The Surprising Variance in Shortest-Derivation Parsing

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

Abstract

We investigate full-scale shortest-derivation parsing (SDP), wherein the parser selects an analysis built from the fewest number of training fragments. Shortest derivation parsing exhibits an unusual range of behaviors. At one extreme, in the fully unpruned case, it is neither fast nor accurate. At the other extreme, when pruned with a coarse unlexicalized PCFG, the shortest derivation criterion becomes both fast and surprisingly effective, rivaling more complex weighted-fragment approaches. Our analysis includes an investigation of tie-breaking and associated dynamic programs. At its best, our parser achieves an accuracy of 87% F1 on the English WSJ task with minimal annotation, and 90% F1 with richer annotation.

1 Introduction

One guiding intuition in parsing, and data-driven NLP more generally, is that, all else equal, it is advantageous to memorize large fragments of training examples. Taken to the extreme, this intuition suggests *shortest derivation parsing* (SDP), wherein a test sentence is analyzed in a way which uses as few training fragments as possible (Bod, 2000; Goodman, 2003). SDP certainly has appealing properties: it is simple and parameter free – there need not even be an explicit lexicon. However, SDP may be too simple to be competitive.

In this paper, we consider SDP in both its pure form and with several direct modifications, finding a range of behaviors. In its pure form, with no pruning or approximation, SDP is neither fast nor accurate, achieving less than 70% F1 on the English WSJ task. Moreover, basic tie-breaking variants and lexical augmentation are insufficient to achieve competitive accuracies.[1] On the other hand, SDP is dramatically improved in both speed and accuracy when a simple, unlexicalized PCFG is used for coarse-to-fine pruning (and tie-breaking). On the English WSJ, the coarse PCFG and the fine SDP together achieve 87% F1 with basic treebank annotation (see Table 2) and up to 90% F1 with richer treebank annotation (see Table 4).

The main contribution of this work is to analyze the behavior of shortest derivation parsing, showing both when it fails and when it succeeds. Our final parser, which combines a simple PCFG coarse pass with an otherwise pure SPD fine pass, can be quite accurate while being straightforward to implement.

2 Implicit Grammar for SDP

The all-fragments grammar (AFG) for a (binarized) treebank is formally the tree-substitution grammar (TSG) (Resnik, 1992; Bod, 1993) that consists of all fragments (elementary trees) of all training trees in the treebank, with some weighting on each fragment. AFGs are too large to fully extract explicitly; researchers therefore either work with a tractable subset of the fragments (Sima'an, 2000; Bod, 2001; Post and Gildea, 2009; Cohn and Blunsom, 2010) or use a PCFG reduction like that of Goodman (1996a), in which each treebank node token X_i is given its own unique grammar symbol.

We follow Bansal and Klein (2010) in choosing the latter, both to permit comparison to their results and because SDP is easily phrased as a PCFG reduction. Bansal and Klein (2010) use a carefully pa-

[1] Bod (2000) presented another SDP parser, but with a sampled subset of the training fragments.

rameterized weighting of the substructures in their grammar in an effort to extend the original DOP1 model (Bod, 1993; Goodman, 1996a). However, for SDP, the grammar is even simpler (Goodman, 2003). In principle, the implicit SDP grammar needs just two rule schemas: CONTINUE ($X_p \rightarrow Y_q\, Z_r$) and SWITCH ($X_p \rightarrow X_q$), with additive costs 0 and 1, respectively. CONTINUE rules walk along training trees, while SWITCH rules change between trees for a unit cost.[2] Assuming that the SWITCH rules are in practice broken down into BEGIN and END sub-rules as in Bansal and Klein (2010), the grammar is linear in the size of the treebank.[3] Note that no lexicon is needed in this grammar: lexical switches are like any other.

A derivation in our grammar has weight (cost) w where w is the number of switches (or the number of training fragments minus one) used to build the derivation (see Figure 1). The Viterbi dynamic program for finding the shortest derivation is quite simple: it requires CKY to store only byte-valued switch-counts $s(X_p, i, j)$ (i.e., the number of switches) for each chart item and compute the derivation with the least switch-count. Specifically, in the dynamic program, if we use a SWITCH rule $X_p \rightarrow X_q$, then we update

$$s(X_p, i, j) := s(X_q, i, j) + 1.$$

If we use a continue rule $X_p \rightarrow Y_q\, Z_r$, then the update is

$$s(X_p, i, j) := s(Y_q, i, k) + s(Z_r, k, j),$$

where k is a split point in the chart. Using this dynamic program, we compute the exact shortest derivation parse in the full all-fragments grammar (which is reduced to a PCFG with 2 rules schemas as described above).

3 Basic SDP: Inaccurate and Slow

SDP in its most basic form is appealingly simple, but has two serious issues: it is both slow and inaccurate. Because there are millions of grammar symbols, exact SDP parsing takes more than 45 seconds per sentence in our implementation (in addition to being highly memory-intensive). Many methods exist for speeding up parsing through approximation, but basic SDP is too inaccurate to merit them. When implemented as described in Section 2, SDP achieves only 66% F1 on the WSJ task (dev set, $\leq$ 40 words).

[2]This grammar is a very minor variant of the reduction of SDP suggested by Goodman (2003).

[3]For a compact WSJ training set with graph packing (see Bansal and Klein (2010)) and one level of parent annotation and markovization, our grammar has 0.9 million indexed symbols compared to 7.5 million unbinarized (and 0.75 million binarized) explicitly-extracted fragments of just depth 1 and 2.

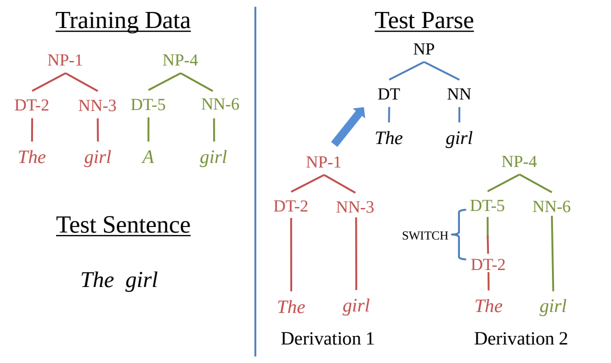

Figure 1: SDP - the best parse corresponds to the shortest derivation (fewest switches).

Why does SDP perform so poorly? One reason for low accuracy may be that there are many shortest derivations, i.e. derivations that are all built with the fewest number of fragments, and that tie breaking could be at fault. To investigate this, we tried various methods for tie-breaking: FIRST/LAST (procedurally break ties), UNIFORM (sample derivations equally), FREQ (use the frequency of local rules). However, none of these methods help much, giving results within a percentage of F1. In fact, even *oracle* tie-breaking, where ties are broken to favor the number of gold constituents in the derivation achieves only 80% F1, indicating that correct derivations are often not the shortest ones. Another reason for the poor performance of SDP may be that the parameter-free treatment of the lexical layer is particularly pathological. Indeed, this hypothesis is partially verified by the result that using a lexicon (similar to that in Petrov et al. (2006)) at the terminal layer brings the uniform tie-breaking result up to 80% F1. However, combining a lexicon with oracle tie-breaking yields only 81.8% F1.

These results at first seem quite discouraging, but we will show that they can be easily improved with information from even a simple PCFG.

4 Improvements from a Coarse PCFG

The additional information that makes shortest derivation parsing work comes from a coarse unlexicalized PCFG. In the standard way, our PCFG consists of the local (depth-1) rules $X \rightarrow Y\ Z$ with probability $P(Y\ \ Z|X)$ computed using the count of the rule and the count of the nonterminal X in the given treebank (no smoothing was used). Our coarse grammar uses a lexicon with unknown word classes, similar to that in Petrov et al. (2006). When taken from a binarized treebank with one level of parent annotation (Johnson, 1998) and horizontal markovization, the PCFG is quite small, with around 3500 symbols and 25000 rules; it achieves an accuracy of 84% on its own (see Table 2), so the PCFG on its own is better than the basic SDP, but still relatively weak.

When filtered by a coarse PCFG pass, however, SDP becomes both fast and accurate, even for the basic, lexicon-free SDP formulation. Summed marginals (posteriors) are computed in the coarse PCFG and used for pruning and tie-breaking in the SDP chart, as described next. Pruning works in the standard coarse-to-fine (CTF) way (see Charniak et al. (2006)). If a particular base symbol X is pruned by the PCFG 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 SDP pass we do not allow building any indexed symbol X_l of type X for span (i, j). In all our pruning-based experiments, we use a log posterior threshold of -3.8, tuned on the WSJ development set.

We also use the PCFG coarse pass for tie-breaking. During Viterbi shortest-derivation parsing (after coarse-pruning), if two derivations have the same cost (i.e., the number of switches), then we break the tie between them by choosing the derivation which has a higher sum of coarse posteriors (i.e., the sum of the coarse PCFG chart-cell posteriors $P(X, i, j|s)$ used to build the derivation).[4] The coarse PCFG has an extremely beneficial interaction with the fine all-fragments SDP grammar, wherein the accuracy of the combined grammars is significantly higher than either individually (see Table 2). In addition, the speed of parsing and memory-requirements improve by more than an order of magnitude over the exact SDP pass alone.

	dev ($\leq$ 40)		test ($\leq$ 40)	
Model	F1	EX	F1	EX
B&K2010 pruned	88.4	33.7	88.5	33.0
B&K2010 unpruned	87.9	32.4	88.1	31.9

Table 1: Accuracy (F1) and exact match (EX) for Bansal and Klein (2010). The pruned row shows their original results with coarse-to-fine pruning. The unpruned row shows new results for an unpruned version of their parser; these accuracies are very similar to their pruned counterparts.

It is perhaps surprising that coarse-pass pruning improves accuracy by such a large amount for SDP. Indeed, given that past all-fragments work has used a coarse pass for speed, and that we are the first (to our knowledge) to actually parse at scale with an implicit grammar *without* such a coarse pass, it is a worry that previous results could be crucially dependent on fortuitous coarse-pass pruning. To check one such result, we ran the full, weighted AFG construction of Bansal and Klein (2010) without any pruning (using the maximum recall objective as they did). Their results hold up without pruning: the results of the unpruned version are only around 0.5% less (in parsing F1) than the results achieved with pruning (see Table 1). However, in the case of our shortest-derivation parser, the coarse-pass is essential for high accuracies (and for speed and memory, as always).

5 Results

We have seen that basic, unpruned SDP is both slow and inaccurate, but improves greatly when complemented by a coarse PCFG pass; these results are shown in Table 2. Shortest derivation parsing with a PCFG coarse-pass (PCFG+SDP) achieves an accuracy of nearly 87% F1 (on the WSJ test set, $\leq$ 40 word sentences), which is significantly higher than the accuracy of the PCFG or SDP alone.[5] When the coarse PCFG is combined with basic SDP, the majority of the improvement comes from pruning with the coarse-posteriors; tie-breaking with coarse-posteriors contributes around 0.5% F1 over pruning.

[4]This is similar to the maximum recall objective for approximate inference (Goodman, 1996b). The product of posteriors also works equally well.

[5]PCFG+SDP accuracies are around 3% higher in F1 and 10% higher in EX than the PCFG-only accuracies.

	dev (≤ 40)		test (≤ 40)		test (all)	
Model	F1	EX	F1	EX	F1	EX
SDP	66.2	18.0	66.9	18.4	64.9	17.3
PCFG	83.8	20.0	84.0	21.6	83.2	20.1
PCFG+SDP	86.4	30.6	86.9	31.5	86.0	29.4

Table 2: Our primary results on the WSJ task. SDP is the basic unpruned shortest derivation parser. PCFG results are with one level of parent annotation and horizontal markovization. PCFG+SDP incorporates the coarse PCFG posteriors into SDP. See end of Section 5 for a comparison to other parsing approaches.

Figure 2 shows the number of fragments for shortest derivation parsing (averaged for each sentence length). Note that the number of fragments is of course greater for the combined PCFG+SDP model than the exact basic SDP model (which is guaranteed to be minimal). This result provides some analysis of how coarse-pruning helps SDP: it illustrates that the coarse-pass filters out certain short but inaccurate derivations (that the minimal SDP on its own is forced to choose) to improve performance.

Figure 3 shows the parsing accuracy of the PCFG+SDP model for various pruning thresholds in coarse-to-fine pruning. Note how this is different from the standard coarse-pass pruning graphs (see Charniak et al. (1998), Petrov and Klein (2007), Bansal and Klein (2010)) where only a small improvement is achieved from pruning. In contrast, coarse-pass pruning provides large accuracy benefits here, perhaps because of the unusual complementarity of the two grammars (typical coarse passes are designed to be as similar as possible to their fine counterparts, even explicitly so in Petrov and Klein (2007)).

Our PCFG+SDP parser is more accurate than recent sampling-based TSG's (Post and Gildea, 2009; Cohn and Blunsom, 2010), who achieve 83-85% F1, and it is competitive with more complex weighted-fragment approaches.[6] See Bansal and Klein (2010) for a more thorough comparison to other parsing work. In addition to being accurate, the PCFG+SDP parser is simple and fast, requiring negligible training and tuning. It takes 2 sec/sentence, less than 2 GB of memory and is written in less than 2000 lines of Java code, including I/O.[7]

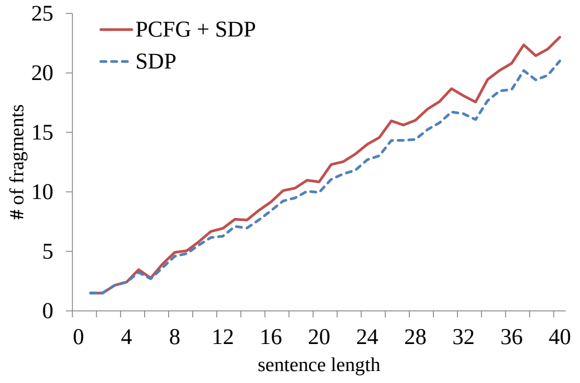

Figure 2: The average number of fragments in shortest derivation parses, computed using the basic version (SDP) and the pruned version (PCFG+SDP), for WSJ dev-set (≤ 40 words).

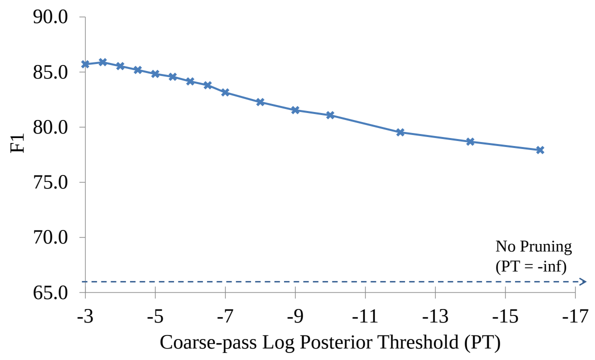

Figure 3: Parsing accuracy for various coarse-pass pruning thresholds (on WSJ dev-set ≤ 40 words). A larger threshold means more pruning. These are results without the coarse-posterior tie-breaking to illustrate the sole effect of pruning.

5.1 Other Treebanks

One nice property of the parameter-free, all-fragments SDP approach is that we can easily transfer it to any new domain with a treebank, or any new annotation of an existing treebank. Table 3 shows domain adaptation performance by the results for training and testing on the Brown and German datasets.[8] On Brown, we perform better than the relatively complex lexicalized Model 1 of Collins (1999). For German, our parser outperforms Dubey (2005) and we are not far behind latent-variable parsers, for which parsing is substantially

[6]Bansal and Klein (2010) achieve around 1.0% higher F1 than our results without a lexicon (character-level parsing) and 1.5% higher F1 with a lexicon.

[7]These statistics can be further improved with standard parsing micro-optimization.

[8]See Gildea (2001) and Petrov and Klein (2007) for the exact experimental setup that we followed here.

	test ($\leq$ 40)		test (all)	
Model	F1	EX	F1	EX
BROWN				
Gildea (2001)	84.1	–	–	–
This Paper (PCFG+SDP)	84.7	34.6	83.1	32.6
GERMAN				
Dubey (2005)	76.3	–	–	–
Petrov and Klein (2007)	80.8	40.8	80.1	39.1
This Paper (PCFG+SDP)	78.1	39.3	77.1	38.2

Table 3: Results for training and testing on the Brown and German treebanks. Gildea (2001) uses the lexicalized Collins' Model 1 (Collins, 1999).

	test ($\leq$ 40)		test (all)	
Annotation	F1	EX	F1	EX
STAN-ANNOTATION	88.1	34.3	87.4	32.2
BERK-ANNOTATION	90.0	38.9	89.5	36.8

Table 4: Results with richer WSJ-annotations from Stanford and Berkeley parsers.

more complex.

5.2 Treebank Annotations

PCFG+SDP achieves 87% F1 on the English WSJ task using basic annotation only (i.e., one level of parent annotation and horizontal markovization). Table 4 shows that by pre-transforming the WSJ treebank with richer annotation from previous work, we can obtain state-of-the-art accuracies of up to 90% F1 with no change to our simple parser. In STAN-ANNOTATION, we annotate the treebank symbols with annotations from the Stanford parser (Klein and Manning, 2003). In BERK-ANNOTATION, we annotate with the splits learned via hard-EM and 5 split-merge rounds of the Berkeley parser (Petrov et al., 2006).

6 Conclusion

Our investigation of shortest-derivation parsing showed that, in the exact case, SDP performs poorly. When pruned (and, to a much lesser extent, tie-broken) by a coarse PCFG, however, it is competitive with a range of other, more complex techniques. An advantage of this approach is that the fine SDP pass is actually quite simple compared to typical fine passes, while still retaining enough complementarity to the coarse PCFG to increase final accuracies. One aspect of our findings that may apply more broadly is the caution that coarse-to-fine methods may sometimes be more critical to end system quality than generally thought.

Acknowledgments

We would like to thank Adam Pauls, Slav Petrov and the anonymous reviewers for their helpful suggestions. This research is supported by BBN under DARPA contract HR0011-06-C-0022 and by the Office of Naval Research under MURI Grant No. N000140911081.

References

Mohit Bansal and Dan Klein. 2010. Simple, Accurate Parsing with an All-Fragments Grammar. In *Proceedings of ACL.*

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

Rens Bod. 2000. Parsing with the Shortest Derivation. In *Proceedings of COLING.*

Rens Bod. 2001. What is the Minimal Set of Fragments that Achieves Maximum Parse Accuracy? 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. Multi-level Coarse-to-fine PCFG Parsing. In *Proceedings of HLT-NAACL.*

Trevor Cohn and Phil Blunsom. 2010. Blocked Inference in Bayesian Tree Substitution Grammars. In *Proceedings of NAACL.*

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

A. Dubey. 2005. What to do when lexicalization fails: parsing German with suffix analysis and smoothing. In *ACL '05.*

Daniel Gildea. 2001. Corpus variation and parser performance. In *Proceedings of EMNLP.*

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.*

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

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

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

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

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*.

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