Learning Grammars for Different Parsing Tasks by Partition Search

Anja Belz
ITRI
University of Brighton
Lewes Road
Brighton BN2 4GJ, UK
Anja.Belz@itri.brighton.ac.uk

Abstract

This paper describes a comparative application of Grammar Learning by Partition Search to four different learning tasks: deep parsing, NP identification, flat phrase chunking and NP chunking. In the experiments, base grammars were extracted from a treebank corpus. From this starting point, new grammars optimised for the different parsing tasks were learnt by Partition Search. No lexical information was used. In half of the experiments, local structural context in the form of parent phrase category information was incorporated into the grammars. Results show that grammars which contain this information outperform grammars which do not by large margins in all tests for all parsing tasks. It makes the biggest difference for deep parsing, typically corresponding to an improvement of around 5%. Overall, Partition Search with parent phrase category information is shown to be a successful method for learning grammars optimised for a given parsing task, and for minimising grammar size. The biggest margin of improvement over a base grammar was a 5.4% increase in the F-Score for deep parsing. The biggest size reductions were 93.5% fewer nonterminals (for NP identification), and 31.3% fewer rules (for XP chunking)

1 Introduction

Grammar Learning by Partition Search is a computational learning method that constructs probabilistic grammars optimised for a given parsing task. It can be used with varying degrees of supervision and prior knowledge, but its main practical application is the adaptation of grammars to new tasks, where prior knowledge is used in the form of an existing grammar, and learning is supervised in the sense that parsed data samples representing a given parsing task are used. One example application is the adaptation of conventional, "deep" grammars to the shallow parsing tasks (such as NP extraction) involved in many practical NLP applications.

The ability to automatically adapt an existing grammar to a new parsing task saves time and expense. Furthermore, using Partition Search to adapt deep grammars to shallow parsing tasks has a specific advantage: it permits those parts of deeper structural analysis to be retained that are useful for detecting more shallow components, while discarding the rest.

The experiments reported in this paper also investigate the usefulness of parent phrase category information for the different parsing tasks. The general idea is that using *Local Structural Context (LSC)* in PCFG parsing increases parsing accuracy (Belz, 2001, shows this).

The general aim in the experiments was to test the PCFG Partition Search method (without lexicalisation and with parent phrase category information) on a range of parsing tasks for which there are existing results, and to compare the results across the different parsing tasks. Results show that this approach has very different effects on the four parsing tasks.

The remainder of this paper is organised in two main sections. Section 2 provides a brief summary of the main features of Grammar Learning by Partition Search (a much more detailed description can be found in (Belz, 2002)). Section 3 describes experiments and results for the four parsing tasks.

2 Partition Search PCFG Learning

The Partition Search method for PCFG learning belongs to a generic class of CFG learning algorithms in which new CFGs are derived from old by merging and splitting subsets of the old CFG's set of nonterminals. The following is a simple example, where merging the nonterminals NP-SUBJ and NP-OBJ in the CFG G results in a derived CFG G' which has one nonterminal NP instead (assuming a standard 4-tuple definition of CFGs[1]):

$G = (W, N, N^S, R)$,

```
  W =    { NNS, DET, NN, VBD, JJ }
  N =    { S, NP-SUBJ, VP, NP-OBJ }
  N^S =  { S }
  R =    {   S -> NP-SUBJ VP,
             NP-SUBJ -> NNS,
             NP-SUBJ -> DET NN,
             VP -> VBD NP-OBJ,
             NP-OBJ -> NNS,
             NP-OBJ -> DET JJ NNS }
```

$G' = (W, N', N^S, R')$,

```
  W =    { NNS, DET, NN, VBD, JJ }
  N' =   { S, NP, VP }
  N^S =  { S }
  R' =   {   S -> NP VP,
             NP -> NNS,
             NP -> DET NN,
             VP -> VBD NP,
             NP -> DET JJ NNS }
```

Splitting nonterminals is the converse operation. A given *base CFG* together with the merge and split operations defines an infinite search space that can be searched by different search algorithms for a CFG that optimises some given objective function, for example minimising grammar size.

Partition Search is an instance of this generic CFG learning method. It also starts from a given base grammar, but avoids the split operation and makes the search space finite by defining a *maximally split nonterminals set (Max Set)* and the corresponding *maximally split grammar (Max Grammar)* in advance.

Each point in the search space corresponds to a candidate grammar which can be derived from the Max Grammar together with a partition[2] of its set of nonterminals. The set of all partitions of the Max Set thus corresponds to the set of points in the search space[3].

In the current implementation, a variant of beam search is used to search the partition space. Search starts with the Max Grammar and proceeds in one direction by progressively merging more and more nonterminals without permitting a decline in the performance on the given parsing task.

In the experiments reported in this paper, Partition Search is applied to probabilistic CFGs, and is used for supervised grammar learning using prior knowledge in the form of a base grammar derived from a treebank corpus. The Max Grammars are defined on the basis of parent phrase category information. The objective is to find a grammar that is as small as possible and performs as well as possible on a given parsing task (e.g. NP chunking).

3 Learning Grammars for Different Parsing Tasks

3.1 Experiment Set-up

In each set of experiments the objective was to start from a base grammar automatically derived from the Wall Street Journal Corpus (WSJC), define a Max Grammar using information available in the corpus, and then to find a new grammar by Partition Search that performs as well as possible on the given parsing task.

Data: Sections 15–18 of WSJC were used for deriving the base grammar and as the base training corpus, and different randomly selected subsets of Section 1 from the same corpus were used as task-specific training corpora during search. Section 1 was also used as a testing set during the development of the Partition Search method, while Section 20 was used for final performance tests only.

Parsing tasks: Results are reported in this paper for four different parsing tasks:

In *full parsing* (or 'deep' parsing) the task is to assign a complete parse to the input sentence. The parses in the WSJC are used without modification as the gold standard for this task.

In *NP identification* the task is to identify in the input sentence all noun phrases[4], nested and otherwise, in the corresponding WSJC parse.

[1]A Context-Free Grammar (CFG) is a 4-tuple (W, N, N^S, R), where W is a set of terminal symbols, N is a set of nonterminal symbols, N^S is a set of start symbols, and R is a set of production rules.

[2]A partition of a nonempty set A is a subset Π of 2^A such that $\emptyset$ is not an element of Π and each element of A is in one and only one set in Π.

[3]The concept of context-free grammar partitioning in this paper is not directly related to that in (Korenjak, 1969; Weng and Stolcke, 1995).

[4]WSJC categories NP, NX, WHNP and NAC.

The task of *XP chunking* was defined by (Tjong Kim Sang and Buchholz, 2000), and involves dividing a sentence into flat chunks of 11 different types[5]. The target parses are derived from WSJC parses by a conversion procedure[6].

NP chunking was first defined by (Abney, 1991), and involves the identification of flat noun phrase chunks. Target parses are derived from WSJC parses by a conversion procedure[6].

Evaluation: The Brill Tagger was used for POS tagging testing data, and achieved an average accuracy of 97.5%[7]. An existing parser[8] was used to obtain Viterbi parses. If the parser failed to find a complete parse for a sentence, a simple grammar extension method was used to obtain partial parses instead (based on Schmid and Schulte im Walde (2000, p. 728)). The `evalb` program by Sekine and Collins[9] was used to obtain Precision and Recall figures on the parses produced by the parser.

Base grammar derivation: A simple treebank grammar[10] was derived from Sections 15–18 of the WSJ corpus by the following procedure:

1. Iteratively edit the corpus by deleting (i) brackets and labels that correspond to empty category expansions; (ii) brackets and labels containing a single constituent that is not labelled with a POS-tag; (iii) cross-indexation tags; (iv) brackets that become empty through a deletion.
2. Convert each remaining bracketting in the corpus into the corresponding production rule.
3. Collect sets of terminals, nonterminals and start symbols from the corpus. Calculate probabilities for production rules from the rule frequencies by Maximum Likelihood Estimation.

The result of applying this procedure to Sections 15–18 is the base grammar *BARE* which has 10,118 rules and 147 nonterminals.

[5]Corresponding to WSJC categories `NP`, `VP`, `ADVP`, `ADJP`, `PP`, `SBAR`, `CNJP`, `PRT`, `INTJ`, `LST` and `UCP`.

[6]Devised by E. Tjong Kim Sang.

[7]The data was automatically POS tagged in order to make results comparable, in particular to those produced by the LCG Project (Nerbonne et al., 2001) and for CONLL Workshop shared tasks.

[8]LoPar (Schmid, 2000) in its non-head-lexicalised mode. Available from `http://www.ims.uni-stuttgart.de/projekte/gramotron/SOFTWARE/LoPar-en.html`.

[9]`http://cs.nyu.edu/cs/projects/proteus/evalb/`.

[10]The term was coined by Charniak (1996).

Max Grammar definition: Two different approaches to defining the Max Grammar were used. In the first, the base grammar *BARE* itself was used as the Max Grammar. This results in a very restrictive search space definition and amounts to optimising the base grammar in terms of size and performance on a given task without adding any information to it.

In the second approach, Local Structural Context (LSC) was used for defining the Max Grammar (resulting in grammar *PN*). Using LSC means weakening the independence assumptions inherent in PCFGs by making the rules' expansion probabilities dependent on part of their immediate context. The LSC that was used here was parent phrase categories, where the parent of a phrase is the phrase that immediately contains it. Several previous investigations have demonstrated improvement in parsing results due to the inclusion of parent phrase category information (Charniak and Carroll, 1994; Johnson, 1998; Verdú-Mas et al., 2000).

Algorithm parameters: All results were averaged over 5 runs differing only in random selection of the training data set. In all experiments, the variable algorithm parameters were training data set size $x = 50$, beam width $b = 5$, and size of current best set $n = 5$. In the current implementation of the algorithm, the search space is reduced further by avoiding duplicate partitions, and by only considering merges of nonterminals that have the same phrase prefix `NP-`, `VP-` etc.

3.2 Full Parsing

The full parsing results achieved by Partition Search are compared with existing results in the table below. The first result shown is that obtained with grammar *BARE*, i.e. the unmodified treebank grammar extracted with the method described above. The next two results are for two other nonlexicalised treebank methods (Charniak, 1996; Krotov et al., 2000), and the last two are for the best existing lexicalised results (Charniak, 2000; Collins, 2000). The results highlighted in boldface font are the best individual runs of Partition Search (PS) for Sections 1 and 20 of WSJC respectively (both obtained with grammar *PN*). Results in brackets are the *unlabelled* F-Score equivalents (Charniak does not provide labelled results).

Max Grammar	Result type	Iter.	Eval.	F-Score (subset)	Grammar		F-Score	
					Rules	NTs	S 01	S 20
BARE	Max Grammar result	—	—	—	10,118	147	74.09	73.49
	Average	90.2	2,000.8	83.43	9,288.4	59.8	74.05	73.59
	Best size and F-score S 01	114	2,686	86.04	8,409	35	74.54	73.6
	Best F-score S 20	94	2,174	85.12	9,476	55	74.07	73.82
PN	Max Grammar result	—	—	—	16,480	970	77.8	77.57
	Average	426.8	10,559.4	87.87	15,850	545.2	77.87	77.24
	Best size	656	16,335	91.38	15,403	316	77.81	77.14
	Best F-score S 01	625	15,554	89.44	15,608	347	**78.01**	77.41
	Best F-score S 20	145	3,329	86.67	16,338	827	78.00	**77.46**

Table 1: Partition Search results for full parsing task.

	Full parsing
Grammar *BARE* (S 1/20)	74.09/73.49
Nonlex.: Krotov et al. (2000)	(79.12) 76.09
Charniak (1996)	(79.59) — —
Best S 1 PS result (PN)	**(80.54) 78.01**
Best S 20 PS result (PN)	**(79.99) 77.46**
Lexicalised: Charniak (2000)	— — 90.10
Collins (2000)	— — 90.25

The different results are not completely comparable, because the earlier results were obtained training on WSJC Sections 2-21 and testing on Section 23, but given that only 3 sections were used for training in Partition Search, that the earlier results leave out partially parsed sentences, and given that the WSJC is a very homogeneous corpus, the comparison is at least fair to the earlier results.

The huge improvements in parsing accuracy due to lexicalisation are well known, so it is not surprising that Partition Search using an unlexicalised Max Grammar does not get near the best lexicalised results. The main comparison is with the other two nonlexicalised results and there Partition Seach does considerably better.

Table 1 shows the complete set of results for the full parsing experiments. The first column shows the Max Grammar used in a given batch of experiments. The second column indicates the type of result, where the Max Grammar result is grammar size, performance and F-Scores of the Max Grammar itself, and the remaining results are the average and single best results for size and F-Scores achieved by Partition Search. The third and fourth columns show the number of iterations and evaluations carried out during search. Columns 5–9 show details of the final solution grammars: column 5 shows the evaluation score on the training data, column 6 the number of rules, column 7 the number of nonterminals, and columns 8 and 9 the overall F-Score on Sections 1 and 20 respectively.

The best result for Section 1 (78.01) represents a 5.2% increase over base grammar *BARE*, that for Section 20 (77.46) a 5.4% increase.

Results for Section 20 are slightly worse than for Section 1, and the same is true for all other grammar/task combinations, here and below.

3.3 NP identification

For the remaining parsing tasks, the existing results use Section 20 as a testing set. For NP identification, the base grammar *BARE* achieves an F-Score of 78.996 on this section, and Partition Search improves this to 80.89.

The following table shows how these results compare with existing results, all of which are reported in Nerbonne et al. (2001, p. 103).

	NP identification
Chunk Tag Baseline	67.56
Grammar *BARE* (S 20)	78.996
Current best nonlexicalised	80.15
Best S 20 PS result (PN)	**80.89**
Current best lexicalised	83.79

The chunk tag baseline F-Score is obtained by tagging each POS tag in a sentence with its most frequent chunk tag, which is a standard baseline for tasks like this one[11]. The best lexicalised result was achieved with a cascade of memory-based learners. The nonlexicalised result was for a treebank grammar with LSC. The best

[11]Chunk tags may correspond to e.g. the beginning/end/inside/outside of an NP.

Max Grammar	Result type	Iter.	Eval.	F-Score (subset)	Grammar Rules	Grammar NTs	F-Score S 01	F-Score S 20
BARE	Max Grammar result	—	—	—	10,118	147	79.29	78.996
	Average	111.4	2,629	87.83	8,655	37.6	79.10	78.14
	Best size	113	2,679	86.14	8,374	36	78.90	78.06
	Best F-score S 01	114	2,694	90.25	8,541	41	79.51	78.16
	Best F-score S 20	114	2,667	86.88	8,627	35	79.08	78.29
PN	Max Grammar result	—	—	—	16,480	970	82.01	81.31
	Average	852.6	21,051	91.21	13,202.8	119.4	81.41	80.33
	Best size	909	22,474	91.88	12,513	63	80.98	80.22
	Best F-scores	658	16,286	89.57	15,305	314	**82.05**	**80.89**

Table 2: Partition Search results for NP identification task.

Max Grammar	Result type	Iter.	Eval.	F-Score (subset)	Grammar Rules	Grammar NTs	F-Score S 01	F-Score S 20
BARE	Max Grammar result	—	—	—	10,118	147	88.11	87.82
	Average	114.8	2701.4	89.40	7,925.6	34.2	88.66	88.2
	Best size	120	2,791	89.36	6,951	29	88.68	88.13
	Best F-score S 01	115	2,681	85.96	7,679	34	88.75	88.25
	Best F-score S 20	112	2,672	88.34	8,447	37	88.7	88.3
PN	Max Grammar result	—	—	—	16,480	970	89.11	88.52
	Average	674.25	16,413.5	94.3	14,420.5	297.75	88.96	88.34
	Best size	902	22,363	96.150	12,874	70	89.06	88.4
	Best F-scores	679	16,905	95.113	14,895	293	**89.23**	**88.69**

Table 3: Partition Search results for XP chunking task.

Partition Search result for this task also lags behind the best lexicalised result, although by a much smaller margin than for the full parsing task above.

Table 2 (same format as Table 1) contains the complete set of results for Partition Search and the NP identification task. Here the relevant comparison is between the base grammar *BARE* and the best grammars learned by Partition Search. The best result (boldface) was an F-Score of 82.05% for Section 1 (base result was 79.29%), although it was accompanied by an increase in the number of rules from 10,118 to 15,305. This improves grammar *BARE* by 3.48%.

3.4 XP chunking

Base grammar *BARE* gets an F-Score of 87.82 on Section 20 for the XP chunking task, compared to the best Partition Search result of 88.69. The table below compares these results to existing results (again, all for Section 20). The best nonlexicalised result was again for an LSC grammar (Nerbonne et al., 2001, p. 103). The best lexicalised result is for Support Vector Machines (Kudoh and Matsumoto, 2000).

	XP chunking
Chunk Tag Baseline	77.07
Grammar *BARE* (S 20)	87.82
Current best nonlexicalised	88.07
Best S 20 PS result (PN)	**88.52**
Current best lexicalised	93.48

Table 3 contains the complete set of results. The best result (boldface) was an F-Score of 89.23 (base result was 88.11), although this was accompanied by an increase in the number of rules from 10,118 to 14,895. This corresponds to a 1.27% increase over grammar *BARE*.

3.5 NP chunking

Base grammar *BARE* achieves an F-Score of 87.26 on the NP chunking task and Section 20, and the best Partition Search result for Section 20 was 88.83. The best nonlexicalised result was achieved with the decision-tree learner C5.0 (Tjong Kim Sang et al., 2000), and the current overall best result for NP chunking is for

Max Grammar	Result type	Iter.	Eval.	F-Score (subset)	Grammar Rules	Grammar NTs	F-Score S 01	F-Score S 20
BARE	Max Grammar result	—	—	—	10,118	147	88.25	87.26
	Average	116.8	2,749.6	89.64	7,849.6	32.2	88.57	87.51
	Best size	119	2,806	89.79	7,541	30	88.51	87.6
	Best F-score S 01	114	2,674	87.93	7,777	35	88.70	87.59
	Best F-score S 20	116	2,729	93.92	8,071	33	88.69	87.60
PN	Max Grammar result	—	—	—	16,480	970	89.86	88.68
	Average	526	13,007.8	94.85	14,538.3	446	89.83	88.67
	Best size and F-scores	877	21,822	93.85	11,972	95	**90.24**	**88.83**

Table 4: Partition Search results for NP chunking task.

memory-based learning and a lexicalised chunker (Tjong Kim Sang et al., 2000)[12].

	NP chunking
Chunk Tag Baseline	79.99
Grammar *BARE* (S 20)	87.26
Best S 20 PS result (PN)	**88.83**
Current Best: nonlexicalised	90.12
lexicalised	93.25 (93.86)

Table 4 shows the complete set of results. The best result (boldface) was an F-Score of 90.24 (compared to the base result of 88.25), while the number of rules increased from 10,118 to 11,972. This result improves the performance of grammar *BARE* by 2.2%.

4 Comments and Further Research

Partition Search is able to reduce grammar size by merging groups of nonterminals (hence groups of rules) that do not need to be distinguished for a given parsing task. It is able to improve parsing performance firstly by changing parse probabilities (the most likely parse for a sentence under a learned grammar can differ from its most likely parse under the base grammar), and secondly by grammar generalisation (learned grammars parse a superset of the sentences parsed by the base grammar).

The margins of improvement over baseline results were biggest for full parsing and NP identification. Partition Search improved the Max Grammar performance most reliably for NP chunking (grammars *BARE* and *PN*), and XP chunking (*BARE*). It seems particularly difficult to remove significant amounts of parent phrase category information without loss of performance for the NP identification task, where only one run of Partition Search succeeded in improving the *PN* Max Grammar result. The only parsing task for which Partition Search results do *not* outperform the best existing nonlexicalised results is NP chunking. This implies that parent category information and deeper structural analysis (the distinguishing features of this approach) are not as useful for this task.

The results presented in the previous section also show what happens if Partition Search is used as a grammar compression method (when existing grammars are used as Max Grammars). For example, in Table 4, when directly applied to the base grammar *BARE* (top half of table), in the best run for size it reduces the number of nonterminals from 147 to 30 and the number of rules from $10,118$ to $7,541$, while *improving* the overall F-Score. The size reductions on the *PN* base grammar are even bigger: 970 nonterminals down to 95, and $16,480$ rules down to $11,972$, again with a slight improvement in the F-Score (although on average, the F-Score remained about the same). Unlike other grammar compression methods (Charniak, 1996; Krotov et al., 2000), Partition Search achieves lossless compression, in the sense that the compressed grammars are guaranteed to parse all of the sentences parsed by the original grammar.

Grammars incorporating parent phrase category information outperform grammars without such information on all parsing tasks. The difference is biggest for the full parsing task where it typically represents an improvement of around 5%. For grammars *BARE* and *PN* (the only difference between which is parent phrase category information), the difference is 5.01%

[12]Nerbonne et al. (2001) report a slightly better result of 93.86 achieved by combining seven different learning systems.

and 5.55% for Sections 1 and 20 respectively.

Compared to other approaches using parent phrase category information, the approach presented here has the advantage of being able to select a subset of all parent phrase category information on the basis of its usefulness for a given parsing task. This saves on grammar complexity, hence parsing cost.

Further research will look at additionally incorporating lexicalisation, search techniques other than beam search, and optimising the variable parameter combination of the Partition Search algorithm. In a related project, less supervised ways of using Partition Search for grammar learning will be investigated.

5 Acknowledgements

The research reported in this paper was in part funded under the European Union's TMR programme (Grant No. ERBFMRXCT980237).

References

S. Abney. 1991. Parsing by chunks. In R. Berwick, S. Abney, and C. Tenny, editors, *Principle-Based Parsing*, pages 257–278. Kluwer Academic Publishers, Boston.

A. Belz. 2001. Optimising corpus-derived probabilistic grammars. In *Proceedings of Corpus Linguistics 2001*, pages 46–57.

A. Belz. 2002. PCFG learning by nonterminal partition search. In *Proceedings of ICGI 2002 (to appear)*.

E. Charniak and G. Carroll. 1994. Context-sensitive statistics for improved grammatical language models. Technical Report CS-94-07, Department of Computer Science, Brown University.

E. Charniak. 1996. Tree-bank grammars. Technical Report CS-96-02, Department of Computer Science, Brown University.

E. Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of NAACL-2000*, pages 132–139.

M. Collins. 2000. Discriminative reranking for natural language parsing. In *Proceedings of ICML 2000*.

M. Johnson. 1998. PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632.

A. J. Korenjak. 1969. A practical method for constructing LR(k) processors. *Communications of the ACM*, 12(11).

A. Krotov, M. Hepple, R. Gaizauskas, and Y. Wilks. 2000. Evaluating two methods for treebank grammar compaction. *Natural Language Engineering*, 5(4):377–394.

T. Kudoh and Y. Matsumoto. 2000. Use of support vector learning for chunk identification. In *Proceedings of CoNLL-2000 and LLL-2000*, pages 142–144.

J. Nerbonne, A. Belz, N. Cancedda, H. Déjean, J. Hammerton, R. Koeling, S. Konstantopoulos, M. Osborne, F. Thollard, and E. Tjong Kim Sang. 2001. Learning computational grammars. In *Proceedings of CoNLL 2001*, pages 97–104.

H. Schmid and S. Schulte Im Walde. 2000. Robust German noun chunking with a probabilistic context-free grammar. In *Proceedings of COLING 2000*, pages 726–732.

H. Schmid. 2000. LoPar: Design and implementation. Bericht des Sonderforschungsbereiches "Sprachtheoretische Grundlagen für die Computerlinguistik" 149, Institute for Computational Linguistics, University of Stuttgart.

E. Tjong Kim Sang and S. Buchholz. 2000. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of CoNLL-2000 and LLL-2000*, pages 127–132.

E. Tjong Kim Sang, W. Daelemans, H. Déjean, R. Koeling, Y. Krymolowski, V. Punyakanok, and D. Roth. 2000. Applying system combination to base noun phrase identification. In *Proceedings of COLING 2000*, pages 857–863.

J. Luis Verdú-Mas, J. Calera-Rubio, and R. C. Carrasco. 2000. A comparison of PCFG models. In *Proceedings of CoNLL-2000 and LLL-2000*, pages 123–125.

F. L. Weng and A. Stolcke. 1995. Partitioning grammars and composing parsers. In *Proceedings of the 4th International Workshop on Parsing Technologies.*