Simple, Accurate Parsing with an All-Fragments Grammar

Mohit Bansal and **Dan Klein**
Computer Science Division
University of California, Berkeley
{mbansal, klein}@cs.berkeley.edu

Abstract

We present a simple but accurate parser which exploits both large tree fragments and symbol refinement. We parse with *all* fragments of the training set, in contrast to much recent work on tree selection in data-oriented parsing and tree-substitution grammar learning. We require only simple, deterministic grammar symbol refinement, in contrast to recent work on latent symbol refinement. Moreover, our parser requires no explicit lexicon machinery, instead parsing input sentences as character streams. Despite its simplicity, our parser achieves accuracies of over 88% F1 on the standard English WSJ task, which is competitive with substantially more complicated state-of-the-art lexicalized and latent-variable parsers. Additional specific contributions center on making implicit all-fragments parsing efficient, including a coarse-to-fine inference scheme and a new graph encoding.

1 Introduction

Modern NLP systems have increasingly used data-intensive models that capture many or even all substructures from the training data. In the domain of syntactic parsing, the idea that all training fragments[1] might be relevant to parsing has a long history, including tree-substitution grammar (data-oriented parsing) approaches (Scha, 1990; Bod, 1993; Goodman, 1996a; Chiang, 2003) and tree kernel approaches (Collins and Duffy, 2002). For machine translation, the key modern advancement has been the ability to represent and memorize large training substructures, be it in contiguous phrases (Koehn et al., 2003) or syntactic trees (Galley et al., 2004; Chiang, 2005; Deneefe and Knight, 2009). In all such systems, a central challenge is efficiency: there are generally a combinatorial number of substructures in the training data, and it is impractical to explicitly extract them all. On both efficiency and statistical grounds, much recent TSG work has focused on fragment selection (Zuidema, 2007; Cohn et al., 2009; Post and Gildea, 2009).

At the same time, many high-performance parsers have focused on *symbol refinement* approaches, wherein PCFG independence assumptions are weakened not by increasing rule sizes but by subdividing coarse treebank symbols into many subcategories either using structural annotation (Johnson, 1998; Klein and Manning, 2003) or lexicalization (Collins, 1999; Charniak, 2000). Indeed, a recent trend has shown high accuracies from models which are dedicated to inducing such subcategories (Henderson, 2004; Matsuzaki et al., 2005; Petrov et al., 2006). In this paper, we present a simplified parser which combines the two basic ideas, using both large fragments and symbol refinement, to provide non-local and local context respectively. The two approaches turn out to be highly complementary; even the simplest (deterministic) symbol refinement and a basic use of an all-fragments grammar combine to give accuracies substantially above recent work on tree-substitution grammar based parsers and approaching top refinement-based parsers. For example, our best result on the English WSJ task is an F1 of over 88%, where recent TSG parsers[2] achieve 82-84% and top refinement-based parsers[3] achieve 88-90% (e.g., Table 5).

Rather than select fragments, we use a simplification of the PCFG-reduction of DOP (Goodman,

[1] In this paper, a *fragment* means an elementary tree in a tree-substitution grammar, while a *subtree* means a fragment that bottoms out in terminals.

[2] Zuidema (2007), Cohn et al. (2009), Post and Gildea (2009). Zuidema (2007) incorporates deterministic refinements inspired by Klein and Manning (2003).

[3] Including Collins (1999), Charniak and Johnson (2005), Petrov and Klein (2007).

Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, pages 1098–1107,
Uppsala, Sweden, 11-16 July 2010.

1996a) to work with all fragments. This reduction is a flexible, implicit representation of the fragments that, rather than extracting an intractably large grammar over fragment *types*, indexes all nodes in the training treebank and uses a compact grammar over indexed node *tokens*. This indexed grammar, when appropriately marginalized, is equivalent to one in which all fragments are explicitly extracted. Our work is the first to apply this reduction to *full-scale* parsing. In this direction, we present a coarse-to-fine inference scheme and a compact graph encoding of the training set, which, together, make parsing manageable. This tractability allows us to avoid selection of fragments, and work with all fragments.

Of course, having a grammar that includes all training substructures is only desirable to the extent that those structures can be appropriately weighted. Implicit representations like those used here do not allow arbitrary weightings of fragments. However, we use a simple weighting scheme which does decompose appropriately over the implicit encoding, and which is flexible enough to allow weights to depend not only on frequency but also on fragment size, node patterns, and certain lexical properties. Similar ideas have been explored in Bod (2001), Collins and Duffy (2002), and Goodman (2003). Our model empirically affirms the effectiveness of such a flexible weighting scheme in full-scale experiments.

We also investigate parsing without an explicit lexicon. The all-fragments approach has the advantage that parsing down to the character level requires no special treatment; we show that an explicit lexicon is not needed when sentences are considered as strings of characters rather than words. This avoids the need for complex unknown word models and other specialized lexical resources.

The main contribution of this work is to show practical, tractable methods for working with an all-fragments model, without an explicit lexicon. In the parsing case, the central result is that accuracies in the range of state-of-the-art parsers (i.e., over 88% F1 on English WSJ) can be obtained with no sampling, no latent-variable modeling, no smoothing, and even no explicit lexicon (hence negligible training overall). These techniques, however, are not limited to the case of monolingual parsing, offering extensions to models of machine translation, semantic interpretation, and other areas in which a similar tension exists between the desire to extract many large structures and the computational cost of doing so.

2 Representation of Implicit Grammars

2.1 All-Fragments Grammars

We consider an *all-fragments grammar* G (see Figure 1(a)) derived from a binarized treebank B. G is formally a tree-substitution grammar (Resnik, 1992; Bod, 1993) wherein each subgraph of each training tree in B is an elementary tree, or *fragment* f, in G. In G, each derivation d is a tree (multiset) of fragments (Figure 1(c)), and the weight of the derivation is the product of the weights of the fragments: $\omega(d) = \prod_{f \in d} \omega(f)$. In the following, the derivation weights, when normalized over a given sentence s, are interpretable as conditional probabilities, so G induces distributions of the form $P(d|s)$.

In models like G, many derivations will generally correspond to the same unsegmented tree, and the parsing task is to find the tree whose sum of derivation weights is highest: $t_{max} = \arg\max_t \sum_{d \in t} \omega(d)$. This final optimization is intractable in a way that is orthogonal to this paper (Sima'an, 1996); we describe minimum Bayes risk approximations in Section 4.

2.2 Implicit Representation of G

Explicitly extracting all fragment-rules of a grammar G is memory and space intensive, and impractical for full-size treebanks. As a tractable alternative, we consider an *implicit grammar* G^I (see Figure 1(b)) that has the same posterior probabilities as G. To construct G^I, we use a simplification of the PCFG-reduction of DOP by Goodman (1996a).[4] G^I has *base* symbols, which are the symbol types from the original treebank, as well as *indexed* symbols, which are obtained by assigning a unique index to each node token in the training treebank. The vast majority of symbols in G^I are therefore indexed symbols. While it may seem that such grammars will be overly large, they are in fact reasonably compact, being linear in the treebank size B, while G is exponential in the length of a sentence. In particular, we found that G^I was smaller than explicit extraction of all depth 1 and 2 unbinarized fragments for our

[4]The difference is that Goodman (1996a) collapses our BEGIN and END rules into the binary productions, giving a larger grammar which is less convenient for weighting.

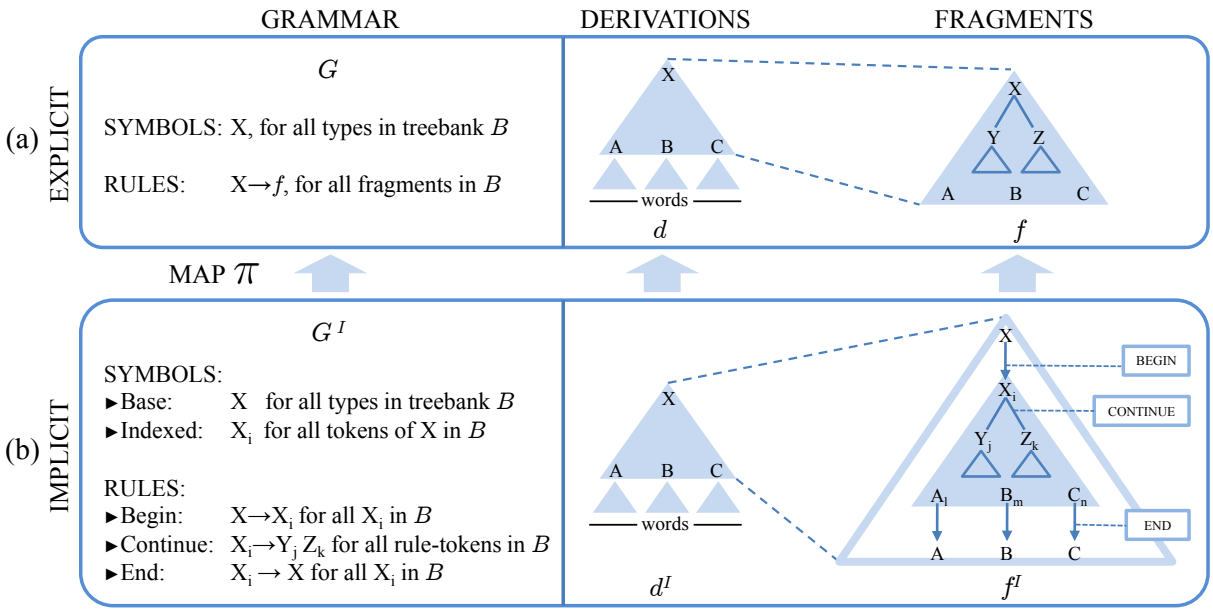

Figure 1: Grammar definition and sample derivations and fragments in the grammar for (a) the explicitly extracted all-fragments grammar G, and (b) its implicit representation G^I.

treebanks – in practice, even just the raw treebank grammar grows almost linearly in the size of B.[5]

There are 3 kinds of rules in G^I, which are illustrated in Figure 1(d). The BEGIN rules transition from a base symbol to an indexed symbol and represent the beginning of a fragment from G. The CONTINUE rules use only indexed symbols and correspond to specific depth-1 binary fragment tokens from training trees, representing the internal continuation of a fragment in G. Finally, END rules transition from an indexed symbol to a base symbol, representing the frontier of a fragment.

By construction, all derivations in G^I will segment, as shown in Figure 1(d), into regions corresponding to *tokens* of fragments from the training treebank B. Let π be the map which takes appropriate fragments in G^I (those that begin and end with base symbols and otherwise contain only indexed symbols), and maps them to the corresponding f in G. We can consider any derivation d^I in G^I to be a tree of fragments f^I, each fragment a token of a fragment type $f = \pi(f^I)$ in the original grammar G. By extension, we can therefore map any derivation d^I in G^I to the corresponding derivation $d = \pi(d^I)$ in G.

The mapping π is an onto mapping from G^I to G. In particular, each derivation d in G has a non-empty set of corresponding derivations $\{d^I\} = \pi^{-1}(d)$ in G^I, because fragments f in d correspond to multiple fragments f^I in G^I that differ only in their indexed symbols (one f^I per occurrence of f in B). Therefore, the set of derivations in G is preserved in G^I. We now discuss how weights can be preserved under π.

2.3 Equivalence for Weighted Grammars

In general, arbitrary weight functions ω on fragments in G do not decompose along the increased locality of G^I. However, we now consider a usefully broad class of weighting schemes for which the posterior probabilities under G of derivations d are preserved in G^I. In particular, assume that we have a weighting ω on rules in G^I which does not depend on the specific indices used. Therefore, any fragment f^I will have a weight in G^I of the form:

$$\omega_I(f^I) = \omega_{\text{BEGIN}}(b) \prod_{r \in \text{C}} \omega_{\text{CONT}}(r) \prod_{e \in \text{E}} \omega_{\text{END}}(e)$$

where b is the BEGIN rule, r are CONTINUE rules, and e are END rules in the fragment f^I (see Figure 1(d)). Because ω is assumed to not depend on the specific indices, all f^I which correspond to the same f under π will have the same weight $\omega_I(f)$ in G^I.

In this case, we can define an induced weight

[5] Just half the training set (19916 trees) itself had 1.7 million depth 1 and 2 unbinarized rules compared to the 0.9 million indexed symbols in G^I (after graph packing). Even extracting binarized fragments (depth 1 and 2, with one order of parent annotation) gives us 0.75 million rules, and, practically, we would need fragments of greater depth.

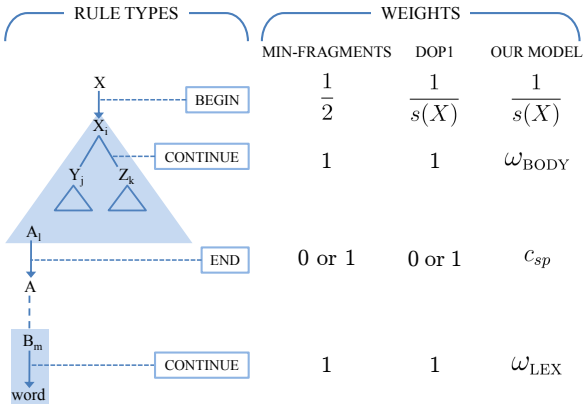

Figure 2: Rules defined for grammar G^I and weight schema for the DOP1 model, the Min-Fragments model (Goodman (2003)) and our model. Here $s(X)$ denotes the total number of fragments rooted at base symbol X.

for fragments f in G by

$$\omega_G(f) = \sum_{f^I \in \pi^{-1}(f)} \omega_I(f^I) = n(f)\omega_I(f)$$
$$= n(f)\omega_{\text{BEGIN}}(b') \prod_{r' \in \text{C}} \omega_{\text{CONT}}(r') \prod_{e' \in \text{E}} \omega_{\text{END}}(e')$$

where now b', r' and e' are non-indexed type abstractions of f's member productions in G^I and $n(f) = |\pi^{-1}(f)|$ is the number of tokens of f in B.

Under the weight function $\omega_G(f)$, any derivation d in G will have weight which obeys

$$\omega_G(d) = \prod_{f \in d} \omega_G(f) = \prod_{f \in d} n(f)\omega_I(f)$$
$$= \sum_{d^I \in d} \omega_I(d^I)$$

and so the posterior $P(d|s)$ of a derivation d for a sentence s will be the same whether computed in G or G^I. Therefore, provided our weighting function on fragments f in G decomposes over the derivational representation of f in G^I, we can equivalently compute the quantities we need for inference (see Section 4) using G^I instead.

3 Parameterization of Implicit Grammars

3.1 Classical DOP1

The original data-oriented parsing model 'DOP1' (Bod, 1993) is a particular instance of the general weighting scheme which decomposes appropriately over the implicit encoding, described in Section 2.3. Figure 2 shows rule weights for DOP1 in the parameter schema we have defined. The END rule weight is 0 or 1 depending on whether A is an intermediate symbol or not.[6] The local fragments in DOP1 were flat (non-binary) so this weight choice simulates that property by not allowing switching between fragments at intermediate symbols.

The original DOP1 model weights a fragment f in G as $\omega_G(f) = n(f)/s(X)$, i.e., the frequency of fragment f divided by the number of fragments rooted at base symbol X. This is simulated by our weight choices (Figure 2) where each fragment f^I in G^I has weight $\omega_I(f^I) = 1/s(X)$ and therefore, $\omega_G(f) = \sum_{f^I \in \pi^{-1}(f)} \omega_I(f^I) = n(f)/s(X)$. Given the weights used for DOP1, the recursive formula for the number of fragments $s(X_i)$ rooted at indexed symbol X_i (and for the CONTINUE rule $X_i \rightarrow Y_j\ Z_k$) is

$$s(X_i) = (1 + s(Y_j))(1 + s(Z_k)), \qquad (1)$$

where $s(Y_j)$ and $s(Z_k)$ are the number of fragments rooted at indexed symbols Y_j and Z_k (non-intermediate) respectively. The number of fragments $s(X)$ rooted at base symbol X is then $s(X) = \sum_{X_i} s(X_i)$.

Implicitly parsing with the full DOP1 model (no sampling of fragments) using the weights in Figure 2 gives a 68% parsing accuracy on the WSJ dev-set.[7] This result indicates that the weight of a fragment should depend on more than just its frequency.

3.2 Better Parameterization

As has been pointed out in the literature, large-fragment grammars can benefit from weights of fragments depending not only on their frequency but also on other properties. For example, Bod (2001) restricts the size and number of words in the frontier of the fragments, and Collins and Duffy (2002) and Goodman (2003) both give larger fragments smaller weights. Our model can incorporate both size and lexical properties. In particular, we set $\omega_{\text{CONT}}(r)$ for each binary CONTINUE rule r to a learned constant ω_{BODY}, and we set the weight for each rule with a POS parent to a

[6] Intermediate symbols are those created during binarization.

[7] For DOP1 experiments, we use no symbol refinement. We annotate with full left binarization history to imitate the flat nature of fragments in DOP1. We use mild coarse-pass pruning (Section 4.1) without which the basic all-fragments chart does not fit in memory. Standard WSJ treebank splits used: sec 2-21 training, 22 dev, 23 test.

Rule score: $r(A \rightarrow B\ C, i, k, j) = \sum_x \sum_y \sum_z O(A_x, i, j)\omega(A_x \rightarrow B_y\ C_z)I(B_y, i, k)I(C_z, k, j)$

Max-Constituent:	$q(A, i, j) = \frac{\sum_x O(A_x,i,j)I(A_x,i,j)}{\sum_r I(root_r,0,n)}$	$t_{max} = \operatorname{argmax}_t \sum_{c \in t} q(c)$
Max-Rule-Sum:	$q(A \rightarrow B\ C, i, k, j) = \frac{r(A \rightarrow B\ C,i,k,j)}{\sum_r I(root_r,0,n)}$	$t_{max} = \operatorname{argmax}_t \sum_{e \in t} q(e)$
Max-Variational:	$q(A \rightarrow B\ C, i, k, j) = \frac{r(A \rightarrow B\ C,i,k,j)}{\sum_x O(A_x,i,j)I(A_x,i,j)}$	$t_{max} = \operatorname{argmax}_t \prod_{e \in t} q(e)$

Figure 3: Inference: Different objectives for parsing with posteriors. A, B, C are base symbols, A_x, B_y, C_z are indexed symbols and i,j,k are between-word indices. Hence, (A_x, i, j) represents a constituent labeled with A_x spanning words i to j. $I(A_x, i, j)$ and $O(A_x, i, j)$ denote the inside and outside scores of this constituent, respectively. For brevity, we write $c \equiv (A, i, j)$ and $e \equiv (A \rightarrow B\ C, i, k, j)$. Also, t_{max} is the highest scoring parse. Adapted from Petrov and Klein (2007).

constant ω_{LEX} (see Figure 2). Fractional values of these parameters allow the weight of a fragment to depend on its size and lexical properties.

Another parameter we introduce is a 'switching-penalty' c_{sp} for the END rules (Figure 2). The DOP1 model uses binary values (0 if symbol is intermediate, 1 otherwise) as the END rule weight, which is equivalent to prohibiting fragment switching at intermediate symbols. We learn a fractional constant a_{sp} that allows (but penalizes) switching between fragments at annotated symbols through the formulation $c_{sp}(X_{intermediate}) = 1 - a_{sp}$ and $c_{sp}(X_{non-intermediate}) = 1 + a_{sp}$. This feature allows fragments to be assigned weights based on the binarization status of their nodes.

With the above weights, the recursive formula for $s(X_i)$, the total weighted number of fragments rooted at indexed symbol X_i, is different from DOP1 (Equation 1). For rule $X_i \rightarrow Y_j\ Z_k$, it is

$$s(X_i) = \omega_{\text{BODY}}.(c_{sp}(Y_j)+s(Y_j))(c_{sp}(Z_k)+s(Z_k)).$$

The formula uses ω_{LEX} in place of ω_{BODY} if r is a lexical rule (Figure 2).

The resulting grammar is primarily parameterized by the training treebank B. However, each setting of the hyperparameters $(\omega_{\text{BODY}}, \omega_{\text{LEX}}, a_{sp})$ defines a different conditional distribution on trees. We choose amongst these distributions by directly optimizing parsing F1 on our development set. Because this objective is not easily differentiated, we simply perform a grid search on the three hyperparameters. The tuned values are $\omega_{\text{BODY}} = 0.35$, $\omega_{\text{LEX}} = 0.25$ and $a_{sp} = 0.018$. For generalization to a larger parameter space, we would of course need to switch to a learning approach that scales more gracefully in the number of tunable hyperparameters.[8]

[8]Note that there has been a long history of DOP estimators. The generative DOP1 model was shown to be inconsistent by Johnson (2002). Later, Zollmann and Sima'an (2005) presented a statistically consistent estimator, with the basic insight of optimizing on a held-out set. Our estimator is not intended to be viewed as a generative model of trees at all, but simply a loss-minimizing conditional distribution within our parametric family.

	dev ($\leq$ 40)		test ($\leq$ 40)		test (all)	
Model	F1	EX	F1	EX	F1	EX
Constituent	**88.4**	33.7	**88.5**	33.0	**87.6**	30.8
Rule-Sum	88.2	**34.6**	88.3	33.8	87.4	31.6
Variational	87.7	34.4	87.7	**33.9**	86.9	**31.6**

Table 1: All-fragments WSJ results (accuracy F1 and exact match EX) for the constituent, rule-sum and variational objectives, using parent annotation and one level of markovization.

4 Efficient Inference

The previously described implicit grammar G^I defines a posterior distribution $P(d^I|s)$ over a sentence s via a large, indexed PCFG. This distribution has the property that, when marginalized, it is equivalent to a posterior distribution $P(d|s)$ over derivations in the correspondingly-weighted all-fragments grammar G. However, even with an explicit representation of G, we would not be able to tractably compute the parse that maximizes $P(t|s) = \sum_{d \in t} P(d|s) = \sum_{d^I \in t} P(d^I|s)$ (Sima'an, 1996). We therefore approximately maximize over trees by computing various existing approximations to $P(t|s)$ (Figure 3). Goodman (1996b), Petrov and Klein (2007), and Matsuzaki et al. (2005) describe the details of constituent, rule-sum and variational objectives respectively. Note that all inference methods depend on the posterior $P(t|s)$ only through marginal expectations of labeled constituent counts and anchored local binary tree counts, which are easily computed from $P(d^I|s)$ and equivalent to those from $P(d|s)$. Therefore, no additional approximations are made in G^I over G.

As shown in Table 1, our model (an all-fragments grammar with the weighting scheme

shown in Figure 2) achieves an accuracy of 88.5% (using simple parent annotation) which is 4-5% (absolute) better than the recent TSG work (Zuidema, 2007; Cohn et al., 2009; Post and Gildea, 2009) and also approaches state-of-the-art refinement-based parsers (e.g., Charniak and Johnson (2005), Petrov and Klein (2007)).[9]

4.1 Coarse-to-Fine Inference

Coarse-to-fine inference is a well-established way to accelerate parsing. Charniak et al. (2006) introduced multi-level coarse-to-fine parsing, which extends the basic pre-parsing idea by adding more rounds of pruning. Their pruning grammars were coarse versions of the raw treebank grammar. Petrov and Klein (2007) propose a multi-stage coarse-to-fine method in which they construct a sequence of increasingly refined grammars, reparsing with each refinement. In particular, in their approach, which we adopt here, coarse-to-fine pruning is used to quickly compute approximate marginals, which are then used to prune subsequent search. The key challenge in coarse-to-fine inference is the construction of coarse models which are much smaller than the target model, yet whose posterior marginals are close enough to prune with safely.

Our grammar G^I has a very large number of indexed symbols, so we use a coarse pass to prune away their unindexed abstractions. The simple, intuitive, and effective choice for such a coarse grammar G^C is a minimal PCFG grammar composed of the base treebank symbols X and the minimal depth-1 binary rules $X \rightarrow Y\ Z$ (and with the same level of annotation as in the full grammar). If a particular base symbol X is pruned by the coarse pass for a particular span (i, j) (i.e., the posterior marginal $P(X, i, j|s)$ is less than a certain threshold), then in the full grammar G^I, we do not allow building any indexed symbol X_l of type X for that span. Hence, the projection map for the coarse-to-fine model is π^C : $X_l\ (indexed\ symbol) \rightarrow X\ (base\ symbol)$.

We achieve a substantial improvement in speed and memory-usage from the coarse-pass pruning. Speed increases by a factor of 40 and memory-usage decreases by a factor of 10 when we go from no pruning to pruning with a -6.2 log posterior threshold.[10] Figure 4 depicts the variation in parsing accuracies in response to the amount of pruning done by the coarse-pass. Higher posterior pruning thresholds induce more aggressive pruning. Here, we observe an effect seen in previous work (Charniak et al. (1998), Petrov and Klein (2007), Petrov et al. (2008)), that a certain amount of pruning helps accuracy, perhaps by promoting agreement between the coarse and full grammars (model intersection). However, these 'fortuitous' search errors give only a small improvement and the peak accuracy is almost equal to the parsing accuracy without any pruning (as seen in Figure 5).[11] This outcome suggests that the coarse-pass pruning is critical for tractability but not for performance.

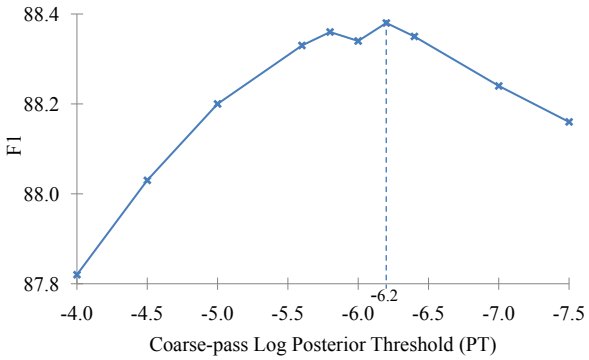

Figure 4: Effect of coarse-pass pruning on parsing accuracy (for WSJ dev-set, $\leq$ 40 words). Pruning increases to the left as log posterior threshold (PT) increases.

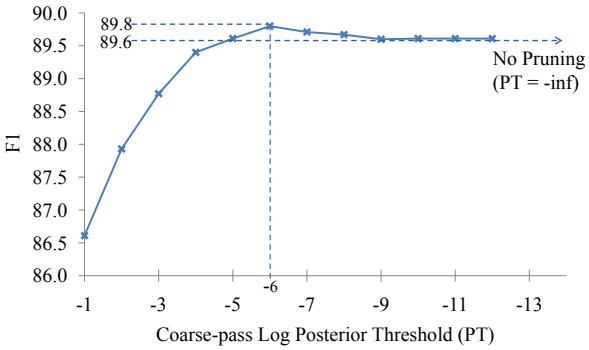

Figure 5: Effect of coarse-pass pruning on parsing accuracy (WSJ, training $\leq$ 20 words, tested on dev-set $\leq$ 20 words). This graph shows that the fortuitous improvement due to pruning is very small and that the peak accuracy is almost equal to the accuracy without pruning (the dotted line).

[9] All our experiments use the constituent objective except when we report results for max-rule-sum and max-variational parsing (where we use the parameters tuned for max-constituent, therefore they unsurprisingly do not perform as well as max-constituent). Evaluations use EVALB, see `http://nlp.cs.nyu.edu/evalb/`.

[10] Unpruned experiments could not be run for 40-word test sentences even with 50GB of memory, therefore we calculated the improvement factors using a smaller experiment with full training and sixty 30-word test sentences.

[11] To run experiments without pruning, we used training and dev sentences of length $\leq$ 20 for the graph in Figure 5.

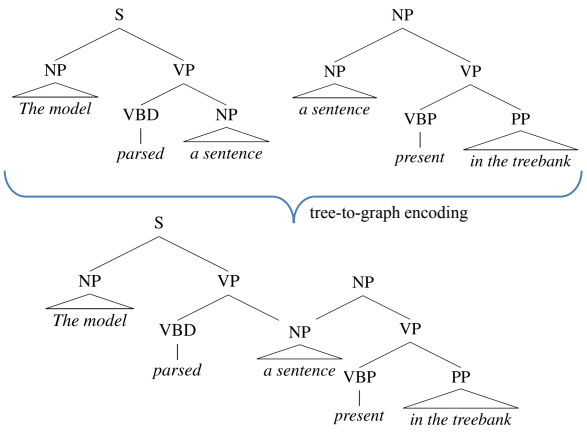

Figure 6: Collapsing the duplicate training subtrees converts them to a graph and reduces the number of indexed symbols significantly.

Parsing Model	No. of Indexed Symbols
Word-level Trees	1,900,056
Word-level Graph	903,056
Character-level Trees	12,280,848
Character-level Graph	1,109,399

Table 2: Number of indexed symbols for word-level and character-level parsing and their graph versions (for all-fragments grammar with parent annotation and one level of markovization).

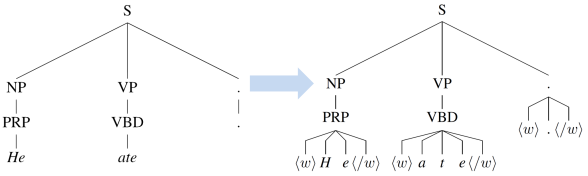

Figure 7: Character-level parsing: treating the sentence as a string of characters instead of words.

4.2 Packed Graph Encoding

The implicit all-fragments approach (Section 2.2) avoids explicit extraction of all rule fragments. However, the number of indexed symbols in our implicit grammar G^I is still large, because every node in each training tree (i.e., every symbol token) has a unique indexed symbol. We have around 1.9 million indexed symbol tokens in the word-level parsing model (this number increases further to almost 12.3 million when we parse character strings in Section 5.1). This large symbol space makes parsing slow and memory-intensive.

We reduce the number of symbols in our implicit grammar G^I by applying a compact, packed graph encoding to the treebank training trees. We collapse the duplicate *subtrees* (fragments that bottom out in terminals) over all training trees. This keeps the grammar unchanged because in an tree-substitution grammar, a node is defined (identified) by the subtree below it. We maintain a hashmap on the subtrees which allows us to easily discover the duplicates and bin them together. The collapsing converts all the training trees in the treebank to a graph with multiple parents for some nodes as shown in Figure 6. This technique reduces the number of indexed symbols significantly as shown in Table 2 (1.9 million goes down to 0.9 million, reduction by a factor of 2.1). This reduction increases parsing speed by a factor of 1.4 (and by a factor of 20 for character-level parsing, see Section 5.1) and reduces memory usage to under 4GB.

We store the duplicate-subtree counts for each indexed symbol of the collapsed graph (using a hashmap). When calculating the number of fragments $s(X_i)$ parented by an indexed symbol X_i (see Section 3.2), and when calculating the inside and outside scores during inference, we account for the collapsed subtree tokens by expanding the counts and scores using the corresponding multiplicities. Therefore, we achieve the compaction with negligible overhead in computation.

5 Improved Treebank Representations

5.1 Character-Level Parsing

The all-fragments approach to parsing has the added advantage that parsing below the word level requires no special treatment, i.e., we do not need an explicit lexicon when sentences are considered as strings of characters rather than words.

Unknown words in test sentences (unseen in training) are a major issue in parsing systems for which we need to train a complex lexicon, with various unknown classes or suffix tries. Smoothing factors need to be accounted for and tuned. With our implicit approach, we can avoid training a lexicon by building up the parse tree from characters instead of words. As depicted in Figure 7, each word in the training trees is split into its corresponding characters with start and stop boundary tags (and then binarized in a standard right-branching style). A test sentence's words are split up similarly and the test-parse is built from training fragments using the same model and inference procedure as defined for word-level parsing (see Sections 2, 3 and 4). The lexical items (alphabets, digits etc.) are now all known, so unlike word-level parsing, no sophisticated lexicon is needed.

We choose a slightly richer weighting scheme

Model	dev ($\leq$ 40) F1	dev ($\leq$ 40) EX	test ($\leq$ 40) F1	test ($\leq$ 40) EX	test (all) F1	test (all) EX
Constituent	**88.2**	33.6	**88.0**	31.9	**87.1**	29.8
Rule-Sum	88.0	33.9	87.8	**33.1**	87.0	**30.9**
Variational	87.6	**34.4**	87.2	32.3	86.4	30.2

Table 3: All-fragments WSJ results for the character-level parsing model, using parent annotation and one level of markovization.

Parsing Model	F1
No Refinement (P=0, H=0)*	71.3
Basic Refinement (P=1, H=1)*	80.0
All-Fragments + No Refinement (P=0, H=0)	85.7
All-Fragments + Basic Refinement (P=1, H=1)	88.4

Table 4: F1 for a basic PCFG, and incorporation of basic refinement, all-fragments and both, for WSJ dev-set ($\leq$ 40 words). $P = 1$ means parent annotation of all non-terminals, including the preterminal tags. $H = 1$ means one level of markovization. *Results from Klein and Manning (2003).

for this representation by extending the two-weight schema for CONTINUE rules (ω_{LEX} and ω_{BODY}) to a three-weight one: ω_{LEX}, ω_{WORD}, and ω_{SENT} for CONTINUE rules in the lexical layer, in the portion of the parse that builds words from characters, and in the portion of the parse that builds the sentence from words, respectively. The tuned values are $\omega_{\text{SENT}} = 0.35$, $\omega_{\text{WORD}} = 0.15$, $\omega_{\text{LEX}} = 0.95$ and $a_{sp} = 0$. The character-level model achieves a parsing accuracy of 88.0% (see Table 3), *despite lacking an explicit lexicon*.[12]

Character-level parsing expands the training trees (see Figure 7) and the already large indexed symbol space size explodes (1.9 million increases to 12.3 million, see Table 2). Fortunately, this is where the packed graph encoding (Section 4.2) is most effective because duplication of character strings is high (e.g., suffixes). The packing shrinks the symbol space size from 12.3 million to 1.1 million, a reduction by a factor of 11. This reduction increases parsing speed by almost a factor of 20 and brings down memory-usage to under 8GB.[13]

5.2 Basic Refinement: Parent Annotation and Horizontal Markovization

In a pure all-fragments approach, compositions of units which would have been independent in a basic PCFG are given joint scores, allowing the representation of certain non-local phenomena, such as lexical selection or agreement, which in fully local models require rich state-splitting or lexicalization. However, at substitution sites, the coarseness of raw unrefined treebank symbols still creates unrealistic factorization assumptions. A standard solution is symbol refinement; Johnson (1998) presents the particularly simple case of parent annotation, in which each node is marked with its parent in the underlying treebank. It is reasonable to hope that the gains from using large fragments and the gains from symbol refinement will be complementary. Indeed, previous work has shown or suggested this complementarity. Sima'an (2000) showed modest gains from enriching structural relations with semi-lexical (prehead) information. Charniak and Johnson (2005) showed accuracy improvements from composed local tree features on top of a lexicalized base parser. Zuidema (2007) showed a slight improvement in parsing accuracy when enough fragments were added to learn enrichments beyond manual refinements. Our work reinforces this intuition by demonstrating how complementary they are in our model ($\sim$20% error reduction on adding refinement to an all-fragments grammar, as shown in the last two rows of Table 4).

Table 4 shows results for a basic PCFG, and its augmentation with either basic refinement (parent annotation and one level of markovization), with all-fragments rules (as in previous sections), or both. The basic incorporation of large fragments alone does not yield particularly strong performance, nor does basic symbol refinement. However, the two approaches are quite additive in our model and combine to give nearly state-of-the-art parsing accuracies.

5.3 Additional Deterministic Refinement

Basic symbol refinement (parent annotation), in combination with all-fragments, gives test-set accuracies of 88.5% ($\leq$ 40 words) and 87.6% (all), shown as the Basic Refinement model in Table 5. Klein and Manning (2003) describe a broad set of simple, deterministic symbol refinements beyond parent annotation. We included ten of their simplest annotation features, namely: UNARY-DT, UNARY-RB, SPLIT-IN, SPLIT-AUX, SPLIT-CC, SPLIT-%, GAPPED-S, POSS-NP, BASE-NP and DOMINATES-V. None of these annotation schemes use any head information. This additional annotation (see Ad-

[12]Note that the word-level model yields a higher accuracy of 88.5%, but uses 50 complex unknown word categories based on lexical, morphological and position features (Petrov et al., 2006). Cohn et al. (2009) also uses this lexicon.

[13]Full char-level experiments (w/o packed graph encoding) could not be run even with 50GB of memory. We calculate the improvement factors using a smaller experiment with 70% training and fifty 20-word test sentences.

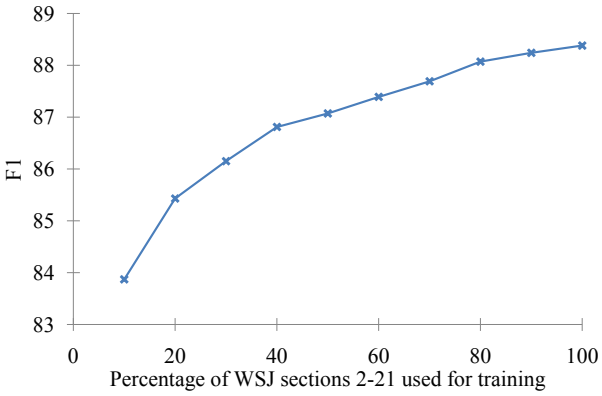

Figure 8: Parsing accuracy F1 on the WSJ dev-set ($\leq$ 40 words) increases with increasing percentage of training data.

Parsing Model	test ($\leq$ 40) F1	test ($\leq$ 40) EX	test (all) F1	test (all) EX
FRAGMENT-BASED PARSERS				
Zuidema (2007)	–	–	83.8*	26.9*
Cohn et al. (2009)	–	–	84.0	–
Post and Gildea (2009)	82.6	–	–	–
THIS PAPER				
All-Fragments				
+ Basic Refinement	88.5	33.0	87.6	30.8
+ Additional Refinement	88.7	33.8	88.1	31.7
REFINEMENT-BASED PARSERS				
Collins (1999)	88.6	–	88.2	–
Petrov and Klein (2007)	90.6	39.1	90.1	37.1

Table 5: Our WSJ test set parsing accuracies, compared to recent fragment-based parsers and top refinement-based parsers. Basic Refinement is our all-fragments grammar with parent annotation. Additional Refinement adds deterministic refinement of Klein and Manning (2003) (Section 5.3). *Results on the dev-set ($\leq$ 100).

ditional Refinement, Table 5) improves the test-set accuracies to 88.7% ($\leq$ 40 words) and 88.1% (all), which is equal to a strong lexicalized parser (Collins, 1999), even though our model does not use lexicalization or latent symbol-split induction.

6 Other Results

6.1 Parsing Speed and Memory Usage

The word-level parsing model using the whole training set (39832 trees, all-fragments) takes approximately 3 hours on the WSJ test set (2245 trees of $\leq$40 words), which is equivalent to roughly 5 seconds of parsing time per sentence; and runs in under 4GB of memory. The character-level version takes about twice the time and memory. This novel tractability of an all-fragments grammar is achieved using both coarse-pass pruning and packed graph encoding. Micro-optimization may further improve speed and memory usage.

6.2 Training Size Variation

Figure 8 shows how WSJ parsing accuracy increases with increasing amount of training data (i.e., percentage of WSJ sections 2-21). Even if we train on only 10% of the WSJ training data (3983 sentences), we still achieve a reasonable parsing accuracy of nearly 84% (on the development set, $\leq$ 40 words), which is comparable to the full-system results obtained by Zuidema (2007), Cohn et al. (2009) and Post and Gildea (2009).

6.3 Other Language Treebanks

On the French and German treebanks (using the standard dataset splits mentioned in Petrov and Klein (2008)), our simple all-fragments parser achieves accuracies in the range of top refinement-based parsers, even though the model parameters were tuned out of domain on WSJ. For German, our parser achieves an F1 of 79.8% compared to 81.5% by the state-of-the-art and substantially more complex Petrov and Klein (2008) work. For French, our approach yields an F1 of 78.0% vs. 80.1% by Petrov and Klein (2008).[14]

7 Conclusion

Our approach of using all fragments, in combination with basic symbol refinement, and even without an explicit lexicon, achieves results in the range of state-of-the-art parsers on full scale treebanks, across multiple languages. The main takeaway is that we can achieve such results in a very knowledge-light way with (1) no latent-variable training, (2) no sampling, (3) no smoothing beyond the existence of small fragments, and (4) no explicit unknown word model at all. While these methods offer a simple new way to construct an accurate parser, we believe that this general approach can also extend to other large-fragment tasks, such as machine translation.

Acknowledgments

This project is funded in part by BBN under DARPA contract HR0011-06-C-0022 and the NSF under grant 0643742.

[14] All results on the test set ($\leq$ 40 words).

References

Rens Bod. 1993. Using an Annotated Corpus as a Stochastic Grammar. In *Proceedings of EACL*.

Rens Bod. 2001. What is the Minimal Set of Fragments that Achieves Maximum Parse Accuracy? In *Proceedings of ACL*.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of ACL*.

Eugene Charniak, Sharon Goldwater, and Mark Johnson. 1998. Edge-Based Best-First Chart Parsing. In *Proceedings of the 6th Workshop on Very Large Corpora*.

Eugene Charniak, Mark Johnson, et al. 2006. Multilevel Coarse-to-fine PCFG Parsing. In *Proceedings of HLT-NAACL*.

Eugene Charniak. 2000. A Maximum-Entropy-Inspired Parser. In *Proceedings of NAACL*.

David Chiang. 2003. Statistical parsing with an automatically-extracted tree adjoining grammar. In *Data-Oriented Parsing*.

David Chiang. 2005. A Hierarchical Phrase-Based Model for Statistical Machine Translation. In *Proceedings of ACL*.

Trevor Cohn, Sharon Goldwater, and Phil Blunsom. 2009. Inducing Compact but Accurate Tree-Substitution Grammars. In *Proceedings of NAACL*.

Michael Collins and Nigel Duffy. 2002. New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete Structures, and the Voted Perceptron. In *Proceedings of ACL*.

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

Steve Deneefe and Kevin Knight. 2009. Synchronous Tree Adjoining Machine Translation. In *Proceedings of EMNLP*.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule? In *Proceedings of HLT-NAACL*.

Joshua Goodman. 1996a. Efficient Algorithms for Parsing the DOP Model. In *Proceedings of EMNLP*.

Joshua Goodman. 1996b. Parsing Algorithms and Metrics. In *Proceedings of ACL*.

Joshua Goodman. 2003. Efficient parsing of DOP with PCFG-reductions. In *Bod R, Scha R, Sima'an K (eds.) Data-Oriented Parsing. University of Chicago Press, Chicago, IL*.

James Henderson. 2004. Discriminative Training of a Neural Network Statistical Parser. In *Proceedings of ACL*.

Mark Johnson. 1998. PCFG Models of Linguistic Tree Representations. *Computational Linguistics*, 24:613–632.

Mark Johnson. 2002. The DOP Estimation Method Is Biased and Inconsistent. In *Computational Linguistics 28(1)*.

Dan Klein and Christopher Manning. 2003. Accurate Unlexicalized Parsing. In *Proceedings of ACL*.

Philipp Koehn, Franz Och, and Daniel Marcu. 2003. Statistical Phrase-Based Translation. In *Proceedings of HLT-NAACL*.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic CFG with latent annotations. In *Proceedings of ACL*.

Slav Petrov and Dan Klein. 2007. Improved Inference for Unlexicalized Parsing. In *Proceedings of NAACL-HLT*.

Slav Petrov and Dan Klein. 2008. Sparse Multi-Scale Grammars for Discriminative Latent Variable Parsing. In *Proceedings of EMNLP*.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning Accurate, Compact, and Interpretable Tree Annotation. In *Proceedings of COLING-ACL*.

Slav Petrov, Aria Haghighi, and Dan Klein. 2008. Coarse-to-Fine Syntactic Machine Translation using Language Projections. In *Proceedings of EMNLP*.

Matt Post and Daniel Gildea. 2009. Bayesian Learning of a Tree Substitution Grammar. In *Proceedings of ACL-IJCNLP*.

Philip Resnik. 1992. Probabilistic Tree-Adjoining Grammar as a Framework for Statistical Natural Language Processing. In *Proceedings of COLING*.

Remko Scha. 1990. Taaltheorie en taaltechnologie; competence en performance. In *R. de Kort and G.L.J. Leerdam (eds.): Computertoepassingen in de Neerlandistiek*.

Khalil Sima'an. 1996. Computational Complexity of Probabilistic Disambiguation by means of Tree-Grammars. In *Proceedings of COLING*.

Khalil Sima'an. 2000. Tree-gram Parsing: Lexical Dependencies and Structural Relations. In *Proceedings of ACL*.

Andreas Zollmann and Khalil Sima'an. 2005. A Consistent and Efficient Estimator for Data-Oriented Parsing. *Journal of Automata, Languages and Combinatorics (JALC)*, 10(2/3):367–388.

Willem Zuidema. 2007. Parsimonious Data-Oriented Parsing. In *Proceedings of EMNLP-CoNLL*.