Accurate Parsing with Compact Tree-Substitution Grammars: Double-DOP

Federico Sangati and Willem Zuidema
Institute for Logic, Language and Computation
University of Amsterdam
Science Park 904, 1098 XH Amsterdam, The Netherlands
{f.sangati,zuidema}@uva.nl

Abstract

We present a novel approach to Data-Oriented Parsing (DOP). Like other DOP models, our parser utilizes syntactic fragments of arbitrary size from a treebank to analyze new sentences, but, crucially, it uses only those which are encountered at least twice. This criterion allows us to work with a relatively small but representative set of fragments, which can be employed as the symbolic backbone of several probabilistic generative models. For parsing we define a transform-backtransform approach that allows us to use standard PCFG technology, making our results easily replicable. According to standard Parseval metrics, our best model is on par with many state-of-the-art parsers, while offering some complementary benefits: a simple generative probability model, and an explicit representation of the larger units of grammar.

1 Introduction

Data-oriented Parsing (DOP) is an approach to wide-coverage parsing based on assigning structures to new sentences using fragments of variable size extracted from a treebank. It was first proposed by Scha in 1990 and formalized by Bod (1992), and preceded many developments in statistical parsing (e.g., the "treebank grammars" of Charniak 1997) and linguistic theory (e.g., the current popularity of "constructions", Jackendoff 2002). A rich literature on DOP has emerged since, yielding state-of-the-art results on the Penn treebank benchmark test (Bod, 2001; Bansal and Klein, 2010) and inspiring developments in related frameworks including tree kernels (Collins and Duffy, 2002), reranking (Charniak and Johnson, 2005) and Bayesian adaptor and fragment grammars (e.g., Johnson et al., 2007; O'Donnell et al., 2009; Cohn et al., 2010). By formalizing the idea of using large fragments of earlier language experience to analyze new sentences, DOP captures an important property of language cognition that has shaped natural language. It therefore complements approaches that have focused on properties like lexicalization or incrementality, and might bring supplementary strengths in other NLP tasks.

Early versions of DOP (e.g., Bod et al., 2003) aimed at extracting all subtrees of all trees in the treebank. The total number of constructions, however, is prohibitively large for non-trivial treebanks: it grows exponentially with the length of the sentences, yielding the astronomically large number of approximately 10^{48} for section 2-21 of the Penn WSJ corpus. These models thus rely on a big sample of fragments, which inevitably includes a substantial portion of overspecialized constructions. Later DOP models have used the Goodman transformation (Goodman, 1996, 2003) to obtain a compact representation of all fragments in the treebank (Bod, 2003; Bansal and Klein, 2010). In this case the grammatical constructions are no longer explicitly represented, and substantial engineering effort is needed to optimally tune the models and make them efficient.

In this paper we present a novel DOP model (Double-DOP) in which we extract a restricted yet representative subset of fragments: those recurring at least twice in the treebank. The explicit representation of the fragments allows us to derive simple

ways of estimating probabilistic models on top of the symbolic grammar. This and other implementation choices aim at making the methodology transparent and easily replicable. The accuracy of Double-DOP is well within the range of state-of-the-art parsers currently used in other NLP-tasks, while offering the additional benefits of a simple generative probability model and an explicit representation of grammatical constructions.

The contributions of this paper are summarized as follows: (i) we describe an efficient tree-kernel algorithm which allows us to extract all recurring fragments, reducing the set of potential elementary units from the astronomical 10^{48} to around 10^6. (ii) We implement and compare different DOP estimation techniques to induce a probability model (PTSG) on top of the extracted symbolic grammar. (iii) We present a simple transformation of the extracted fragments into CFG-rules that allows us to use off-the-shelf PCFG parsing and inference. (iv) We integrate Double-DOP with recent state-splitting approaches (Petrov et al., 2006), yielding an even more accurate parser and a better understanding of the relation between DOP and state-splitting.

The rest of the paper is structured as follows. In section 2 we describe the symbolic backbone of the grammar formalism that we will use for parsing. In section 3 we illustrate the probabilistic extension of the grammar, including our transformation of PTSGs to PCFGs that allows us to use a standard PCFG parser, and a different transform that allows us to use a standard implementation of the inside-outside algorithm. In section 4 we present the experimental setup and the results.

2 The symbolic backbone

The basic idea behind DOP is to allow arbitrarily large fragments from a treebank to be the elementary units of production of the grammar. Fragments can be combined through *substitution* to obtain the phrase-structure tree of a new sentence. Figure 1 shows an example of a complete syntactic tree obtained by combining three elementary fragments. As in previous work, two fragments f_i and f_j can be combined ($f_i \circ f_j$) only if the leftmost substitution site $X\downarrow$ in f_i has the same label as the root node of f_j; in this case the resulting tree will correspond to f_i with f_j replacing X. The DOP formalism is discussed in detail in e.g., Bod et al. (2003).

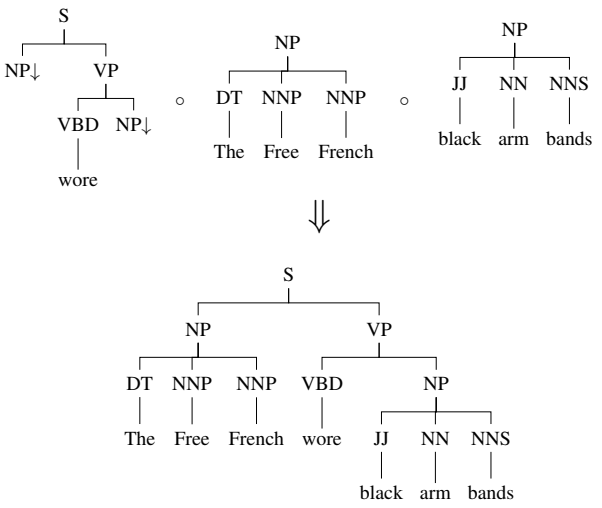

Figure 1: An example of a derivation of a complete syntactic structure (below) obtained combining three elementary fragments (above) by means of the substitution operation ∘. Substitution sites are marked with ↓.

2.1 Finding Recurring Fragments

The first step to build a DOP model is to define its symbolic grammar, i.e. the set of elementary fragments in the model. In the current work we explicitly extract a subset of fragments from the training treebank. To limit the fragment set size, we use a simple but heretofore unexplored constraint: we extract only those fragments that occur two or more times in the treebank[1]. Extracting this particular set of fragments is not trivial, though: a naive approach that filters a complete table of fragments together with their frequencies fails because that set, in a reasonably sized treebank, is astronomically large. Instead, we use a dynamic programming algorithm based on tree-kernel techniques (Collins and Duffy, 2001; Moschitti, 2006; Sangati et al., 2010).

The algorithm iterates over every pair of trees in

[1]More precisely we extract only the largest shared fragments for all pairs of trees in the treebank. All subtrees of these extracted fragments necessarily also occur at least twice, but they are only explicitly represented in our extracted set if they happen to form a largest shared fragment from another pair of trees. Hence, if a large tree occurs twice in the treebank the algorithm will extract from this pair only the full tree as a fragment and not all its (exponentially many) subtrees.

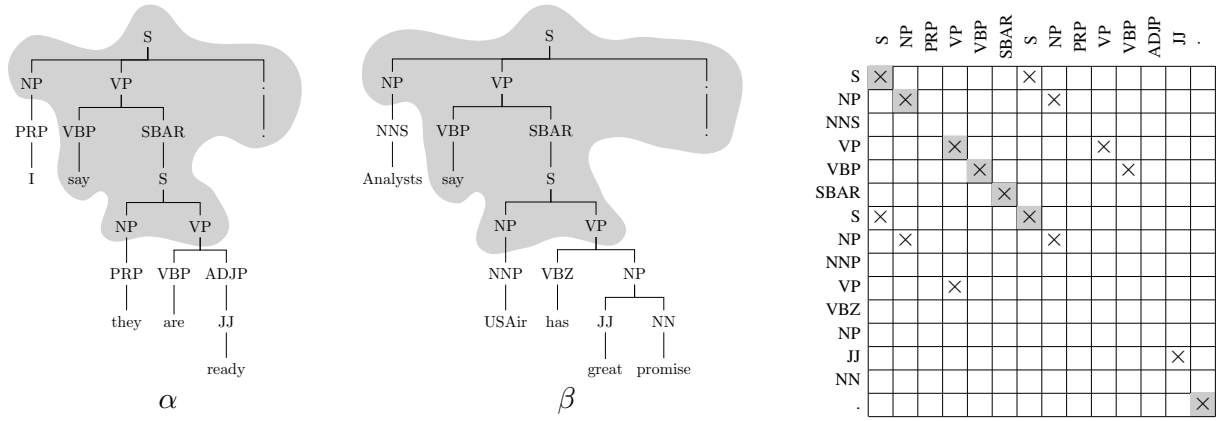

Figure 2: Left: example of two trees sharing a single maximum fragment, circled in the two trees. Right: the chart $\mathcal{M}$ which is used in the dynamic algorithm to extract all maximum fragments shared between the two trees. The highlighted cells in the chart are the ones which contribute to extract the shared fragment. The marked cells are those for which the corresponding nodes in the two tree have equivalent labels but differ in their lists of child nodes.

the treebank to look for common fragments. Figure 2 shows an example of a pair of trees $\langle \alpha, \beta \rangle$ being compared. The algorithm builds a chart $\mathcal{M}$ with one column for every indexed non-terminal node α_i in α, and one row for every indexed non-terminal node β_j in β. Each cell $\mathcal{M}\langle i, j \rangle$ identifies a set of indices corresponding to the largest fragment in common between the two trees starting from α_i and β_j. This set is empty if α_i and β_j differ in their labels, or they don't have the same list of child nodes. Otherwise (if both the labels and the lists of children match) the set is computed recursively as follows:

$$\mathcal{M}\langle i, j \rangle = \{\alpha_i\} \cup \left(\bigcup_{c=\{1,2,\ldots,|ch(\alpha)|\}} \mathcal{M}\langle ch(\alpha_i, c), ch(\beta_j, c) \rangle \right) \quad (1)$$

where $ch(\alpha)$ returns the indices of α's children, and $ch(\alpha, c)$ the index of its c^{th} child.

After filling the chart, the algorithm extracts the set of recurring fragments, and stores them in a table to keep track of their counts. This is done by converting back each fragment implicitly defined in every cell-set[2], and filtering out those that are properly contained in others.

In a second pass over the treebank, exact counts are obtained for each fragment in the extracted set. Parse trees in the training corpus are not necessarily covered entirely by recurring fragments; to ensure coverage, we also include in the symbolic backbone of our Double-DOP model all PCFG-productions not included in the set of extracted fragments.

2.2 Comparison with previous DOP work

Explicit grammars The number of recurring fragments in our symbolic grammar, extracted from the training sections of the Penn WSJ treebank[3], is around 1 million, and thus is significantly lower than previous work extracting explicit fragments (e.g., Bod, 2001, used more than 5 million fragments up to depth 14).

When looking at the extracted fragments we ask if we could have predicted which fragments occur twice or more. Figure 3 attempts to tackle this question by reporting some statistics on the extracted fragments. The majority of fragments are rather small with a limited number of words or substitution sites in the frontier. Yet, there is a significant portion of fragments, in the tail of the distribution, with more than 10 words or substitution sites. Since the space of all fragments with such characteristics is enormously large, selecting big recurring fragments using random sampling technique is like finding a needle in a haystack. Hence, random sampling processes (like Bod, 2001), will tend to represent fre-

[2] A cell-set containing a single index corresponds to the fragment including the node with that index together with all its children.

[3] This is after the treebank has been preprocessed. See also section 4.

quent recurring constructions such as *from NP to NP* or *whether S or not*, together with infrequent overspecialized fragments like *from Houston to NP*, while missing large generic constructions such as *everything you always wanted to know about NP but were afraid to ask*. These large constructions are excluded completely by models that only allow elementary trees up to a certain depth (typically 4 or 5) into the symbolic grammar (Zollmann and Sima'an, 2005; Zuidema, 2007; Borensztajn et al., 2009), or only elementary trees with exactly one lexical anchor (Sangati and Zuidema, 2009).

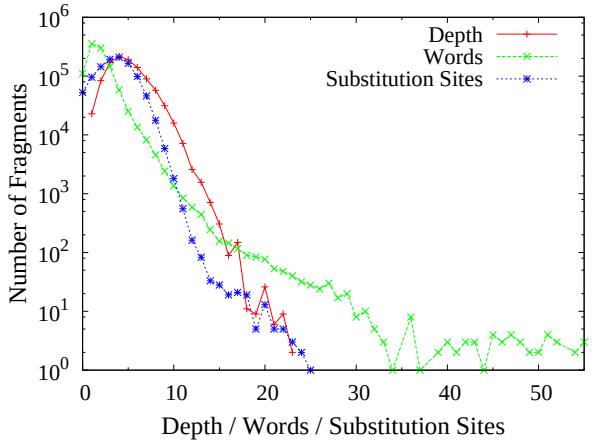

Figure 3: Distribution of the recurring fragments types according to several features: depth, number of words, and number of substitution sites. Their corresponding curves peak at 4 (depth), 1 (words), and 4 (substitution sites).

Implicit grammars Goodman (1996, 2003) defined a transformation for some versions of DOP to an equivalent PCFG-based model, with the number of rules extracted from each parse tree linear in the size of the trees. This transform, representing larger fragments only *implicitly*, is used in most recent DOP parsers (e.g., Bod, 2003; Bansal and Klein, 2010). Bod has promoted the Goodman transform as the solution to the computational challenges of DOP (e.g., Bod, 2003); it's important to realize, however, that the resulting grammars are still very large: WSJ sections 2-21 yield about 2.5 million rules in the basic version of Goodman's transform. Moreover, the transformed grammars differ from untransformed DOP grammars in that larger fragments are no longer explicitly represented. Rather, information about their frequency is distributed over many CFG-rules: if a construction occurs n times and contains m context-free productions, Goodman's transform uses the weights of $7nm + m$ rules to encode this fact. Thus, the information that the idiomatic fragment *(PP (IN "out") (PP (IN "of") (NP (NN "town")))))* occurs 3 times in WSJ sections 2-21, is distributed over 132 rules. This way, an attractive feature of DOP, viz. the explicit representation of the 'productive units' of language, is lost[4].

In addition, grammars that implicitly encode all fragments found in a treebank are strongly biased to over-represent big constructions: the great majority of the entire set of fragments belongs in fact to the largest tree in the treebank[5]. DOP models relying on Goodman's transform, need therefore to counteract this tendency. Bansal and Klein (2010), for instance, rely on a sophisticated tuning technique to correctly adjust the weights of the rules in the grammar. In our Double-DOP approach, instead, the number of fragments extracted from each tree varies much less (it ranges between 4 and 1,759). This comparison is shown in figure 4.

3 The probabilistic model

Like CFG grammars, our symbolic model produces extremely many parse trees for a given test sentence. We therefore need to disambiguate between the possible parses by means of a probability model that assigns probabilities to fragments, and defines a proper distribution over the set of possible full parse trees. For every nonterminal X in the treebank we have:

$$\sum_{f \in F_X} p(f) = 1 \qquad (2)$$

where F_X is the set of fragments in our symbolic grammar rooted in X. A derivation $d = f_1, f_2, \ldots, f_n$ of t is a sequence of the fragments that through left-most substitution produces t. The probability of a derivation is computed as the product of

[4]Bansal and Klein (2010) address this issue for contiguous constructions by extending the Goodman transform with a 'Packed Graph Encoding' for fragments that "bottom out in terminals". However, constructions with variable slots, such as *whether S or not*, are left unchanged.

[5]In fact, the number of extracted fragments increase exponentially with the size of the tree.

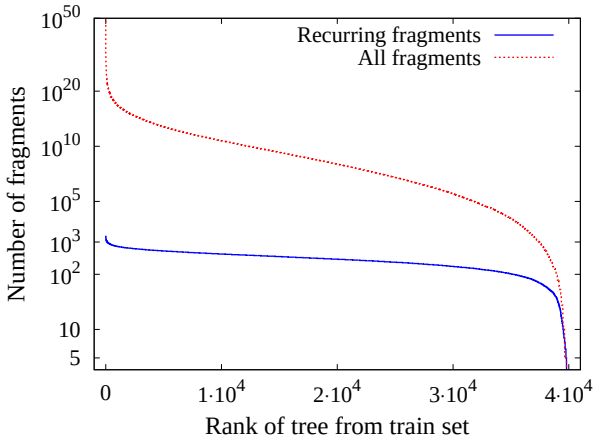

Figure 4: Number of fragments extracted from each tree in sections 2-21 of the WSJ treebank, when considering all-fragments (dotted line) and recurring-fragments (solid line). Trees on the x-axis are ranked according to the number of fragments. Note the double logarithmic scale on the y-axis.

the probability of each of its fragments.

$$P(d) = \prod_{f \in d} p(f) \quad (3)$$

In section 3.2 we describe ways of obtaining different probability distributions over the fragments in our grammar. In the following section we assume a given probabilistic model, and illustrate how to use standard PCFG parsing.

3.1 Parsing

It is possible to define a simple transform of our probabilistic fragment grammar, such that off-the-shelf parsers can be used. In order to perform the PTSG/PCFG conversion, every fragment in our grammar must be mapped to a CFG rule which will keep the same probability as the original fragment. The corresponding rule will have as the left hand side the root of the fragment and as the right hand side its yield, i.e., a sequence of terminals and non-terminals (substitution sites).

It might occur that several fragments are mapped to the same CFG rule[6]. These are interesting cases of syntactic ambiguity as shown in figure 5. In order to resolve this problem we need to map each ambiguous fragment to two unique CFG rules chained by a unique artificial node, as shown at the bottom of the same figure. To the first CFG rule in the chain we assign the probability of the fragment, while the second will receive probability 1, so the product gives back the original probability. The ambiguous and unambiguous PTSG/PCFG mappings need to be stored in a table, in order to convert back the compressed CFG derivations to the original PTSG model after parsing.

Such a transformed PCFG will generate the same derivations as the original PTSG grammar with identical probabilities. In our experiment we use a standard PCFG parser to produce a list of k-best Viterbi derivations. These, in turn, will be used to maximize possible objectives as described in section 3.3.

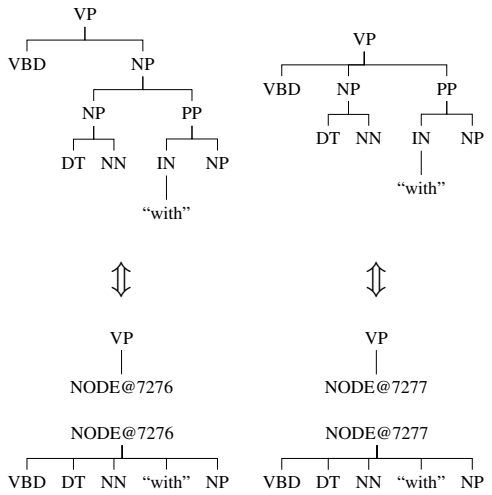

Figure 5: Above: example of 2 ambiguous fragments mapping to the same CFG rule VP → VBD DT NN "with" NP. The first fragment occurs 5 times in the training treebank, (e.g. in the sentence *was an executive with a manufacturing concern*) while the second fragment occurs 4 times (e.g. in the sentence *began this campaign with such high hopes*). Below: the two pairs of CFG rules that are used to map the two fragments to separate CFG derivations.

3.2 Inducing probability distributions

Relative Frequency Estimate (RFE) The simplest way to assign probabilities to fragments is to make them proportional to their counts[7] in the training set. When enforcing equation 2, that gives the

[6] In our binarized treebank we have 31,465 fragments types that are ambiguous in this sense.

[7] We refer to the counts of each fragment as returned by our extraction algorithm in section 2.1.

Relative Frequency Estimate (RFE):

$$p_{\text{RFE}}(f) = \frac{count(f)}{\sum_{f' \in F_{root(f)}} count(f')} \quad (4)$$

Unlike RFE for PCFGs, however, the RFE for PTSGs has no clear probabilistic interpretation. In particular, it does not yield the maximum likelihood solution, and when used as an estimator for an all-fragments grammar, it is strongly biased since it assigns the great majority of the probability mass to big fragments (Johnson, 2002). As illustrated in figure 4 this bias is much weaker when restricting the set of fragments with our approach. Although this does not solve all theoretical issues, it makes RFE a reasonable first choice again.

Equal Weights Estimate (EWE) Various other ways of choosing the weights of a DOP grammar have been worked out. The best empirical results have been reported by Bod (2003) with the EWE proposed by Goodman (2003). Goodman defined it for grammars in the Goodman transform, but for explicit grammars it becomes:

$$w_{\text{EWE}}(f) = \sum_{t \in TB} \frac{count(f,t)}{|\{f' \in t\}|} \quad (5)$$

$$p_{\text{EWE}}(f) = \frac{w_{\text{EWE}}(f)}{\sum_{f' \in F_{root(f)}} w_{\text{EWE}}(f')} \quad (6)$$

where the first sum is over all parse trees t in the treebank (TB), $count(f,t)$ gives the number of times fragment f occurs in t, and $|\{f' \in t\}|$ is the total number of subtrees of t that were included in the symbolic grammar.

Maximum Likelihood (ML) For reestimation, we can aim at maximizing the likelihood (ML) of the treebank. For this, it turns out that we can define another transformation of our PTSG, such that we can apply standard Inside-Outside algorithm for PCFGs (Lari and Young, 1990). The original version of IO is defined over *string rewriting* PCFGs, and maximizes the likelihood of the training set consisting of plain sentences. Reestimation shifts probability mass between alternative parse trees for a sentence. In contrast, our grammars consist of fragments of various size, and our training set consists of parse trees. Reestimation here shifts probability mass between alternative derivations for a parse tree.

Our transformation approach is illustrated with an example in figure 6. In step (b) the fragments in the grammar as well as the original parse trees in the treebank are "flattened" into bracket notation. In step (c) each fragment is transformed into a CFG rule in the transformed meta-grammar, whose right-hand side is constituted by the bracket notation of the fragment. Each substitution site $X\!\downarrow$ is raised to a meta-nonterminal X', and all other symbols, including parentheses, become meta-terminals. The left-hand side of the rule is constituted by the original root symbol R of the fragment raised to a meta-nonterminal R'.

The resulting PCFG generates trees in bracket notation, and we can run an of-the-shelf inside-outside algorithm by presenting it parse trees from the train corpus in bracket notation[8]. In the experiments that we report below we used the RFE from section 3, to generate the initial weights for the grammar.

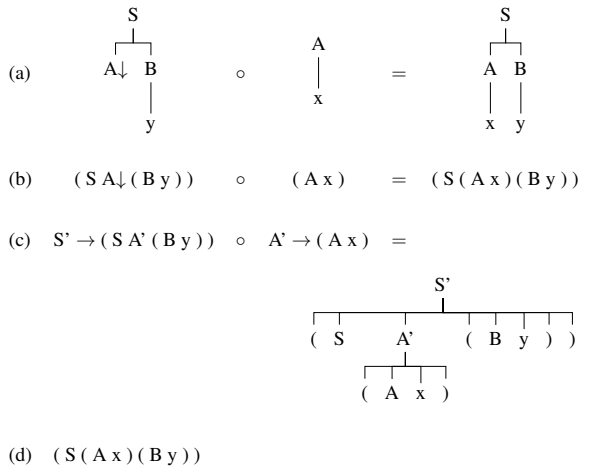

Figure 6: Rule and tree transforms that turn PTSG reestimation into PCFG reestimation; (a) a derivation of the sentence $x\,y$ through successive substitutions of elementary trees from a PTSG; (b) the same elementary trees and resulting parse tree in bracket notation; (c) an equivalent derivation with the meta-grammar, where the original substitution sites reappear as meta-nonterminals (marked with a prime) and all other symbols as meta-terminals; (d) the yield of the derivation in c.

[8] However, the results with inside-outside reported in this paper were obtained with an earlier version of our code that uses an equivalent but special-purpose implementation.

3.3 Maximizing Objectives

MPD The easiest objective in parsing, is to select the most probable derivation (MPD), obtained by maximizing equation 3.

MPP A DOP grammar can often generate the same parse tree t through different derivations $D(t) = d_1, d_2, \ldots d_m$. The probability of t is therefore obtained by summing the probabilities of all its possible derivations.

$$P(t) = \sum_{d \in D(t)} p(d) = \sum_{d \in D(t)} \prod_{f \in d} p(f) \qquad (7)$$

An intuitive objective for a parser is to select, for a given sentence, the parse tree with highest probability according to equation 7, i.e., the most probable parse (MPP): unfortunately, identifying the MPP is computationally intractable (Sima'an, 1996). However, we can approximate the MPP by deriving a list of k-best derivations, summing up the probabilities of those resulting in the same parse tree, and select the tree with maximum probability.

MCP, MRS Following Goodman (1998), Sima'an (1999, 2003), and others, we also consider other objectives, in particular, the max constituent parse (MCP), and the max rule sum (MRS).

MCP maximizes a weighted average of the expected labeled recall L/N_C and (approximated) labeled precision L/N_G under the given posterior distribution, where L is the number of correctly labeled constituents, N_C the number of constituents in the correct tree, and N_G the number of constituents in the guessed tree. Recall is easy to maximize since the estimated N_C is constant. L/N_C can be in fact maximized in:

$$\hat{t} = \arg\max_t \sum_{lc \in t} P(lc) \qquad (8)$$

where lc ranges over all labeled constituents in t and $P(lc)$ is the marginalized probability of all the derivation trees in the grammar yielding the sentence under consideration which contains lc.

Precision, instead, is harder because the denominator N_G depends on the chosen guessed tree. Goodman (1998) proposes to look at another metric which is strongly correlated with precision, which is the mistake rate $(N_G - L)/N_C$ that we want to minimize. We combine recall with mistake rate through linear interpolation:

$$\hat{t} = \arg\max_t \mathcal{E}(\frac{L}{N_C} - \lambda\frac{N_G - L}{N_C}) \qquad (9)$$

$$= \arg\max_t \sum_{lc \in t} P(lc) - \lambda(1 - P(lc)) \qquad (10)$$

where 10 is obtained from 9 assuming N_C constant, and the optimal level for λ has to be evaluated empirically.

Unlike MPP, the MCP can be calculated efficiently using dynamic programming techniques over the parse forest. However, in line with the aims of this paper to produce an easily reproducible implementation of DOP, we developed an accurate approximation of the MCP using a list of k-best derivations, such as those that can be obtained with an off-the-shelf PCFG parser.

We do so by building a standard CYK chart, where every cell corresponds to a specific span in the test sentence. We store in each cell the probability of seeing every label in the grammar yielding the corresponding span, by marginalizing the probabilities of all the parse trees in the obtained k-best derivations that contains that label covering the same span. We then compute the Viterbi-best parse maximizing equation 10.

We implement max rule sum (MRS) in a similar way, but do not only keep track of labels in every cell, but of each CFG rule that span the specific yield (see also Sima'an, 1999, 2003). We haven't implemented the max rule product (MRP) where posteriors are multiplied instead of added (Petrov and Klein, 2007; Bansal and Klein, 2010).

4 Experimental Setup

In order to build and test our Double-DOP model[9], we employ the Penn WSJ Treebank (Marcus et al., 1993). We use sections 2-21 for training, section 24 for development and section 23 for testing.

Treebank binarization We start with some preprocessing of the treebank, following standard prac-

[9]The software produced for running our model is publicly available and included in the supplementary material to this paper. To the best of our knowledge this is the first DOP software released that can be used to parse the WSJ PTB.

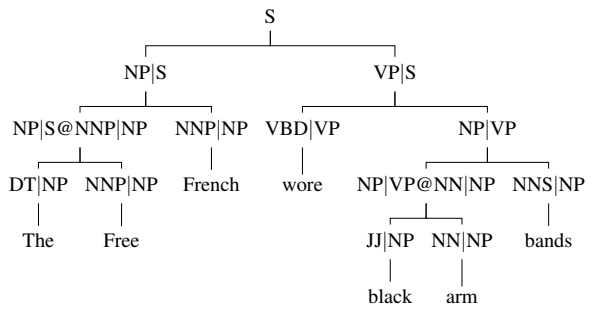

Figure 7: The binarized version of the tree in figure 1, with H=1 and P=1.

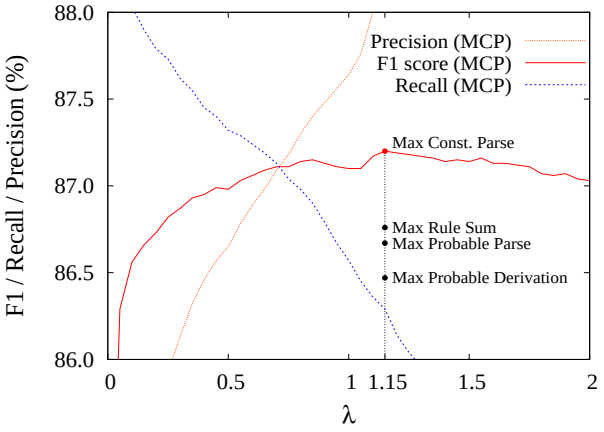

Figure 8: Double-DOP results on the development section (≤ 40) with different maximizing objectives.

tice in WSJ parsing. We remove traces and functional tags. We apply a left binarization of the training treebank as in Matsuzaki et al. (2005) and Klein and Manning (2003), setting the horizontal history H=1 and the parent labeling P=1. This means that when a node has more than 2 children, the i^{th} child (for $i \geq 3$) is conditioned on child $i-1$. Moreover the labels of all non-lexical nodes are enriched with the labels of their parent node. Figure 7 shows the binarized version of the tree structure in figure 1.

Unknown words We replace words appearing less than 5 times in the training data by one of 50 unknown word categories based on the presence of lexical features as implemented in Petrov (2009). In some of the experiments we also perform a smoothing over the lexical elements assigning low counts ($\epsilon = 0.01$) to open-class $\langle$words, PoS-tags$\rangle$ pairs not encountered in the training corpus[10].

Fragment extraction We extract the symbolic grammar and fragment frequencies from this preprocessed treebank as explained in section 2. This is the the most time-consuming step (around 160 CPU hours[11]).

In the extracted grammar we have in total 1,029,342 recurring fragments and 17,768 unseen CFG rules. We test several probability distributions over the fragments (section 3.2) and various maximization objectives (section 3.3).

[10] A PoS-tag is an open class if it rewrites to at least 50 different words in the training corpus. A word is an open class word if it has been seen only with open-class PoS-tags.

[11] Although our code could still be optimized further, it does already allow for running the job on M CPUs in parallel, reducing the time required by a factor M (10 hours with 16-CPUs).

Parsing We convert our PTSG into a PCFG (section 3.1) and use `Bitpar`[12] for parsing. For approximating MPP and other objectives we marginalize probabilities from the 1,000 best derivations.

4.1 Results

We start by presenting in figure 8 the results we obtain on the development set (section 24). Here we compare the *maximizing objectives* presented in section 3.3, using RFE to obtain the probability distribution over the fragments. We conclude that, empirically, MCP for $\lambda = 1.15$, is the best choice to maximize F1, followed by MRS, MPP, and MPD.

We also compare the various *estimators* presented in section 3.2, on the same development set, keeping MCP with $\lambda = 1.15$ as the maximizing objective. We find that RFE is the best estimator (87.2 F1[13]) followed by EWE (86.8) and ML (86.6). Our best results with ML are obtained when removing fragments occurring less than 6 times (apart from CFG-rules) and when stopping at the second iteration. This filtering is done in order to limit the number of big fragments in the grammar. It is well known that IO for DOP tends to assign most of the probability mass to big fragments, quickly overfitting the training data. It is surprising that EWE and ML perform worse than RFE, in contrast to earlier findings (Bod, 2003).

[12] `http://www.ims.uni-stuttgart.de/tcl/SOFTWARE/BitPar.html`

[13] We computed F1 scores with EvalB (`http://nlp.cs.nyu.edu/evalb/`) using parameter file *new.prm*.

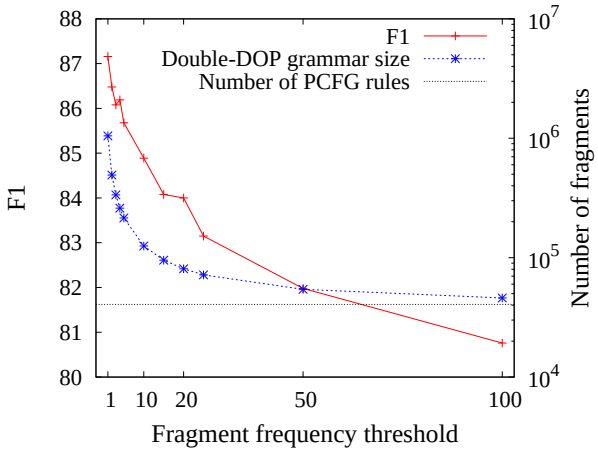

Figure 9: Performance (on the development set) and size of Double-DOP when considering only fragments whose occurring frequency in the training treebank is above a specific threshold (x-axis). In all cases, all PCFG-rules are included in the grammars. For instance, at the right-hand side of the plot a grammar is evaluated which included only 6754 fragments with a frequency > 100 as well as 39227 PCFG rules.

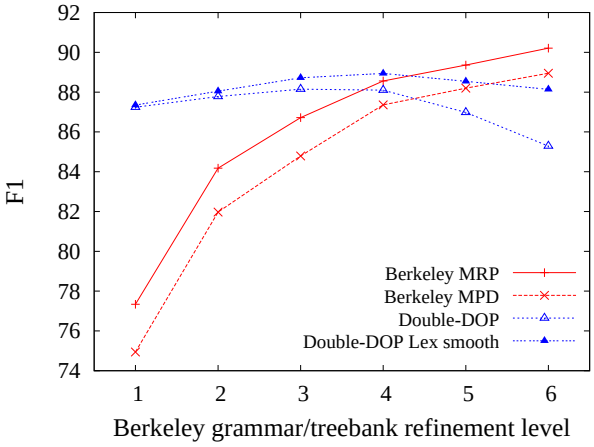

Figure 10: Comparison on section 24 between the performance of Double-DOP (using RFE and MCP with $\lambda = 1.15$, H=0, P=0) and Berkeley parser on different stages of refinement of the treebank/grammar.

We also investigate how a further restriction on the set of extracted fragments influences the performance of our model. In figure 9 we illustrate the performance of Double-DOP when restricting the grammar to fragments having frequencies greater than $1, 2, \ldots, 100$. We can notice a rather sharp decrease in performance as the grammar becomes more and more compact.

Next, we present some results on various Double-DOP grammars extracted from the same training treebank after refining it using the Berkeley state-splitting model[14] (Petrov et al., 2006; Petrov and Klein, 2007). In total we have 6 increasingly refined versions of the treebank, corresponding to the 6 cycles of the Berkeley model. We observe in figure 10 that our grammar is able to benefit from the state splits for the first four levels of refinement, reaching the maximum score at cycle 4, where we improve over our base model. For the last two data points, the treebank gets too refined, and using Double-DOP model on top of it, no longer improves accuracy.

We have also compared our best Double-DOP base model and the Berkeley parser on per-category performance. Here we observe an interesting trend: the Berkeley parser outperforms Double-DOP on very frequent categories, while Double-DOP performs better on infrequent ones. A detailed comparison is included in table 1.

Finally, in table 2 we present our results on the test set (section 23). Our best model (according to the best settings on the development set) performs slightly worse than the one by Bansal and Klein (2010) when trained on the original corpus, but outperforms it (and the version of their model with additional refinements) when trained on the refined version, in particular for the exact match score.

5 Conclusions

We have described Double-DOP, a novel DOP approach for parsing, which uses all constructions recurring at least twice in a treebank. This methodology is driven by the linguistic intuition that constructions included in the grammar should prove to be reusable in a representative corpus.

The extracted set of fragments is significantly smaller than in previous approaches. Moreover constructions are explicitly represented, which makes them potentially good candidates as semantic or translation units to be used in other applications.

Despite earlier reported excellent results with DOP parsers, they are almost never used in other

[14] We use the Berkeley grammar labeler following the base settings for the WSJ: trees are right-binarized, H=0, and P=0. Berkeley parser package is available at `http://code.google.com/p/berkeleyparser/`

Category label	% in gold	F1 Berkeley	F1 Double-DOP
NP	41.42	**91.4**	89.5
VP	20.46	**90.6**	88.6
S	13.38	**90.7**	87.6
PP	12.82	**85.5**	84.1
SBAR	3.47	**86.0**	82.1
ADVP	3.36	**82.4**	81.0
ADJP	2.32	**68.0**	67.3
QP	0.98	82.8	**84.6**
WHNP	0.88	**94.5**	92.0
WHADVP	0.33	**92.8**	91.9
PRN	0.32	**83.0**	77.9
NX	0.29	**9.50**	7.70
SINV	0.28	**90.3**	88.1
SQ	0.14	**82.1**	79.3
FRAG	0.10	26.4	**34.3**
SBARQ	0.09	84.2	**88.2**
X	0.06	72.0	**83.3**
NAC	0.06	54.6	**88.0**
WHPP	0.06	**91.7**	44.4
CONJP	0.04	55.6	**66.7**
LST	0.03	**61.5**	33.3
UCP	0.03	30.8	**50.0**
INTJ	0.02	44.4	**57.1**

Table 1: Comparison of the performance (per-category F1 score) on the development set between the Berkeley parser and the best Double-DOP model.

Parsing Model	test (≤ 40) F1	test (≤ 40) EX	test (all) F1	test (all) EX
PCFG Baseline				
PCFG (H=1, P=1)	77.6	17.2	76.5	15.9
PCFG (H=1, P=1) Lex smooth.	78.5	17.2	77.4	16.0
FRAGMENT-BASED PARSERS				
Zuidema (2007)*	83.8	26.9	-	-
Cohn et al. (2010) MRS	85.4	27.2	84.7	25.8
Post and Gildea (2009)	82.6	-	-	-
Bansal and Klein (2010) MCP	88.5	33.0	87.6	30.8
Bansal and Klein (2010) MCP + Additional Refinement	88.7	33.8	88.1	31.7
THIS PAPER				
Double-DOP	87.7	33.1	86.8	31.0
Double-DOP Lex smooth.	**87.9**	**33.7**	**87.0**	**31.5**
Double-DOP-Sp	88.8	35.9	88.2	33.8
Double-DOP-Sp Lex smooth.	**89.7**	**38.3**	**89.1**	**36.1**
REFINEMENT-BASED PARSERS				
Collins (1999)	88.6	-	88.2	-
Petrov and Klein (2007)	**90.6**	**39.1**	**90.1**	**37.1**

Table 2: Summary of the results of different parsers on the test set (sec 23). Double-DOP experiments use RFE, MCP with $\lambda = 1.15$, H=1, P=1; those on state-splitting (Double-DOP-Sp) use Berkeley cycle 4, H=0, P=0. Results from Petrov and Klein (2007) already include smoothing which is performed similarly to our smoothing technique (see section 4). (* Results on a development set, with sentences up to length 20.)

NLP tasks: where other successful parsers often feature as components of machine translation, semantic role labeling, question-answering or speech recognition systems, DOP is conspicuously absent in these neighboring fields (but for a possible application of closely related formalisms see, e.g., Yamangil and Shieber, 2010). The reasons for this are many, but most important are probably the computational inefficiency of many instances of the approach, the lack of downloadable software and the difficulties with replicating some of the key results.

In this paper we have addressed all three obstacles: our efficient algorithm for identifying the recurrent fragments in a treebank runs in polynomial time. The transformation to PCFGs that we define allows us to use a standard PCFG parser, while retaining the benefit of explicitly representing larger fragments. A different transform also allows us to run the popular inside-outside algorithm. Although IO results are slightly worse than with the naive relative frequency estimate, it is important to establish that the standard method for dealing with latent information (i.e., the derivations of a given parse) is not the best choice in this case. We expect that other re-estimation methods, for instance Variational Bayesian techniques, could be formulated in the same manner.

Finally, the availability of our programs, as well as the third party software that we use, also addresses the replicability issue. Where some researchers in the field have been skeptical of the DOP approach to parsing, we believe that our independent development of a DOP parser adds credibility to the idea that an approach that uses very many large subtrees, can lead to very accurate parsers.

Acknowledgments

We gratefully acknowledge funding by the Netherlands Organization for Scientific Research (NWO): FS is funded through a Vici-grant "Integrating Cognition" (277.70.006) to Rens Bod, and WZ through a Veni-grant "Discovering Grammar" (639.021.612). We also thank Rens Bod, Gideon Borensztajn, Jos de Bruin, Andreas van Cranenburgh, Phong Le, Remko Scha, Khalil Sima'an and the anonymous reviewers for very useful comments.

References

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

Rens Bod. 1992. A computational model of language performance: Data oriented parsing. In *Proceedings COLING'92 (Nantes, France)*, pages 855–859. Association for Computational Linguistics, Morristown, NJ.

Rens Bod. 2001. What is the minimal set of fragments that achieves maximal parse accuracy? In *Proceedings of the ACL*. Morgan Kaufmann, San Francisco, CA.

Rens Bod. 2003. An efficient implementation of a new DOP model. In *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics - Volume 1*, EACL '03, pages 19–26. Association for Computational Linguistics, Morristown, NJ, USA.

Rens Bod, Khalil Sima'an, and Remko Scha. 2003. *Data-Oriented Parsing*. University of Chicago Press, Chicago, IL, USA.

Gideon Borensztajn, Willem Zuidema, and Rens Bod. 2009. Children's Grammars Grow More Abstract with Age—Evidence from an Automatic Procedure for Identifying the Productive Units of Language. *Topics in Cognitive Science*, 1(1):175–188.

Eugene Charniak. 1997. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 598–603. AAAI Press/MIT Press.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proc. 43nd Meeting of Association for Computational Linguistics (ACL 2005)*.

Trevor Cohn, Phil Blunsom, and Sharon Goldwater. 2010. Inducing tree-substitution grammars. *Journal of Machine Learning Research*, 11:3053–3096.

Michael Collins and Nigel Duffy. 2001. Convolution Kernels for Natural Language. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *NIPS*, pages 625–632. MIT Press.

Michael Collins and Nigel Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of 40th Annual Meeting of the ACL*, pages 263–270. Association for Computational Linguistics, Philadelphia, Pennsylvania, USA.

Michael J. Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Joshua Goodman. 1996. Efficient algorithms for parsing the DOP model. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 143–152.

Joshua Goodman. 2003. Efficient parsing of DOP with PCFG-reductions. In Bod et al. (2003).

Joshua T. Goodman. 1998. *Parsing inside-out*. Ph.D. thesis, Harvard University, Cambridge, MA, USA.

Ray Jackendoff. 2002. *Foundations of Language*. Oxford University Press, Oxford, UK.

Mark Johnson. 2002. The dop estimation method is biased and inconsistent. *Computational Linguistics*, 28:71–76.

Mark Johnson, Thomas L. Griffiths, and Sharon Goldwater. 2007. Adaptor grammars: A framework for specifying compositional nonparametric bayesian models. In *Advances in Neural Information Processing Systems*, volume 16, pages 641–648.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *ACL '03: Proceedings of the 41st Annual Meeting on ACL*, pages 423–430. Association for Computational Linguistics, Morristown, NJ, USA.

K. Lari and S. J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic cfg with latent annotations. In *ACL '05: Proceedings of the 43rd Annual Meeting on ACL*, pages 75–82. Association for Computational Linguistics, Morristown, NJ, USA.

Alessandro Moschitti. 2006. Efficient Convolution Kernels for Dependency and Constituent Syntactic Trees. In *ECML*, pages 318–329. Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Proceedings, Berlin, Germany.

Timothy J. O'Donnell, Noah D. Goodman, and Joshua B. Tenenbaum. 2009. Fragment Grammars: Exploring Computation and Reuse in Language. Technical Report MIT-CSAIL-TR-2009-013, MIT.

Slav Petrov. 2009. *Coarse-to-Fine Natural Language Processing*. Ph.D. thesis, University of California at Bekeley, Berkeley, CA, USA.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *ACL-44: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL*, pages 433–440. Association for Computational Linguistics, Morristown, NJ, USA.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the ACL; Proceedings of the Main Conference*, pages 404–411. Association for Computational Linguistics, Rochester, New York.

Matt Post and Daniel Gildea. 2009. Bayesian learning of a tree substitution grammar. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 45–48. Association for Computational Linguistics, Suntec, Singapore.

Federico Sangati and Willem Zuidema. 2009. Unsupervised Methods for Head Assignments. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 701–709. Association for Computational Linguistics, Athens, Greece.

Federico Sangati, Willem Zuidema, and Rens Bod. 2010. Efficiently extract recurring tree fragments from large treebanks. In *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)*. European Language Resources Association (ELRA), Valletta, Malta.

Remko Scha. 1990. Taaltheorie en taaltechnologie: competence en performance. In Q. A. M. de Kort and G. L. J. Leerdam, editors, *Computertoepassingen in de Neerlandistiek*, LVVN-jaarboek, pages 7–22. Landelijke Vereniging van Neerlandici, Almere. [Language theory and language technology: Competence and Performance] in Dutch.

Khalil Sima'an. 1996. Computational complexity of probabilistic disambiguation by means of tree-grammars. In *Proceedings of the 16th conference on Computational linguistics*, pages 1175–1180. Association for Computational Linguistics, Morristown, NJ, USA.

Khalil Sima'an. 1999. *Learning Efficient Disambiguation*. Ph.D. thesis, Utrecht University and University of Amsterdam.

Khalil Sima'an. 2003. On maximizing metrics for syntactic disambiguation. In *Proceedings of the International Workshop on Parsing Technologies (IWPT'03)*.

Elif Yamangil and Stuart M. Shieber. 2010. Bayesian synchronous tree-substitution grammar induction and its application to sentence compression. In *Proceedings of the 48th Annual Meeting of the ACL*, ACL '10, pages 937–947. Association for Computational Linguistics, Stroudsburg, PA, USA.

Andreas Zollmann and Khalil Sima'an. 2005. A consistent and efficient estimator for data-oriented parsing. *Journal of Automata, Languages and Combinatorics*, 10(2/3):367–388.

Willem Zuidema. 2007. Parsimonious Data-Oriented Parsing. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 551–560. Association for Computational Linguistics, Prague, Czech Republic.