Learning Grammars for Noun Phrase Extraction 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 an application of Grammar Learning by Partition Search to noun phrase extraction, an essential task in information extraction and many other NLP applications. Grammar Learning by Partition Search is a general method for automatically constructing grammars for a range of parsing tasks; it constructs an optimised probabilistic context-free grammar by searching a space of nonterminal set partitions, looking for a partition that maximises parsing performance and minimises grammar size. The idea is that the considerable time and cost involved in building new grammars can be avoided if instead existing grammars can be automatically adapted to new parsing tasks and new domains. This paper presents results for applying Partition Search to the tasks of (i) identifying flat NP chunks, and (ii) identifying all NPs in a text. For NP chunking, Partition Search improves a general baseline result by 12.7%, and a method-specific baseline by 2.2%. For NP identification, Partition Search improves the general baseline by 21.45%, and the method-specific one by 3.48%. Even though the grammars are nonlexicalised, results for NP identification closely match the best existing results for lexicalised approaches.

1. Introduction

Grammar Learning by Partition Search is a computational learning method that constructs probabilistic grammars optimised for a given parsing task. Its main practical application is the adaptation of grammars to new tasks, in particular the adaptation of conventional, "deep" grammars to the shallow parsing tasks involved in many NLP applications. The parsing tasks investigated in this paper are NP identification and NP chunking both of which involve the detection of NP boundaries, a task which is fundamental to information extraction and retrieval, text summarisation, document classification, and other applications.

The ability to automatically adapt an existing grammar to a new parsing task saves time and expense. Furthermore, adapting deep grammars to shallow parsing tasks has a specific advantage. Existing approaches to NP extraction are mostly completely flat. They do not carry out any structural analysis above the level of the chunks and phrases they are meant to detect. Using Partition Search to adapt deep grammars for shallow parsing permits those parts of deeper structural analysis to be retained that are useful for the detection of more shallow components.

The remainder of this paper is organised in two main sections. Section 2. describes Grammar Learning by Partition Search. Section 3. reports experiments and results for NP identification and NP chunking.

2. Learning PCFGs by Partition Search

Partition Search Grammar Learning starts from the idea that new context-free grammars can be created from old simply by modifying the nonterminal sets, *merging* and *splitting* subsets of nonterminals. For example, for certain parsing tasks it is useful to *split* a single verb phrase category into verb phrases that are headed by a modal verb and those that are not, whereas for other parsing tasks, the added grammar complexity is avoidable. In another context, it may not be necessary to distinguish noun phrases in subject position from first objects and second objects, making it possible to *merge* the three categories into one.

The usefulness of such split and merge operations can be objectively measured by their effect on a grammar's size (number of rules and nonterminals) and performance (parsing accuracy on a given task). Grammar Learning by Partition Search automatically tries out different combinations of merge and split operations and therefore can automatically optimise a grammar's size and performance.

2.1. Preliminary definitions

Definition 1 Set Partition

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.

The partition of A where all elements are singleton sets is called the *trivial partition* of A.

Definition 2 Probabilistic Context-Free Grammar

A Probabilistic Context-Free Grammar (PCFG) is a 4-tuple (W, N, N^S, R), where W is a set of terminal symbols, N is a set of nonterminal symbols, $N^S = \{(s_1, p(s_1)), \ldots (s_l, p(s_l))\}, \{s_1, \ldots s_l\} \subseteq N$ is a set of start symbols with associated probabilities summing to one, and $R = \{(r_1, p(r_1)), \ldots (r_m, p(r_m))\}$ is a set of rules with associated probabilities. Each rule r_i is of the form $n \rightarrow \alpha$, where n is a nonterminal, and α is a string of terminals and nonterminals. For each nonterminal n, the values of all $p(n \rightarrow \alpha_i)$ sum to one, or: $\sum_{i:(n \rightarrow \alpha_i, p(n \rightarrow \alpha_i)) \in R} p(n \rightarrow \alpha_i) = 1$.

2.2. Generalising and Specialising PCFGs through Nonterminal Set Operations

2.2.1. Nonterminal merging

Consider two PCFGs G and G':

$$\begin{array}{lll}
\multicolumn{3}{l}{G = (W, N, N^S, R),} \\
\quad W = & \{\, \texttt{NNS, DET, NN, VBD, JJ} \,\} & \\
\quad N = & \{\, \texttt{S, NP-SUBJ, VP, NP-OBJ} \,\} & \\
\quad N^S = & \{\, (\texttt{S}, 1) \,\} & \\
\quad R = & \{ & (\texttt{S -> NP-SUBJ VP}, 1), \\
 & & (\texttt{NP-SUBJ -> NNS}, 0.5), \\
 & & (\texttt{NP-SUBJ -> DET NN}, 0.5), \\
 & & (\texttt{VP -> VBD NP-OBJ}, 1), \\
 & & (\texttt{NP-OBJ -> NNS}, 0.75), \\
 & & (\texttt{NP-OBJ -> DET JJ NNS}, 0.25) \,\}
\end{array}$$

$$\begin{array}{lll}
\multicolumn{3}{l}{G' = (W, N', N^S, R'),} \\
\quad W = & \{\, \texttt{NNS, DET, NN, VBD, JJ} \,\} & \\
\quad N' = & \{\, \texttt{S, NP, VP} \,\} & \\
\quad N^S = & \{\, (\texttt{S}, 1) \,\} & \\
\quad R' = & \{ & (\texttt{S -> NP VP}, 1), \\
 & & (\texttt{NP -> NNS}, 0.625), \\
 & & (\texttt{NP -> DET NN}, 0.25), \\
 & & (\texttt{VP -> VBD NP}, 1), \\
 & & (\texttt{NP -> DET JJ NNS}, 0.125) \,\}
\end{array}$$

Intuitively, to derive G' from G, the two nonterminals NP-SUBJ and NP-OBJ are merged into a single new nonterminal NP. This merge results in two rules from R becoming identical in R': both NP-SUBJ -> NNS and NP-OBJ -> NNS become NP -> NNS. One way of determining the probability of the new rule NP -> NNS is to sum the probabilities of the old rules and renormalise by the number of nonterminals that are being merged[1]. In the above example therefore $p(\texttt{NP -> NNS}) = (0.5 + 0.75)/2 = 0.625$[2].

An alternative would be to reestimate the new grammar on some corpus, but this is not appropriate in the current context: merge operations are used in a search process (see below), and it would be expensive to reestimate each new candidate grammar derived by a merge. It is better to use any available training data to estimate the original grammar's probabilities, then the probabilities of all derived grammars can simply be calculated as described above without expensive corpus reestimation.

The new grammar G' derived from an old grammar G by merging nonterminals in G is a generalisation of G: the language of G', or $L(G')$, is a superset of the language of G, or $L(G)$. E.g., det jj nns vbd det jj nns is in $L(G')$ but not in $L(G)$. The set of parses assigned to a sentence s by G' differs from the set of parses assigned to s by G. The probabilities of parses for s can change, and so can the probability ranking of the parses, i.e. the most likely parse for s under G may be different from the most likely parse for s under G'. Finally, G' has the same number of rules as G or fewer.

[1]Reestimating the probabilities on the training corpus would of course produce identical results.

[2]Renormalisation is necessary because the probabilities of all rules expanding the same nonterminal sum to one, therefore the probabilities of all rules expanding a new nonterminal resulting from merging n old nonterminals will sum to n.

2.2.2. Nonterminal splitting

Deriving a new PCFG from an old one by splitting nonterminals in the old PCFG is not quite the exact reverse of deriving a new PCFG by merging nonterminals. The difference lies in determining probabilities for new rules. Consider the following grammars G and G':

$$\begin{array}{lll}
\multicolumn{3}{l}{G = (W, N, N^S, R),} \\
\quad W = & \{\, \texttt{NNS, DET, NN, VBD, JJ} \,\} & \\
\quad N = & \{\, \texttt{S, NP, VP} \,\} & \\
\quad N^S = & \{\, (\texttt{S}, 1) \,\} & \\
\quad R = & \{ & (\texttt{S -> NP VP}, 1), \\
 & & (\texttt{NP -> NNS}, 0.625), \\
 & & (\texttt{NP -> DET NN}, 0.25), \\
 & & (\texttt{VP -> VBD NP}, 1), \\
 & & (\texttt{NP -> DET JJ NNS}, 0.125) \,\}
\end{array}$$

$$\begin{array}{lll}
\multicolumn{3}{l}{G' = (W, N', N^S, R'),} \\
\quad W = & \{\, \texttt{NNS, DET, NN, VBD, JJ} \,\} & \\
\quad N' = & \{\, \texttt{S, NP-SUBJ, VP, NP-OBJ} \,\} & \\
\quad N^S = & \{\, (\texttt{S}, 1) \,\} & \\
\quad R' = & \{ & (\texttt{S -> NP-SUBJ VP}, ?), \\
 & & (\texttt{S -> NP-OBJ VP}, ?), \\
 & & (\texttt{NP-SUBJ -> NNS}, ?), \\
 & & (\texttt{NP-SUBJ -> DET NN}, ?), \\
 & & (\texttt{NP-SUBJ -> DET JJ NNS}, ?) \,\} \\
 & & (\texttt{VP -> VBD NP-SUBJ}, ?), \\
 & & (\texttt{VP -> VBD NP-OBJ}, ?), \\
 & & (\texttt{NP-OBJ -> NNS}, ?), \\
 & & (\texttt{NP-OBJ -> DET NN}, ?), \\
 & & (\texttt{NP-OBJ -> DET JJ NNS}, ?) \,\}
\end{array}$$

To derive G' from G, the single nonterminal NP is split into two nonterminals NP-SUBJ and NP-OBJ. This split results in several new rules. For example, for the old rule NP -> NNS, there now are two new rules NP-SUBJ -> NNS and NP-OBJ -> NNS. One possibility for determining the new rule probabilities is to redistribute the old probability mass evenly among them, i.e. $p(\texttt{NP -> NNS}) = p(\texttt{NP-SUBJ -> NNS}) = p(\texttt{NP-SUBJ -> NNS})$. However, then there would be no benefit at all from performing such a split: the resulting grammar would be larger, the most likely parses remain unchanged, and for each parse p under G that contains a nonterminal participating in a split operation, there would be at least two equally likely parses under G'.

The new probabilities cannot be calculated directly from G. The redistribution of the probability mass has to be motivated from a knowledge source outside of G. One way to proceed is to estimate the new rule probabilities on the original corpus — provided that it contains the information on the basis of which a split operation was performed in extractable form. For the current example, a corpus in which objects and subjects are annotated could be used to estimate the probabilities of the rules in G', and might yield the following result (which reflects the fact that in English, the NP in a sentence NP VP is usually a subject, whereas the NP in a VP consisting of a verb followed by an NP is an object):

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

```
    W  =  { NNS, DET, NN, VBD, JJ }
    N' =  { S, NP-SUBJ, VP, NP-OBJ }
    N^S = { (S, 1) }
    R' =  {  (S -> NP-SUBJ VP, 1),
             (S -> NP-OBJ VP, 0),
             (NP-SUBJ -> NNS, 0.5),
             (NP-SUBJ -> DET NN, 0.5),
             (NP-SUBJ -> DET JJ NNS, 0) }
             (VP -> VBD NP-SUBJ, 0),
             (VP -> VBD NP-OBJ, 1),
             (NP-OBJ -> NNS, 0.75),
             (NP-OBJ -> DET NN, 0),
             (NP-OBJ -> DET JJ NNS, 0.25) }
```

With rules of zero probability removed, G' is identical to the original grammar G in the example in the previous section.

2.3. Partition Search

A PCFG together with nonterminal merge and split operations defines a space of derived grammars which can be searched for a new PCFG that optimises some given objective function. The disadvantage of this search space is that it is infinite, and each split operation requires the reestimation of rule probabilities from a training corpus, making it computationally much more expensive than a merge operation.

However, there is a simple way to make the search space finite, and at the same time to make split operations redundant. The resulting method, Grammar Learning by Partition Search, is summarised in this section (Partition Search is described in more detail, including formal definitions and algorithmic details, in Belz (2002)).

2.3.1. PCFG Partitioning

An arbitrary number of merges can be represented by a partition of the set of nonterminals. For the example presented in Section 2.2.1. above, the partition of the nonterminal set N in G that corresponds to the nonterminal set N' in G' is { {S}, {NP-SBJ, NP-OBJ}, {VP} }. The original grammar G together with a partition of its nonterminal set fully specifies the new grammar G': the new rules and probabilities, and the entire new grammar G' can be derived from the partition together with the original grammar G. The process of obtaining a new grammar G', given a base grammar G and a partition of the nonterminal set N of G will be called PCFG Partitioning[3].

2.3.2. Search space

The search space for Grammar Learning by Partition Search can be made finite and searchable entirely by merge operations (grammar partitions).

Making the search space finite: The number of merge operations that can be applied to a nonterminal set is finite, because after some finite number of merges there remains only one nonterminal. On the other hand, the number of split operations that can sensibly be applied to a nonterminal *NT* has an upper bound in the number of different terminals strings dominated by *NT* in a corpus of evidence (e.g. the corpus the PCFG was trained on). For example, when splitting the nonterminal NP into subjects and objects, there would be no point in creating more new nonterminals than the number of different subjects and objects found in the corpus.

Given these (generous) bounds, there is a finite number of distinct grammars derivable from the original grammar by different combinations of merge and split operations. This forms the basic space of candidate solutions for Grammar Learning by Partition Search.

Making the search space searchable by grammar partitioning only: Imposing an upper limit on the number and kind of split operations permitted not only makes the search space finite but also makes it possible to directly derive this *maximally split nonterminal set* (Max Set). Once the Max Set has been defined, the single grammar corresponding to it — the *maximally split Grammar* (Max Grammar) — can be derived and retrained on the training corpus.

The set of points in the search space corresponds to the set of partitions of the Max Set. Search for an optimal grammar can thus be carried out directly in the partition space of the Max Grammar.

Structuring the search space: The finite search space can be given hierarchical structure as shown in Figure 1 for an example of a very simple base nonterminal set {NP, VP, PP}, and a corpus which contains three different NPs, three different VPs and two different PPs.

At the top of the graph is the Max Set. The sets at the next level down (level 7) are created by merging pairs of nonterminals in the Max Set, and so on for subsequent levels. At the bottom is the *maximally merged nonterminal set* (Min Set) consisting of a single nonterminal *NT*. The sets at the level immediately above it can be created by splitting *NT* in different ways. The sets at level 2 are created from those at level 1 by splitting one of their elements. The original nonterminal set ends up somewhere in between the top and bottom (at level 3 in this example).

While this search space definition results in a finite search space and obviates the need for the expensive split operation, the space will still be vast for all but trivial corpora. In Section 3.3. below, alternative ways for defining the Max Set are described that result in much smaller search spaces.

2.3.3. Search task and evaluation function

The input to the Partition Search procedure consists of a base grammar G_0, a base training corpus C, and a task-specific training corpus D^T. G_0 and C are used to create the Max Grammar G. The **search task** can then be defined as follows:

> Given the maximally split PCFG $G = (W, N, N^S, R)$, a data set of sentences D, and a set of target parses D^T for D, find a partition Π_N of N that derives a grammar $G' = (W, \Pi_N, N^{S'}, R')$, such that $|R'|$ is minimised, and $f(G', D, D^T)$ is maximised, where f scores the performance of G' on D as compared to D^T.

[3]The concept of context-free grammar partitioning in this paper is not directly related to that in (Korenjak, 1969; Weng and Stolcke, 1995), and later publications by Weng et al. In these previous approaches, a non-probabilistic CFG's *set of rules* is partitioned into subsets of rules. The partition is drawn along a specific nonterminal NT, which serves as an interface through which the subsets of rules (hence, subgrammars) can communicate after partition (one grammar calling the other).

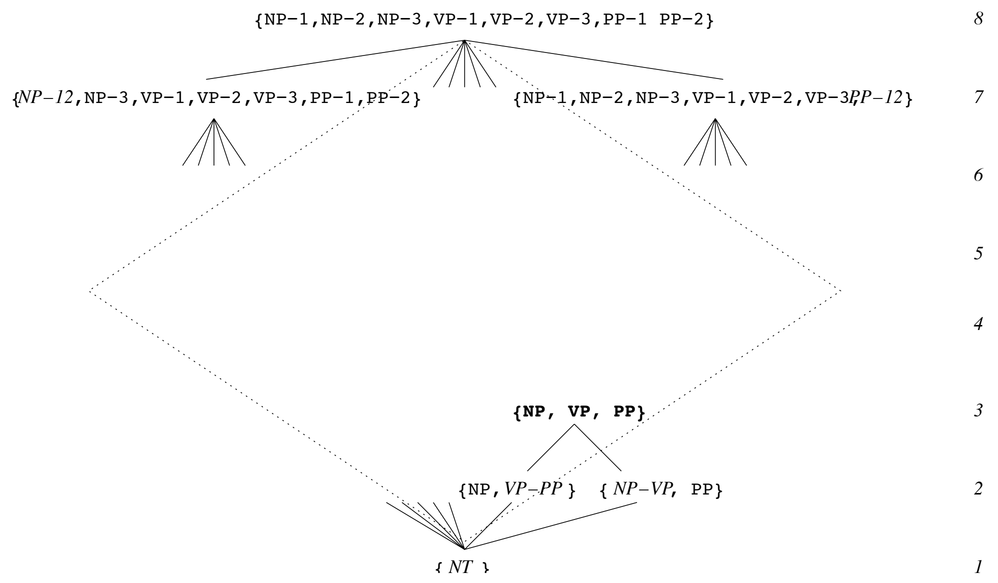
{NP-1,NP-2,NP-3,VP-1,VP-2,VP-3,PP-1 PP-2}
{NP-12,NP-3,VP-1,VP-2,VP-3,PP-1,PP-2}
{NP-1,NP-2,NP-3,VP-1,VP-2,VP-3,PP-12}
{NP, VP, PP}
{NP, VP-PP }
{ NP-VP, PP}
{ NT }
8
7
6
5
4
3
2
1

Figure 1: Simple example of a partition search space.

The size of the nonterminal set and hence of the grammar decreases from the top to the bottom of the search space. Therefore, if the partition space is searched top-down, grammar size is minimised automatically and does not need to be assessed explicitly.

In the current implementation, the **evaluation function** f simply calculates the F-Score achieved by a candidate grammar on D as compared to D^T. The F-Score is obtained by combining the standard PARSEVAL evaluation metrics *Precision* and *Recall*[4] as follows: $2 \times Precision \times Recall / (Precision + Recall)$.

An existing parser[5] 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)).

2.3.4. Search algorithm

Since each point in the search space can be accessed directly by applying the corresponding nonterminal set partition to the Max Grammar, the search space can be searched in any direction by any search method using partitions to represent candidate grammars.

In the current implementation, a variant of beam search is used to search the partition space top down. A list of the n current best candidate partitions is maintained (initialised to the Max Set). For each of the n current best partitions a random subset of size b of its children in the hierarchy is generated and evaluated. From the union of current best partitions and the newly generated candidate partitions, the n best elements are selected and form the new current best set. This process is iterated until either no new partitions can be generated that are better than their parents, or the lowest level of the partition tree is reached. In each iteration the size of the nonterminal set (partition) decreases by one.

The size of the search space grows exponentially with the size i of the Max Set. However, the complexity of the Partition Search algorithm is only $O(nbi)$, because only up to $n \times b$ partitions are evaluated in each of up to i iterations[6].

3. Learning NP Extraction Grammars

3.1. Data and Parsing Tasks

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 20 was used for final performance tests.

Results are reported in this paper for the following two parsing tasks. In **NP identification** the task is to identify in the input sentence all noun phrases[7], nested and otherwise, that are given in the corresponding WSJC parse. **NP chunking** was first defined by (Abney, 1991), and involves the identification of flat noun phrase chunks. Target parses were derived from WSJC parses by an existing conversion procedure[8].

The Brill Tagger was used for POS tagging testing data, and achieved an average accuracy of 97.5% (as evaluated by `evalb`).

3.2. Base grammar

A simple treebank grammar[9] 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

[4]I used the `evalb` program by Sekine and Collins (`http://cs.nyu.edu/cs/projects/proteus/evalb/`) to obtain Precision and Recall figures.

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

[6]As before, n is the number of current best candidate solutions, b is the width of the beam, and i is the size of the Max Set.

[7]Corresponding to the WSJC categories `NP, NX, WHNP` and `NAC`.

[8]Devised by Erik Tjong Kim Sang for the TMR project *Learning Computational Grammars*.

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

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 W, nonterminals N and start symbols N^S from the corpus. Probabilities p for rules $n \rightarrow \alpha$ are calculated from the rule frequencies C by Maximum Likelihood Estimation: $p(n \rightarrow \alpha) = \frac{C(n \rightarrow \alpha)}{\sum_i C(n \rightarrow \alpha^i)}$.

This procedure creates the base grammar *BARE* which has $10,118$ rules and 147 nonterminals.

3.3. Restricting the search space further

The simple method described in Section 2.3.2. for defining the maximally split nonterminal set (Max Set) tends to result in vast search spaces. Using parent node (PN) information to create the Max Set is much more restrictive and linguistically motivated. The Max Grammar *PN* used in the experiments reported below can be seen as making use of *Local Structural Context* (Belz, 2001): the independence assumptions inherent in PCFGs are weakened by making the rules' expansion probabilities dependent on part of their immediate structural context (here, its parent node). To obtain the grammar *PN*, the base grammar's nonterminal set is maximally split on the basis of the *parent node* under which rules are found in the base training corpus[10]. Several previous investigations have demonstrated improvement in parsing results due to the inclusion of parent node information (Charniak and Carroll, 1994; Johnson, 1998; Verdú-Mas et al., 2000).

Another possibility is to use the base grammar *BARE* itself as the Max Grammar. This is a very restrictive search space definition and amounts to an attempt to optimise the base grammar in terms of its size and its performance on a given task without adding any information. Results are given below for both *BARE* and *PN* as Max Grammars.

In the current implementation of the algorithm, the search space is reduced further by avoiding duplicate partitions, and by only allowing merges of nonterminals that have the same phrase prefix NP-*, VP-* etc.

The Max Grammars end up having sets of nonterminals that differ from the bracket labels used in the WSJC: while the phrase categories (e.g. NP) are the same, the tags (e.g. *-S, *-3) on the phrase category labels may differ. In the evaluation, all labels starting with the same phrase category prefix are considered equivalent.

3.4. NP chunking results

Baseline Results. Base grammar *BARE* (see Section 3.2. achieves an F-Score of 88.25 on the NP chunking task. This baseline result compares as follows with existing results:

	NP chunking
Chunk Tag Baseline	79.99
Grammar *BARE*	88.25
Current Best: nonlexicalised	90.12
lexicalised	93.25 (93.86)

[10]The parent node of a phrase is the category of the phrase that immediately contains it.

The chunk tag baseline F-Score is the standard baseline for the NP chunking task and is obtained by tagging each POS tag in a sentence with the label of the phrase that it most frequently appears in, and converting these phrase tags into labelled bracketings (Nerbonne et al., 2001, p. 102). 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 memory-based learning and a lexicalised chunker (Tjong Kim Sang et al., 2000)[11].

Table 1 shows results for Partition Search applied to the NP chunking task. 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 the F-Score, grammar size and number of nonterminals of the Max Grammar itself, and the remaining results are the average and single best results achieved by Partition Search. The third and fourth columns show the number of iterations and evaluations carried out before search stopped. Columns 5–8 show details of the final solution grammars: column 5 shows the evaluation score on the training data, column 6 the overall F-Score on the testing data, column 7 the size, and the last column gives the number of nonterminals.

The best result (boldface) was an F-Score of 90.24% (compared to the base result of 88.25%), and 95 nonterminals (147 in the base grammar), while the number of rules increased from 10,118 to 11,972. This result improves the general baseline by 12.7% and the performance by grammar *BARE* by 2.2%. It also outperforms the best existing result of 90.12% for nonlexicalised NP chunking by a small margin.

3.5. NP identification results

Baseline Results. Base grammar *BARE* achieves an F-Score of 79.29 on the NP identification task. This baseline result compares as follows with existing results:

	NP identification
Chunk Tag Baseline	67.56
Grammar *BARE*	79.29
Current Best: nonlexicalised	80.15
lexicalised	83.79

All results in this table (except for that for grammar *BARE*) are reported in Nerbonne et al. (2001, p. 103). The task definition used there was slightly different in that it omitted two minor NP categories (WSJC brackets labelled NAC and NX). The slightly different task definition has only a very small effect on F-Scores, so the above results are comparable. The chunk tag baseline F-Score was again obtained by tagging each POS tag in a sentence with the label of the phrase that it most frequently appears in. The best lexicalised result was achieved with a cascade of memory-based learners. The same paper also included two results for nonlexicalised NP identification.

Table 2 (same format as Table 1) contains results for Partition Search and the NP identification task. The smallest nonterminal set had 63 nonterminals (147 in the base

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

Max Grammar		Iter.	Eval.	F-Score (subset)	F-Score (WSJC S 1)	Size (rules)	Nonterms
BARE	Max Grammar result:				88.25	10,118	147
	Average:	116.8	2,749.6	89.64	88.57	7,849.6	32.2
	Best (size):	119	2,806	89.79	88.51	7,541	30
	Best (F-score):	114	2,674	87.93	88.70	7,777	35
PN	Max Grammar result:				89.86	16,480	970
	Average:	526	13,007.75	94.85	89.83	14,538.25	446
	Best (size and F-score):	**877**	**21,822**	**93.85**	**90.24**	**11,972**	**95**

Table 1: Partition tree search results for NP chunking task, WSJC Section 1 (averaged over 5 runs, variable parameters: $x = 50, b = 5, n = 5$).

Max Grammar		Iter.	Eval.	F-Score (subset)	F-Score (WSJC S 1)	Size (rules)	Nonterms
BARE	Max Grammar result:				79.29	10,118	147
	Average	111.4	2,629	87.831	79.10	8,655	37.6
	Best (size):	113	2,679	86.144	78.9	8,374	36
	Best (F-score):	114	2,694	90.246	79.51	8,541	41
PN	Max Grammar result:				82.01	16,480	970
	Average:	852.6	21,051	91.2098	81.41308	13,202.8	119.4
	Best (size):	909	22,474	91.881	80.9830	12,513	63
	Best (F-score):	**658**	**16,286**	**89.572**	**82.0503**	**15,305**	**314**

Table 2: Partition tree search results for NP identification task, WSJC Section 1 (averaged over 5 runs, variable parameters: $x = 50, b = 5, n = 5$).

grammar). The best result (boldface) was an F-Score of 82.05% (base result was 79.29%), while the number of rules increased from 10,118 to 15,305. This improves the general baseline by 21.45% and grammar *BARE* by 3.48%. It also outperforms the other two results for nonlexicalised NP chunking by a significant margin, and even comes close to the best lexicalised result (83.79%).

3.6. General comments

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 task. It is able to improve parsing performance firstly by grammar generalisation (partitioned grammars parse a superset of the sentences parsed by the base grammar), and secondly by reranking parse probabilities (the most likely parse for a sentence under a partitioned grammar can differ from its most likely parse under the base grammar).

The margins of improvement over baseline results were bigger for the NP identification task than for NP chunking. The results reported here for NP chunking are no match for the best lexicalised results, whereas the results for NP identification come close to the best lexicalised results. This indicates that the two characteristics that most distinguish the grammars used here from other approaches — some non-shallow structural analysis and parent node information — are more helpful for NP identification.

Preliminary tests revealed that results were surprisingly constant over different combinations of variable parameter values, although training subset size of less then 50 meant unpredictable results for the complete WSJC Section 1. For a random subset of size 50 and above, there is an almost complete correspondence between subset F-Score and Section 1 F-Score, i.e. higher subset F-Score almost always means higher Section 1 F-Score.

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). In Table 1, for example, when applied to the base grammar *BARE* (four top rows), it maximally 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* 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 (even though 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 be able to parse all of the sentences parsed by the original grammar.

Compared to other approaches using parent node information (Charniak and Carroll, 1994; Johnson, 1998; Verdú-Mas et al., 2000), the approach presented here has the advantage of being able to select a subset of all parent node information on the basis of its usefulness for a given parsing task. This saves on grammar complexity, hence parsing cost.

3.7. Nonterminal distinctions preserved/eliminated

The base grammar *BARE* has 26 different phrase category prefixes (S, NP, etc.). The additional tags encoding grammatical function and parent node information results in much larger numbers of nonterminals. One of the aims of partition search is to reduce this number, preserving only useful distinctions. This section looks at nonterminal distinctions that were preserved and eliminated for each task and grammar.

3.7.1. Base grammar *BARE* (functional tags only)

Twelve of the 26 phrase categories are not annotated with functional tags in the WSJC. The remaining 14 phrase categories have between 2 and 28 grammatical function subcategories[12].

In the *BARE* grammar, more nonterminals were merged on average in the NP chunking task (32.2 remaining) than in the NP identification task (37.6 remaining). This is as might be expected since the NP identification task looks the more complex.

Results for NP chunking show a very strong tendency to merge the subcategories of all phrase categories except for two: NP and PP. With only the rare exception, the distinction between different grammatical functions is eliminated for the other 12 out of 14 phrase categories. By contrast, for NP, between 2 and 5 different categories remain (average 2.8), and for PP, between 2 and 4 remain (average 3.6). This implies that for NP chunking only the different grammatical functions of NPs and PPs are useful.

Results for NP identification show a tendency to perserve distinctions among the subcategories of SBAR, NP and PP and to a lesser extent among those of ADVP and ADJP. Other distinctions tend to be eliminated. All subcategories of SBARQ, NX, NAC, INTJ and FRAG are always merged, UCP and SINV nearly always.

3.7.2. Grammar *PN* (parent node tags)

The *PN* grammar has 970 phrase subcategories for the 26 basic phrase categories of which only those with the largest numbers of subcategories are examined here: NP (173), PP (173), ADVP (118), S (76), and VP (62).

Surprisingly, far fewer nonterminals were merged on average in the NP chunking task (446 remaining) than in the NP identification task (only 119.4 remaining).

In both tasks, although more so in the NP chunking task, the strongest tendency was that far more NP subcategories were preserved than any other.

In the NP identification task, the different NAC and NX subcategories were always merged into a single one, whereas in the NP chunking task, at least 4 different NAC and 3 different NX subcategories remained.

In both tasks equally, ADVP and PP distinctions were mostly eliminated. The same goes for VP distinctions although VPs with parent node S, SBAR and VP had a higher tendency to remain unmerged.

These results indicate that by far the most important parent node information for both NP identification and chunking are the parent nodes of the NPs themselves. More detailed analysis of merge sets would be needed to see what exactly this means.

[12] ADJP: 6, ADVP: 18, FRAG: 2, INTJ: 2, NAC: 4, NP: 23, NX: 2, PP: 28, S: 14, SBAR: 20, SBARQ: 3, SINV: 2, UCP: 8, VP: 3.

4. Conclusions and Further Research

Grammar Learning by Partition Search was shown to be an efficient method for constructing PCFGs optimised for a given parsing task. In the nonlexicalised applications reported in this paper, the performance of the base grammar was improved by up to 3.48%. This corresponds to an improvement of up to 21.45% over the standard baseline. The result for NP chunking is slightly better than the best existing result for nonlexicalised NP chunking, whereas the result for NP identification closely matches the best existing result for lexicalised NP identification.

Partition Search can also be used to simply reduce grammar size, if an existing grammar is used as the Max Grammar. In the experiments reported in this paper, Partition Search reduced the size of nonterminal sets by up to 93.5%, and the size of rule sets by up to 27.4%. Compared to other grammar compression techniques, it has the advantage of being lossless.

Further research will look at additionally incorporating lexicalisation, other search methods, and other variable parameter combinations.

5. Acknowledgements

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

6. References

Steven 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. Grammar learning by partition search. In *Proceedings of LREC Workshop on Event Modelling for Multilingual Document Linking*.

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

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

Mark 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.

J. Nerbonne, A. Belz, N. Cancedda, Hervé 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, 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.

Jose Luis Verdú-Mas, Jorge Calera-Rubio, and Rafael 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*.