Naseem and Barzilay, 2011).

3 Terminology

This section introduces stochastic tree substitution grammars. It will also introduce a version of probabilistic bottom up tree automata suited for representation of large stochastic tree substitution grammars. Furthermore it gives a more formal definition of the uDOP-estimate. Some definitions are not standard.[2]

The first definition necessary is the concept of trees.

Definition 1 (Trees). *The set of trees over leaf nodes L and internal nodes I is denoted by $T(L, I)$ and is defined as the smallest set conforming to:*

$$\forall \alpha \in (T(L, I) \cup L)^* : \forall y \in I : y(\alpha) \in T(L, I) \tag{6}$$

Where X^* denotes all tuples over the set X.[3]

If a tree has the form $y(\alpha)$ then $y \in I$ is called the root node. The leftmost node of an element $t \in (L \cup T(L, I))$ is denoted by $lm(t)$ and given by:

$$lm(t) = \begin{cases} t \text{ if } t \in L \\ lm(x_1) \text{ if } t = y(x_1, \ldots, x_n) \end{cases} \tag{7}$$

This definition basically states that trees are bracketed structures. Annotation gives the type of the bracket. Note that the definition of trees excludes trees that consist of only a single leaf node. This is a restriction that is common for STSGs.

The next element that needs to be defined is the concept of *extending a tree.* If a node in a tree has more than two daughters, then the tree can be extended. This is done by replacing two of the daughter nodes by a new node N labeled with any nonterminal and making the two removed daughter nodes the daughter nodes of the new node N. A *complete tree* is a tree that has no extensions. In other words, a complete tree has only nodes with less than two daughters. A tree t is a completion of the tree t' if t is complete and can be generated from t' by any number of completions. Next it is necessary to define subtrees.

Definition 2 (Subtrees). *Let*

$$\begin{aligned} t =& L(\ldots M(N_1(\ldots), \ldots, N_i(\alpha), \\ & \ldots, N_k(\ldots)) \ldots) \end{aligned}$$

be a tree then

$$\begin{aligned} t' =& M(N_1(\ldots), \ldots, N_i(\alpha), \\ & \ldots, N_k(\ldots)) \end{aligned}$$

is a direct subtree of t and if the root of α is in I then

$$\begin{aligned} t'' =& L(\ldots M(N_1(\ldots), \ldots, N_i(), \\ & \ldots, N_k(\ldots)) \ldots) \end{aligned}$$

is also a direct subtree of t. The set of subtrees for a tree t is denoted by $ST(t)$ and contains t and all direct subtrees of trees in $ST(t)$.

The first important fact about subtrees is that each node has either all or none of its daughters included in a subtree. The second important fact is that subtrees of less than two nodes are not allowed.

Definition 3 (Stochastic Tree Substitution Grammar). *A stochastic tree substitution grammar (STSG) is a tuple $\langle \Sigma, N, \tau, N_0, \omega \rangle$ where Σ is a finite alphabet of terminals, N is a finite set of nonterminals, $N_0 \in N$ is the start nonterminal, $\tau \subseteq T((\Sigma \cup N), N)$ is the set of trees*[4] *and $\omega : \tau \to \mathbb{R}^+$ is the weight function, where $\mathbb{R}^+$ is the set of positive real numbers.*

For space reasons it will not be discussed how a STSG defines a distribution over strings and trees. Note that since a CFG can be found that defines the same distribution over strings for every STSG (Goodman, 2003) similar constraints hold for STSGs and CFGs when it comes to defining proper distributions. In order to ensure that all string weights sum up to 1 the trees in

[2]No rank is assumed for the labels of trees, to give an example.

[3]The empty tuple is included.

[4]This set may be finite or infinite. The uDOP Estimate results in a finite set if the corpus is finite.

T for each possible root nonterminal must sum to one.[5]

Definition 4 (Probabilistic Bottom Up Tree Automaton)**.** *A probabilistic bottom up tree automaton (pBTA) is a tuple $\langle Q, \Sigma, \delta, q_0, \omega, \lambda\rangle$ where Q is a finite set of states, Σ is the finite alphabet, $\delta \subseteq Q^+ \times \Sigma \times Q$ is the finite set of transitions where Q^+ denotes all nonempty tuples over the states, q_0 is the start state, $\omega : \delta \to \mathbb{R}^+$ is the transition weight function and $\lambda : \delta \to \mathbb{R}^+$ is the final weight function.*

Definition 5 (Weight of a Tree in a pBTA)**.** *The weight of an element $t \in T(\Sigma, Q \times \Sigma \cup \{q_0\} \cup \Sigma)$ given an automaton $A = \langle Q, \Sigma, \delta, q_0, \omega, \lambda\rangle$ is denoted by $\Omega(t, A)$ and is defined by:*

$$\Omega(q_0, A) = 1 \tag{8}$$

$$\Omega(q, l(\alpha), A) = \sum_{\beta \in Q^n} \omega(\langle\langle\beta\rangle, l, q\rangle) \cdot \prod_{l_m(t_m) \in \alpha} \Omega(q_m, l_m(t_m), A) \tag{9}$$

Where $\alpha = l_1(t_1), \ldots, l_n(t_n)$ and $\beta = q_1, \ldots, q_n$. Where these formulas do not define a weight, it is assumed to be 0.

The final weight of the tree $t = l(l_1(t_1), \ldots, l_n(t_n))$ for the automaton A is denoted by $\Lambda(l(l_1(t_1), \ldots, l_n(t_n)), A)$ and is defined as:

$$\Lambda(l(\alpha), A) = \sum_{q \in Q} \sum_{\beta \in Q^n} \lambda(\langle\langle\beta\rangle, l, q\rangle) \cdot \prod_{t_m \in \alpha} \Omega(q_m, l_m(t_1), A) \tag{10}$$

Where again $\alpha = l_1(t_1), \ldots, l_n(t_n)$ and $\beta = q_1, \ldots, q_n$.

The definitions for pBTAs basically specify a bottom up parsing proceedure in which finished trees are combined. The intermediate trees are labeled with states that guide the derivation process.

Definition 6 (Language)**.** *The Language of a pBTA A denoted $L(A)$ is the set:*

$$L(A) = \{t | \Lambda(t, A) \neq 0\} \tag{11}$$

The penultimate set of definitions is concerned with the language weight of a pBTA, inside and outside weights.

Definition 7 (Language Weight)**.** *The language weight for a pBTA $A = \langle Q, \Sigma, \delta, q_0, \omega, \lambda\rangle$ is denoted by $wl(A)$ and defined by:*

$$wl(A) = \sum_{t \in T(\Sigma, \Sigma)} \Lambda(t, A) \tag{12}$$

The *inside weight* for a state $q \in Q$ for an automaton $A = \langle Q, \Sigma, \delta, q_0, \omega, \lambda\rangle$ is denoted by inside(q, A). It is the language weight of A'. Here A' is A changed so that it only has one final transition from $\langle q\rangle$ to some state with weight 1.

The *outside weight* for a state $q \in Q$ needs a recursive definition. The weight is made up of two summands. The first summand is the outside weight of the right hand side of all transitions q occurs in.[6] This is multiplied with the inside weight of all states other than q in the left hand side of the transition. Finally this value is taken times the number of occurrences of q in the left hand side. The second summand is the same as the first only with the outside weight replaced by the final weight of the transitions.

Now only the uDOP estimate and the connection between STSGs and pBTAs are still missing.

Definition 8 (uDOP Estimate)**.** *For a STSG $G = \langle \Sigma, \{X\}, T, N_0, \omega\rangle$ and a corpus $c = \langle c_1, \ldots, c_m\rangle$ such that each c_l is a tree of the form $L(L_1(x_1), \ldots, L_n(x_n))$ or an extension of such a tree. Let c' be derived from c by replacing each c_l by all the complete trees in $Ext(c_l)$. Then the uDOP estimate $uDOP(t, c)$ is given by:*

$$\omega(t) = \frac{\sum_{c_1 \in c'} num(t, c_1)}{\sum_{t' \in T(N, N \cup \Sigma)} \sum_{c_1 \in c'} num(t', c_1)} \tag{13}$$

where $num(t, x)$ is the number of times subtree t occurs in the tree x.

[5] Ensuring that the weight of the finite strings sums to one is more difficult. See Nederhof and Satta (2006).

[6] In a transition $\langle \alpha, l, q\rangle$ α is the left hand side and q the right hand side.

Here c' is a corpus that contains each completion t' once for every tree t in the original corpus, such that t' is a completion of t. This corpus is of course never generated explictly and only used in the definition.

Definition 9 (STSG Given by a pBTA)**.** *Let $G = \langle \Sigma, N, \tau, N_0, \omega \rangle$ be a STSG. The grammar is given by a pBTA A if $t \in T \leftrightarrow L(A)$ and $\omega(x) = \Omega(x, A)$.*

This definition states that the set of trees is the language of the automaton and the weight of each tree is the weight the automaton assigns to it.

The goal of this paper can now be described in the following way: given a corpus of trees $\langle c_1, \ldots, c_n \rangle$ find a pBTA A that gives the uDOP estimate and is as small as possible. The relevant measure here is the number of transitions. The number of states that are useful, the number of labels that are used and the number of entries for the weight function are all dependent on the number of transitions. This measure is also independent of any specific implementation details. From the connection between STSGs and pBTAs some extensions to the uDOP algorithm are possible that will be discussed at the end of the paper in section 6.

Only completion with the nonterminal X is used. All algorithms given in this paper can be adapted to more brackets by creating a transition for the additional labels whenever one for X is created.

4 Reducing the Size of the Estimate

The generation process for the uDOP estimate that this paper proposes is as follows. First generate a pBTA representing all the complete parse trees for the corpus. Every tree t in the corpus will have as its weight in the automaton the number of times it occurs in the completed corpus. The completed corpus is again the corpus with each tree replaced by all the trees completions.

As a second step manipulate the automaton for the set of completions in such a way that the set of subtrees is given and they are associated with the intended relative weights. Then apply normalization similar to the one employed by Maletti and Satta (2009). The normalization algorithm has to be slightly changed to account for the fact that the trees are not supposed to stand on their own, but rather be used in an STSG. A sketch will be given. For all final transition with label l sum up the final weight of the transition times the inside weight of all states on the left hand side of the transition. Then multiply the weight of final transitions with label l with the multiplication of the inside weights of their left hand side states and divide the weight by the sum for the label l. All other weights are normalized as described in Maletti and Satta (2009).

The reduction that will be proposed here is based on reducing the size of the representation of all trees. Once this is achieved, a simple algorithm can be applied that gives the uDOP estimate and only increases the size of the representation by a maximum factor of $2 \cdot |I| + |I|^2 + 1$ plus one transition for every nonterminal label.[7]

To understand the mapping to subtrees consider the following: If an automaton gives the set of all trees, then the outside weight of any state will be the number of trees this state is embedded in. The inside weight will be the number of trees embedded at this position. This is the case because inside and outside weights sum over the possible derivations.

For each nonterminal label l a state q_l is created only to represent the introduction of l. A transition of the form $\langle \langle q_0 \rangle, l, q_l \rangle$ is added to the representation of all trees.[8] This transition is weighted by 1.

Denote the automaton representing all trees by AT. Let $r = \langle \langle q_1, q_2 \rangle, X, q \rangle$ be a transition in AT. For each label l:

$$\text{inlab}(q_x, l) = \sum_{\langle \alpha, l, q_x \rangle \in \delta} \omega(r) \cdot \prod_{q_y \in \alpha} \text{inside}(q_x, AT) \tag{14}$$

[7] I is the set of labels that occur on internal nodes or - in other words - the nonterminal labels. The factor is explained further into the section. Note that for the standard uDOP approach $|I| = 1$.

[8] q_0 is assumed to be the start state.

For each nonterminal label l create the rules[9]:

$$r1 = \langle\langle q_l, q_2\rangle, X, q\rangle \quad (15)$$
$$r2 = \langle\langle q_1, q_l\rangle, X, q\rangle \quad (16)$$

For each pair of nonterminal labels l_1, l_2 create a rule[10]:

$$r3 = \langle\langle q_{l_1}, q_{l_2}\rangle, X, q\rangle \quad (17)$$

Let w be the weight of the original transition. Set $\omega(r1) = in(q_2, l) \cdot w$,$\omega(r2) = in(q_1, l) \cdot w$ and $\omega(r3) = in(q_1, l_1) \cdot in(q_2, l_2) \cdot w$ respectively. Add final weight $out(q)$ to each transition.[11] This assigns to each subtree the number of counts. After the transformation each transition can be a point at which a derivation ends. Outside weight is assigned according to the number of trees the subtree is embedded in. The derivation can also start with any node. Therefore inside weight is added according to the number of embedded trees.

Normalizing the automaton afterwards gives the weights according to the uDOP estimator.

Bansal and Klein (2010) give a transformation from parse trees to subtrees that reduces the size of the representation even further. Since a version of the transformation from their paper can be applied to any representation of the full parse trees, it is complementary to the approach used here. For this reason it will not be discussed here and it should suffice to say that using this transformation would improve the results in this paper even further.

Before it is discussed how the size of the representation of all trees can be reduced further, the first step will be to present the approach by Goodman (2003).

4.1 The Goodman Reduction

The approach from Goodman (2003) was intended for use with the supervised version of Data Oriented Parsing. We will discuss a version of the algorithm that is based on tree automata and the considerations made so far.

The original approach works by creating a state for every node in the corpus. Each group of daughter nodes is then connected to its mother node by a weighted transition with weight 1. The transition from the daughters of the root node of a sentence is assigned a final weight of 1. Finally, the projection to the subtrees is applied.

The version for unsupervised parsing is similar to and based on parse forests and parse items. The states correspond to parsing items as in the CYK algorithm.[12]

The reduction can be described as follows: create a state/parse item $\langle i, j, k\rangle$ for every sentence k and every range from i to j in the sentence. Also create one state for every type of terminal node. This is illustrated in Figures 2(a) and 2(b). Rules are introduced from the start state for each possible terminal to the terminal type nodes with weight 1. If terminal x occurs in sentence k from i to $i+1$, create a transition from the terminal state for x to the state $\langle i, i+1, k\rangle$ with weight 1. Label those transitions with X or with the appropriate preterminal nonterminal if there is one in the partial corpus tree. All states with a difference greater than 1 between the start and the end point are connected to all state pairs $\langle i, m, k\rangle, \langle m, j, k\rangle$. Here m is a point between i and j. These transitions are again labeled by X unless there is a bracket labeled by L from i to j in this case the transition is labeled by L. The weight for each such transition is 1.

If i is 0 and j is the length of the sentence number k then the final weight for transitions to the state $\langle i, j, k\rangle$ is 1.

In order to comply with the requirement that we only use completions of the given trees, one adjustment is necessary. When a bracket a, b is present, no state i, j, k is proposed such that $a < i < b < j \vee i < a < j < b$. Thereby all crossing brackets are ruled out.

[9]This accounts for the factor $2 \cdot |I|$.

[10]This accounts for the factor $|I|^2$.

[11]An actual implementation would not create a rule if all weights are 0.

[12]See for example (Younger, 1967).

4.2 Making the Representation of All Trees Deterministic

A possible step in size reduction is making the representation deterministic. Generally determinization does not lead to a reduction in size of a finite state automaton. Here however, determinization means simply that states representing equivalent subtrees are merged. This is similar to the graph packing idea used in Bansal and Klein (2010).

Assume a partial tree is given in the string form that was used in section 3, i.e. , as a string of brackets and symbols. Then two identical strings represent identical trees which have identical completions. Let the bracketing for sequence from i to j in sentence k be identical to the bracketing for the sequence from l to m in sentence n. Assume also that the sequences represent the same string. Then the state $\langle i, j, k\rangle$ may be replaced in every transition by the state $\langle l, m, n\rangle$. The only thing that has to be kept track of is how many times a certain string corresponded to a full corpus entry. In the Goodman approach a final weight of 1 is used, since new states are created for every sentence. In the deterministic case the final weight for all transitions reaching a state that represents a bracketed sequence x must be increased by 1 for each time x occurs in the corpus. An illustration of the idea[13] is given in Figures 2(c) and 2(d).

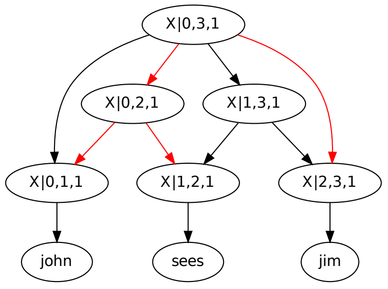
4.3 Using Minimization

Finally one can try finding the minimal deterministic weighted tree automaton for the distribution. This is a well defined notion.

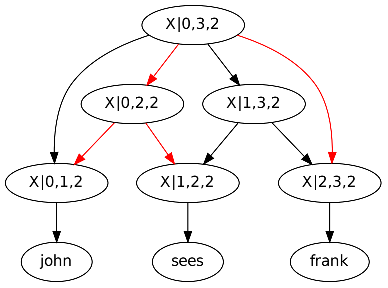
Definition 10 (Minimal deterministic pBTA)**.** *The minimal deterministic pBTA* $A' = \langle Q', \Sigma', \delta', q'_0, \omega', \lambda'\rangle$ *for a given pBTA* $A = \langle Q, \Sigma, \delta, q_0, \omega, \lambda\rangle$ *fulfills* $\Omega(x, p) = \Omega(x, A')$ *and there is no automaton* $A'' = \langle Q'', \Sigma'', \delta'', q''_0, \omega'', \lambda''\rangle$ *fulfilling this criterion with* $|Q''| < |Q'|$.

A minimal deterministic pBTA is unique for the distribution it represents up to renaming the states.

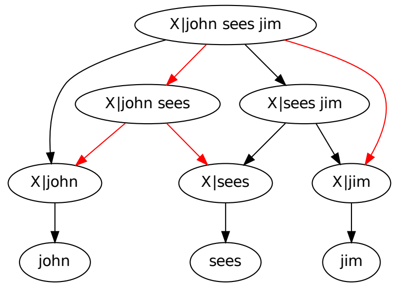
[13] Here shown without any bracketing data



(a) Goodman reduction states



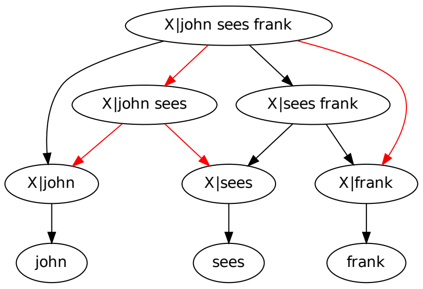
(b) Goodman reduction states



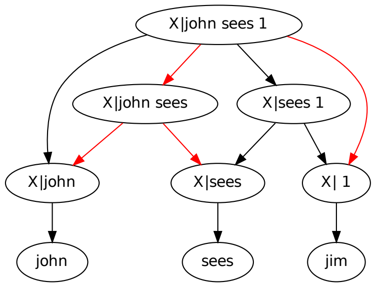
(c) Deterministic reduction states

Note that this means that after the minimization the automaton is as small as possible for a deterministic pBTA. The only way to improve on this while staying in the pBTA framework would be to find a minimal nondeterministic automaton. That this is possible is shown in Bozapalidis (1991). It is however not clear that this problem could be solved in reasonable time for an automaton with hundreds of thousands of states.

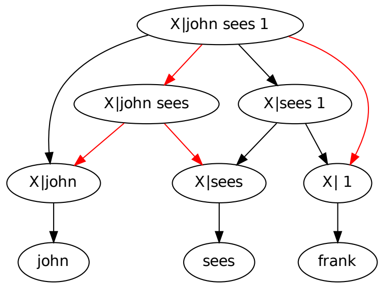
In order to generate a minimal automaton, an efficiently verifiable criterion for two states



(d) Deterministic reduction states



(e) Minimization reduction states



(f) Minimization reduction states

Figure 2: the different reduction approaches illustrated, different edge colors correspond to different parses. The Figures 2(b) and 2(a) are for the Goodman approach. Every span of words has its own state proposed. Figures 2(c) and 2(d) show how equals spans of words will lead to the repetition of the same state in the deterministic approach. Figures 2(e) and 2(f) show how two states that have equal contexts are merged into one state called '1'. This is an illustration of the minimization approach.

to be equivalent is necessary. For deterministic pBTA this is given by Maletti (2009). Since the automaton for all trees is nonrecursive after the deterministic construction, normalization allows minimizing the automaton in linear time, depending on the number of transitions.[14] For the algorithms to work, the fact that a deterministic pBTA is constructed is a necessary precondition.

Figures 2(e) and 2(f) illustrate this approach. The tree $X(jim)$ is distributed equally to the tree $X(frank)$. Since this is the case, a merged state for both trees is introduced. This state is labeled as 1.

5 Experimental Results

The algorithm was tested in two domains. The first one was the Negra Corpus (Skut et al., 1997). The second one was the Brown Corpus (Francis, 1964). The standard approach in unsupervised parsing is to use sequences of tags with certain punctuation removed (Klein and Manning, 2002). This is supposed to simulate spoken language. Once the punctuation is removed all sequences of length 10 or less are used for most approaches in unsupervised parsing. This ensures that the hypothesis space is relatively small for the sentences left in the corpus. The same approach is chosen for this paper, as this is the context in which uDOP grammars are most likely to be evaluated. A slightly different definition of punctuation is used. Note that no bracketing structure is used. This means that for every string in the corpus, a number of transitions has to be created that is limited by n^3 in the worst case, were n is the length of the string.

Note that the removal of more punctuation marks will lead to a sample that is harder to reduce in size by determinization and minimization. Punctuation occurs frequently and can therefore easily net a great number of reductions by merging states.

For the Negra Corpus all tags matching
/\$ \$S^*$/
are removed.[15] This leads to a corpus of 7248

[14]The normalization can also be implemented in linear time for nonrecursive pBTA.

[15]Here
/\$ \$S^*$/
is an regular expression according to the ruby regular expression specifications (Flanagan and Matsumoto,

	negra	brown5000
Goodman Based	1528256	1238717
Deterministic	857150	785427
Minimized	633907	602491
	brown10000	brown15000
Goodman Based	2389442	3603050
Deterministic	1402536	2030252
Minimized	1029786	1457499

Table 1: The results from the experimental evaluation. The numbers given reflect the number of transitions after the transformation to subtrees.

tag sequences of length 10 or less.

For the Brown Corpus the tags that are removed are specified by the regular expression

$/\backslash W^{+}/$

Not the whole sample from the Brown Corpus is used. Instead samples of 5000,10000 and 15000 sequences of tags are used.

The results of the algorithms can be seen in Table 5. In order to make the comparison implementation independent, the number of transitions after the transformation to subtrees, as explained in section 4, is given.

The results show that the minimization algorithm tends to cut the number of transitions in half for all corpora. This means these reductions in the number of transitions could be used to double the size of the corpus used in uDOP.[16] Note that if one was to extend the corpus with more strings of limited size the benefit of the new approaches should become more pronounced. This is the case since the determinstic construction only introduces one state per observed substring. The set of possible tag sequences of length 10 or less is limited. This holds especially true if one considers linguistic constraints. This tendency can be seen from the statistics for 15000 sentences from the Brown corpus.

2008).

[16]This is the case, since the number of states grows linearly with corpus size for fixed sentence length.

6 Possible Extensions

Note that tree automata are closed under intersection (Fülöp and Vogler, 2009). Bansal and Klein (2010) propose improving a DOP estimate by changing the weights of the subtrees. This is done by using a weighting scheme that distributes along the packed representation. This can be extended with the techniques in this paper in the following way: Assume one wants to give the weight of the subtree as the joint probability of a tree automaton model that has previously been given and the uDOP estimate. All that is necessary to achieve this would be to represent the uDOP estimate as a tree automaton, intersect it with the previously given automaton and apply a normalization as discussed above.[17].

The algorithm allows another generalization in addition to the one proposed. This is the case since the mapping to subtrees can be implemented by application of a tree transducer (Knight and Graehl, 2005). Therefore, the final estimation can be made more complex. Simply replace the mapping step by the application of a transducer.

7 Conclusion

In this paper it was discussed how the size of the unsupervised Data Oriented Parsing estimate for STSGs can be reduced. By translating the problem into the domain of finite tree automata, the problem of reducing the grammar size could be handled by solving standard problems in that domain.

The code used for the experiments in this paper can be found at `http://code.google.com/p/gragra/`.

References

Mohit Bansal and Dan Klein. 2010. Simple, accurate parsing with an all-fragments grammar. In *ACL 2010, Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, Uppsala, Sweden*, pages 1098–1107. The Association for Computer Linguistics.

[17]the last step is necessary for the subtree probabilities to sum to 1

Rens Bod. 2006a. An all-subtrees approach to unsupervised parsing. In *ACL-44: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 865–872. The Association for Computational Linguistics.

Rens Bod. 2006b. Unsupervised parsing with u-dop. In *CoNLL-X '06: Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 85–92. Association for Computational Linguistics.

Rens Bod. 2007. Is the end of supervised parsing in sight? In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, pages 400–407. The Association for Computer Linguistics.

Symeon Bozapalidis. 1991. Effective construction of the syntactic algebra of a recognizable series on trees. *Acta Informatica*, 28(4):351–363.

David Flanagan and Yukihiro Matsumoto. 2008. *The ruby programming language.* O'Reilly, first edition.

W. Nelson Francis. 1964. A standard sample of present-day english for use with digital computers. Technical report, Brown University.

Zoltan Fülöp and Heiko Vogler, 2009. *Weighted Tree Automata and Tree Transducers*, chapter 9, pages 313–394. Springer Publishing Company, Incorporated, 1st edition.

Joshua Goodman. 1998. *Parsing Inside-Out.* Ph.D. thesis, Harvard University.

Joshua Goodman. 2003. Efficient algorithms for the dop model. In *Data Oriented Parsing.* Center for the Study of Language and Information, Stanford, California.

Christian Hänig. 2010. Improvements in unsupervised co-occurrence based parsing. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, pages 1–8. Association for Computational Linguistics.

Dan Klein and Christopher D. Manning. 2002. A generative constituent-context model for improved grammar induction. In *Proceedings of the Association for Computational Linguistics*, pages 128–135. Association for Computational Linguistics.

Kevin Knight and Jonathan Graehl. 2005. An overview of probabilistic tree transducers for natural language processing. In *CICLing*, volume volume 3406 of Lecture Notes in Computer Science, pages 1–24.

Andreas Maletti and Giorgio Satta. 2009. Parsing algorithms based on tree automata. In *IWPT '09: Proceedings of the 11th International Conference on Parsing Technologies*, pages 1–12. Association for Computational Linguistics.

Andreas Maletti. 2009. Minimizing deterministic weighted tree automata. *Information and Computation*, 207(11):1284–1299.

Tahira Naseem and Regina Barzilay. 2011. Using semantic cues to learn syntax. In *AAAI 2011: Twenty-Fifth Conference on Artificial Intelligence.*

Mark-Jan Nederhof and Giorgio Satta. 2006. Estimation of consistent probabilistic context-free grammars. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 343–350, Morristown, NJ, USA. Association for Computational Linguistics.

Fernando Pereira and Yves Schabes. 1992. Inside-outside reestimation from partially bracketed corpora. In *Proceedings of the 30th annual meeting on Association for Computational Linguistics*, ACL '92, pages 128–135, Stroudsburg, PA, USA. Association for Computational Linguistics.

Elias Ponvert, Jason Baldridge, and Katrin Erk. 2011. Simple unsupervised grammar induction from raw text with cascaded finite state models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies.*

Jesus Santamaria and Lourdes Araujo. 2010. Identifying patterns for unsupervised grammar induction. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning (CoNLL).* Association for Computational Linguistics.

Wojciech Skut, Brigitte Krenn, Thorsten Brants, and Hans Uszkoreit. 1997. An annotation scheme for free word order languages. In *Proceedings of the fifth conference on Applied natural language processing*, ANLC '97, pages 88–95. Association for Computational Linguistics.

Valentin I. Spitkovsky, Daniel Jurafsky, and Hiyan Alshawi. 2010. Profiting from mark-up: hyper-text annotations for guided parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 1278–1287, Stroudsburg, PA, USA. Association for Computational Linguistics.

Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. 2011. Punctuation: Making a point in unsupervised dependency parsing. In *In Proceedings of the Fifteenth Conference on Computational Natural Language Learning (CoNLL-2011).*

Daniel H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10:189–208.