Data-Oriented Parsing

Rens Bod, Remko Scha and Khalil Sima'an (eds.)

October 10, 2001

CENTER FOR THE STUDY
OF LANGUAGE
AND INFORMATION

1

Compositional partial parsing by memory-based sequence learning

Ido Dagan and Yuval Krymolowski

We present a memory-based learning method that recognizes shallow patterns in a new text based on a bracketed training corpus. The system exploits internal phrase structure in order to recognize compositional patterns as well. The examples are stored as-is, in efficient data structures. Generalization is performed at recognition time by examining POS subsequences. The number of occurences where a subsequence supports a hypothesised instance ("positive" evidence) is compared with the number of other occurences of the subsequence ("negative" evidence). The method is oriented for learning to parse any selected subset of target syntactic structures. It is local, yet can handle also compositional structures. Parts of speech, as well as instances of embedded patterns, are being used simultaneously. The output is a partial parse in which instances of the target structures are marked. Experimental results are presented for recognizing noun and verb phrases, as well as subject-verb and verb-object relations. We discuss analogies with DOP.

1.1 Introduction

Shallow parsing is the task of identifying relatively short and simple syntactic constructs in a sentence (usually phrases) and the functional relations between them, based on "local" evidence within and in the vicinity of the detected construct (Abney, 1991; Greffenstette, 1993). Finding phrases is often replaced for *chunking* – dividing the sentence into non-overlapping sequences of syntactic structures, which may not

Data-Oriented Parsing.
Rens Bod, Remko Scha and Khalil Sima'an (eds.).

necessarily always correspond to phrases (Abney, 1991). Usually these are non-recursive sequences, like "base noun phrases" which do not contain noun-phrase (NP) constituents (Ramshaw and Marcus, 1995, hereafter RM95).

Most of the chunking works have concentrated on noun-phrases, naturally due to their central role in a sentence e.g., (Church, 1988), (Ramshaw and Marcus, 1995), (Veenstra, 1998). Other chunking tasks involve recognizing subject-verb (SV) and verb-object (VO) pairs (Muñoz et al., 1999; Argamon, Dagan, and Krymolowski, 1999, hereafter ADK99). These and other works have demonstrated that good chunking performance may be achieved using *local* information, about a word and its nearby context of two or three words. In addition, good results were obtained using parts-of-speech (POS) information alone.

A *partial parser* (Abney, 1996) can produce an analysis at an intermediate level between shallow and full parsing. A partial parse identifies recursive phrases, like an NP which has NP constituents, or an NP embedded within a verb phrase (VP). Analysis is partial because it includes only certain parts of the phrase structure tree.

Applications of shallow and partial parsing include tasks that require extraction of data from the text rather than a full analysis. Such applications include information extraction, question answering, document classification, and summarization.

A partial parser is typically concerned with only a small number of target syntactic patterns, and may therefore require less training information. That would not be the case when evaluating a full parser on selected target patterns, because its training material would still include full parse-trees labeled with other patterns as well.

We present here a memory-based trainable partial parser, which combines substrings of training instances in order to make generalizations. Our approach attempts to reduce the gap between shallow and full parsing. The algorithm is an extension of a chunking *flat* method (Argamon, Dagan, and Krymolowski, 1998, ADK99) towards handling composite patterns, and discovering multiple syntactic structures simultaneously rather than only one type of structure at a time. The extended method is still local, and maintains the simplicity of the flat algorithm. The partial parser was presented in (Krymolowski and Dagan, 2000).

Previous related work is described in section 1.2. Section 1.3 describes the flat inference algorithm, which relies on POS data only. In Section 1.4, this algorithm is evaluated on three target syntactic patterns in English. Section 1.5 describes the extended algorithm, which takes compositionality into account. Evaluation of the compositional algorithm follows in Section 1.6, with discussion in Section 1.7.

1.2 Background

The concept of a chunk was presented by Abney (1991), grounded on psycholinguistic evidence. Prior to this work, Church (1988) presented a statistical system for finding noun phrases. Ramshaw and Marcus (1995) invoked transformation-based learning (Brill, 1993) for recognizing noun and verb chunks. Later works have also used a chunking approach to finding subject-verb and verb-object pairs. State-of-the-art results on learning to find chunks of most types simultaneously is presenteed in (Tjong Kim Sang and Buchholz, 2000) with references therein.

The algorithm of Daelemans, Buchholz, and Veenstra (1999) finds verb and noun chunks before proceeding with SV and VO detection. Buchholz, Veenstra, and Daelemans (1999) present a cascaded approach for assigning grammatical relations involving verbs. They chunk the text first, then use chunk information for finding adverbial functions, and finally use all information for assigning the relations. The work of Tjong Kim Sang (1999) presents a system for detecting compositional noun phrases by repeated chunking.

The system presented here employs a chunking method for finding compositional structures, using a single model and a single inference step. It learns the structures from *local* POS information, but can detect structures spanning long ranges. The choice of which patterns to learn is dynamic, and it is possible to use the system for partial parsing, for example by learning verb phrases and noun phrases. It is possible to produce a more detailed parse by specifying more patterns.

Skut and Brants (1998a) presented a markov model for partial parsing with limited depth (up to 3). They have used features comprised of a few attributes: structural tags, representing the relation between dominating nodes of the current and the preceding word, POS tag, and two tags denoting the phrasal categories of the parent and grandparent node. They have found the structural and POS tags to provide most of the information. In addition, they have found that a bigram model was enough, the improvement from trigram model being very small. They explained it by the richness of information encoded in the structural tags. In another work, Skut and Brants (1998b) presented a partial parser that uses maximum entropy for estimating contextual probabilities, using similar features as before. At this work they used trigrams, and allowed abstractions on some of the attributes.

An approach which bears some similarity to ours was presented by Sekine (1998, "Apple Pie Parser"). His system is based on learning grammar rules for five non-terminals: sentence, subordinate sentence, infinitive sentence, NP, and base NP; the system also uses lexical dependency

information. Even after 40,000 training sentences, each new sentence produced, on the average, a new rule. The parsing rules were mapped to an automaton in order to handle their large number.

The part similar to out approach is "fitted parsing" – combining partial trees or chunks that the rules produced for creating a full parse tree. With combined chunks restricted to `S` nodes, less than 1% of the test sentences required fitting (further comparison is provided in Section 1.7.)

Like DOP and Apple Pie Parser, our method works by combining evidence fragments. It differs from these hierarchical method because it relies only on *local* information. Each instance is detected independently, making the method more robust as there is no attempt to produce a global parse. In that sense, we employ a shallow-parsing method for partial parsing as well.

1.3 The Flat MBSL Algorithm

The input to the flat Memory-Based Sequence Learning (MBSL) algorithm is a POS-tagged training corpus bracketed with instances of a non-recursive syntactic pattern (*target pattern*). MBSL handles only POS tags, here an example input sentence for base-NP learning: [1]

$$[_{\texttt{NP}}\ \texttt{NNS}\ \texttt{NP}]\ \texttt{VB}\ [_{\texttt{NP}}\ \texttt{DT}\ \texttt{ADJ}\ \texttt{ADJ}\ \texttt{NN}\ \texttt{NNS}\ \texttt{NP}]\ .$$

This sentence contains two base NP's, as shown by the labelled brackets "$[_{\texttt{NP}}$" and "$\texttt{NP}]$".

The goal of MBSL is to find instances of the target pattern in a string of POS tags. By definition, target-pattern instances should be non-overlapping, and not included in each other. The instances are marked by bracketing the input POS string.

The alrogithm uses a *memory* built from the training corpus. MBSL determines the bracketing by first considering each subsequence of the sentence as a *candidate* to be a target instance (see Figure 1). For each candidate c, it computes a score $f_C(c)$ by searching the memory, keeping c if its score is above a threshold θ_C. The algorithm then finds a consistent bracketing for the input sentence, giving preference to subsequences whose score was high. In this section we will describe candidate scoring and selection in more detail.

[1] The POS tag set used throughout the paper is the Penn TreeBank set (Marcus, Santorini, and Marcinkiewicz, 1993): `DT` = determiner, `ADJ` = adjective, `RB` = adverb, `VB`=verb, `PP`=preposition, `NN` = singular noun, and `NNS` = plural noun.

1.3.1 Scoring candidates

A candidate is a sub-sequence of the input sentence, that the algorithm scores as an instance of the target pattern. The idea of the scoring algorithm is to reconstruct a candidate from prefixes and suffixes of pattern instances retrieved from the memory. Context information may be included as well. Candidate reconstruction is carried out by tiling it with prefixes and suffixes, with optional context. Therefore the POS sequences used for reconstruction are called *tiles*.

For example, when the target pattern is NP, suppose the training data contain sentences with the following POS:

1. [NP PRP NP] VBZ VBN IN [NP JJ NNS NP] .
2. [NP DT JJ NN NN NP] VBZ [NP DT NN NP] .
3. [NP DT JJ NN NNS NP] VB [NP DT JJ NN NP] .

and consider the task of finding NPs in a sentence with POS tags:

```
NN  VBZ  DT  JJ JJ  NN  .
1   2    3   4   5   6   7
```

Let words 3-6 be the candidate:

NN VBZ [NP DT JJ JJ NN NP] .

The sequence "[NP DT JJ JJ NN NP]" does not appear in the training data, but can be constructed from other training instances. The tile "[NPDT JJ" appears three times in the training data, as the POS sequence "DT JJ" appears three times as a NP prefix. That POS sequence does not appear elsewhere. We say that this tile has a *positive count* pos_count = 3 and its *negative count* is neg_count = 0. The tile "JJ NN NP]" appears once in Sentence 3, the POS sequence "JJ NN" appears two other times elsewhere. Therefore this tile has pos_count = 1 and neg_count = 2. The positive and negative counts of all the tiles can be obtained similarly, some of them are presented below:

		pos	neg
1.	VBZ [NP	1	1
2.	VBZ [NP DT	1	0
3.	[NP DT JJ	1	0
4.	JJ NN NP]	1	2
5.	JJ NN NP] .	1	0
6.	NN NP] .	2	2

The score $f_T(t)$ of a tile t is defined as the ratio of its positive and total counts (other functions could also be considered).

$$f_T(t) = \frac{\text{pos\_count}(t)}{\text{pos\_count}(t) + \text{neg\_count}(t)} \tag{1.1}$$

MBSL(*sentence, memory,* θ_T, θ_C)

1. Consider each subsequence of *sentence* as a candidate.
2. Evaluate $f_C(c)$ =**CandidateScore**(s, *memory*, θ_T);
3. If $f_C(c) > \theta_C$, add c to *candidate set*;
4. **SelectCandidates**(*candidate set*).

FIGURE 1 The top-level MBSL bracketing algorithm.

For a given threshold θ_T, only tiles satisfying $f_T(t) > \theta_T$ are accepted as evidence, they are called *matching tiles*.

A *cover* for a candidate and possibly some contextis a set of tiles such that:

- each word of the candidate, and possibly its context, is contained in at least one tile,
- no tile contains another tile entirely.

A cover constitutes evidence that the entire candidate is a target instance. In the current example, tiles 3 and 4 cover the candidate, while the other tiles add support by context evidence. Tile 5 differs from Tile 4 by the additional context of a dot at the end of the sentence. This addition is very important because Tile 5 has no negative count. In general, context information serves to provide more support when the evidence from internal POS alone may not be strong.

Although in principle the algorithm may work with unlimited context, in practice we only consider a fixed amount of maximal *left* and *right* contexts. Let the context limit be denoted by $ConLen$, the total number of tiles for a candidate of length l is $n_{\text{tiles}} = 2 \cdot ConLen \cdot (l+2) + 2 \cdot l + ConLen^2 + 1$. For a fixed maximal context, $n_{\text{tiles}} = \mathrm{O}(l)$, whereas for a fixed length the number of tiles grows as $ConLen^2$. A large $ConLen$ will therefore yield an unmanageable number of tiles. Current shallow-parsing results demonstrate that a context of size 2 or 3 provides enough useful information. Moreover, larger context may be much less effective due to sparseness. We have therefore used a context size of up to 3.

Computing the candidate score

The set of all covers for a candidate summarizes all of the evidence for the candidate as a target instance. The score of a candidate is therefore calculated from the set of all its covers. For example, if a candidate has many different covers, it is more likely to be a target instance, since many different pieces of evidence can be brought to bear.

We have empirically found several statistics of the cover set to be useful. These include, for each cover, the number of matches it contains,

the total number of context tags it contains, and the number of positions which more than one match covers (the amount of overlap). We thus compute, for the set of all covers of a candidate c:

- The total number of different covers, $\mathbf{num}(c)$,
- The least number of tiles constituting a cover, $\mathbf{minsize}(c)$,
- The maximum amount of total context in any cover (left plus right context), $\mathbf{maxcontext}(c)$, and
- The maximum over all covers of the total number of tile elements that overlap between connecting tiles, $\mathbf{maxoverlap}(c)$.

In ADK99 we presented the computation method in details. We show there that scoring a candidate of length l takes $O(l^2)$ steps in the worst case. This worst case is rarely reached in practice, because usually only a small fraction of the tiles are matching.

The score of the candidate is a linear function of its cover statistics:

$$f_C(c) = \alpha_1 \mathbf{num}(c) - \alpha_2 \mathbf{minsize}(c) + \alpha_3 \mathbf{maxcontext}(c) + \alpha_4 \mathbf{maxoverlap}(c) \tag{1.2}$$

If candidate c has no covers, $f_C(c) = 0$. Note that **minsize** is weighted negatively, since a cover with fewer tiles provides stronger evidence for the candidate.

In the current implementation, the weights were chosen so as to give a lexicographic ordering on the features, preferring first candidates with more covers, then those with covers containing fewer tiles, then those with larger covered contexts, and finally, when all else is equal, preferring candidates whose covers have more overlap between connecting tiles.

1.3.2 Selecting Candidates

The linguistic patterns for which the flat method is directed are non-overlapping. To apply this constraint we use a simple constraint propagation algorithm that finds the best choice of non-overlapping candidates in an input sentence, given in figure 2. Other methods for determining the preferred bracketing may also be applied; this remains a topic for future research.

1.3.3 Implementing the training memory

The scoring algorithm needs to search the training corpus for many subsequences of each input sentence in order to find matching tiles. Implementing this search efficiently is therefore of prime importance. We do so by encoding all the tiles of all target instances in the training corpus in a trie data structure.

A trie is a tree whose arcs are labeled with POS tags such that each path from the root represents a distinct tile in the training. A

SelectCandidates(*candidate-set*)

1. Examine each candidate $c \in$*candidate-set* such that $f_C(c) > 0$, in descending order of $f_C(c)$:
2. Add c's brackets to the sentence;
3. Remove all candidates overlapping with c from *candidate set*;
4. Return the candidates remaining in *candidate-set* as target instances.

FIGURE 2 Candidate selection algorithm.

trie allows searching for a sequence in time linear in the length of the sequence. Each node in the trie represents the sequence of arc labels along the path reaching it from the root. We store the positive and total (positive+negative) counts of each tile at its associated node in the trie.

1.4 Evaluation of the Flat Algorithm

1.4.1 The Data

We have tested our algorithm in recognizing three syntactic patterns: noun phrase sequences, verb-object, and subject-verb relations. The training and test data were derived from the Penn TreeBank (Marcus, Santorini, and Marcinkiewicz, 1993).

For NP, we used the data prepared by RM95, using WSJ Sections 15-18 as training data and Section 20 as test. These data sets were tagged by Brill's POS tagger (Brill, 1992). For the SV and VO data we used WSJ sections 01-09 as training and Section 00 as test set. The patterns were extracted using T (TreeBank's search script language) scripts [2]. A detailed description of the data and derivation of the SV and VO instances appears in ADK99.

1.4.2 Results

Table 1 summarizes the optimal parameter settings and results for NP, VO, and SV. In order to find the optimal values of the context size and threshold, we tried $0.1 \leq \theta_T \leq 0.95$, and maximum context sizes of 1,2, and 3. We used 5-fold cross-validation on the training data to determine the optimal parameter settings. In addition to recall and precision for each run, we used $F_\beta = \frac{(\beta^2+1)\cdot P \cdot R}{\beta^2 \cdot P + R}$, a common measure in information retrieval (van Rijsbergen, 1979), as a single-figure measure of merit. We use $\beta = 1$ which gives no preference to either recall or precision.

In experimenting with the maximum context size parameter, we

[2] available at `http://www.cs.biu.ac.il/~yuvalk/MBSL`

TABLE 1 Results with optimal parameter settings for context size and threshold, and breakeven points. The NP† line presents results obtained when removing adverbs and prepositions from the beginning of NPs (section 1.4.2. The last two lines show the results of RM95 (with and without lexical features) and Veenstra (1998, using a different POS tagger) recognizing NPs results with the same train/test data.

	Con.	θ_T	BE	Rec. (%)	Prec. (%)	$F_{\beta=1}$
VO	2	0.5	81.3	89.8	77.1	83.0
SV	3	0.6	86.1	84.5	88.6	86.5
NP	3	0.6	91.4	91.6	91.6	91.6
NP†	3	0.6		92.4	92.4	92.4
RM95 (NP)	-	-	-	90.7	90.5	90.6
+ Lex	-	-	-	92.3	91.8	92.0
Veenstra (NP)	-	-	-	94.3	89.0	91.6

found that the difference between the values of F_β for context sizes of 2 and 3 is less than half a percent for the optimal threshold. Scores for a context size of 1 yielded F_β values smaller by more than one percent than the values for the larger contexts.

The results of RM95 are shown in Table 1 for comparison, along with those of Veenstra (1998). All the cited results pertain to a training set of 229,000 words. Using a larger training set, of 950,000 words, RM95 attained a recall/precision of 93.5%/93.1% (F_β=93.3%); this last result was achieved including lexical information.

Two results from RM95 are presented: for learning with and without lexical information. Only the latter should be compared with our results, because it was obtained without using lexical information. All NP results are quite similar, up to recall/precision tradeoff. That may be related to limitations inherent in the particular choice of train/test data.

Some of the errors resulted from noun-phrases beginning with a preposition, or an adverb, such as:

```
[ IN About CD 20,000 NNS years ] RB ago

[ RB yet DT another NN setback ]
```

When we focus on nouns, preceding prepositions and adverbs are not of interest. We have made an experiment in which these words were taken out of NPs (and NPs composed of one word tagged `IN` or `RB` (e.g. `IN that`) were ignored). This experiment yielded better results, as presented in Table 1 at the line marked NP†.

The Penn TreeBank data contain trace markers, tagged as `-NONE-`.

These would not appear in a raw text from another source. In order to see the effect of the trace markers we have removed them and experimented with the optimal parameters. The results are summarized in Table 2. The more poor results indicate that the trace markers actually improved the inference quality, especially for the VO pattern.

Both RM95 and Veenstra approach the chunking task as a classification task, using chunk-tags. Both works use chunk-tag assignments of nearby (up to a distance of 2) words as features, and employ a post-processing step for assuring that the chunk-tags yield proper chunking (for example, correct cases where a chunk is opened within another chunk). Since chunk tags of preceding words were obtained using information about even earlier words - using them as features implies that information about preceding words at larger distances is taken into account. In these approaches there is an asymmetry between words to the left, and words to the right of the current word, as tagging is carried out from left to right.

Our method differs from these methods in that it considers sequences rather than individual words. Tiles of any length may contribute to the decision, with no preference to the beginning or ending of a pattern (in contrast with Cardie and Pierce (1998) who used only the prefixes of NPs). That makes the presented algorithm distinct from finite-state methods, which are directional in nature.

Indeed, we tried to learn a stochastic transducer at an earlier phase of the research, using the method of Ron, Singer, and Tishby (1995). We trained the transducer for detecting NPs in POS sequences, with the sentence presented at different offsets so the transducer could identify different NPs. That method, however, yielded poor results, possibly due to the difficulty of combining generalizations about both the beginning of an NP and its end.

TABLE 2 SV and VO results without trace markers, notice the degradation compared with the original data.

	Con.	Thresh.	Recall (%)	Precision (%)	$F_{\beta=1}$
VO	2	0.5	87.6	69.2	77.3
SV	3	0.6	84.0	87.5	85.7

1.5 The Compositional MBSL Algorithm

The compositional algorithm accepts, like the flat one, a POS string as input. It differs from the flat algorithm in two respects:

- It can handle *compositional* instances, thereby taking internal structure into account.
- It learns *multiple* target patterns simultaneously, therefore able to produce a partial parse consisting of instances of these patterns.

The memory trie is built in the same way as in the flat version (Sec. 1.3.3), and the overall inference process is essentially similar. Due to compositionality, tile generation is more complicated. It involves generating a *sentence graph* representing the phrase structure of the sentence (Sec. 1.5.2). In parsing, detected candidates are added to this graph. Another difference is that detecting candidates goes in order of increasing length (Sec. 1.5.3). Finally, the candidate scoring function takes cover statistics in a different order (Sec. 1.5.4).

1.5.1 An Illustrating Example

```
1.    [NP NNS NP] [VP RB VBZ [NP JJ NNS NP] VP] .
2.    [NP DT JJ NN NN NP] [VP VBZ [NP DT NN NP] VP] .
3.    [NP DT JJ JJ NN NP] [VP VBZ [NP DT JJ NN NP] VP] .
```

FIGURE 3 An example training data

Suppose the training data contains the sentences as in Fig. 3, and the sentence to be parsed is:

```
 NN  CC  NN  RB  VBZ  DT  JJ  NN  .
  1   2   3   4   5    6   7   8  9
```

and consider the task of finding VPs.

In the example above, let words 4-8 be a VP candidate. That POS sequence does not appear as a complete VP in training, but some of its tiles do appear and can provide supporting evidence.

Tiles in the flat version are comprised of POS sequences only. The compositional algorithm considers also embedded structures, hence a tile may include also pattern labels that stand for an embedded pattern instance (typically a phrase).

Some of the VP tiles used by the compositional algorithm are presented in Fig. 4. Tiles 1-5 are composed of POS only, and can be used by the flat version too. Among these tiles, Tiles 2 and 3 provide evidence for the candidate being a VP because together they cover it. Assuming the inner NP was already detected, tiles 6-8 reflect its presence within the composite VP. Tile 6 alone already provides a positive evidence for the entire VP, as it covers the whole candidate. The three negative appearances of Tile 8 are the three NP's at the start of the sentneces.

											pos	neg
1.		NN	[VP								2	3
2.			[VP	RB	VBZ						1	0
3.					VBZ	DT	JJ	NN	VP]		1	0
4.								NN	VP]		2	3
5.								NN	VP]	.	2	0
6.			[VP	RB	VBZ		NP		VP]		1	0
7.					VBZ		NP		VP]		3	0
8.							NP		VP]		3	3

FIGURE 4 Some of the VP tiles derived from the data in Figure 3

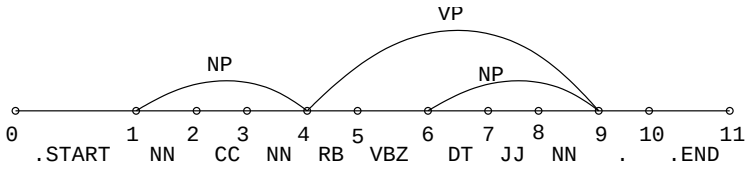

FIGURE 5 A sample sentence graph with context symbols, the target patterns are NP and VP

Note that the flat version had to rely on Tile 3, originally from Sentence 3, in order to collect supporting evidence that covers the whole candidate. The compositional algorithm could produce the evidence without this sentence, by realizing the compositional evidence.

1.5.2 Data Structures and Definitions

Each sentence is represented by a *sentence graph*: the nodes of the graph correspond to positions between the sentence words, and the edges correspond to either POS tags or pattern instances. The edges are *directed* from the beginning of the sentence to its end. Figure 5 shows a sentence graph for the example given above.

We now define the tiles of a pattern instance in terms of the sentence graph. Denote by $ConLen$ the maximal length considered for the preceding and following contexts of the given instance, and let i and j be the starting and ending positions of the instance (e.g., 4 and 9 for the VP in Figure 5). In these terms, a tile t is a path in the sentence graph that satisfies the following conditions:

1. t is embedded within the range of nodes $i-ConLen$ to $j+ConLen$; that is, within the pattern instance and its context.
2. t includes at least one of the instance boundary nodes i, j, which correspond to the boundary brackets (e.g., [NP, NP]).
3. An edge of t that corresponds to a pattern instance (rather than to

a POS) must be fully embedded within the range of the instance itself (nodes i through j). This requirement does not necessarily hold when the graph is created during parsing, as some candidates may be crossing each other.

The equivalent of a sentence graph in the flat version is a graph with the same nodes but only the POS edges.

1.5.3 Algorithm Outline

The partial parsing algorithm receives an input sentence, represented by a POS sequence, and labels of the target patterns. It first constructs a sentence graph consisting of only POS edges and then performs the following two steps:

1. **Scoring candidate instances:** The algorithm considers each subsequence of the sentence as a candidate for each of the target pattern types and assigns a candidate score. Candidates with a positive score are added to the sentence graph.
   Candidates are considered in order of increasing length. This way, when scoring a certain candidate, shorter pattern instances that might be considered as embedded patterns are already represented in the sentence graph, and contribute to tiles and scoring.
   The parsing process is anaolgous to chart parsing, where the sentence graph serves as the chart, storing structures found so far.
2. **Resolving candidate conflicts:** After the first step, some of the candidate instances may be conflicting with each other (crossing edges in the graph). This step resolves such conflicts using the simple constraint propagation scheme used by the flat MBSL method: candidates are approved in order of decreasing score, while eliminating all candidates that conflict with previously approved ones.

1.5.4 Computing Candidate Score

Given a hypothesized candidate for a certain pattern type, spanning between positions i and j, the algorithm searches for all tiles of the candidate in the memory trie. This is done by traversing the sentence graph from each possible starting position for the candidate tiles, using a DFS-like search. Whenever an edge is traversed in the sentence graph, the corresponding edge, if available, is traversed also in the trie to fetch the counts of the corresponding tile. The tile score is calculated by Eq. (1.1) as in the flat algorithm, and only tiles with $f_T(t) > \theta_T$ are accepted as supporting evidence.

Once all supporting tiles are found, the algorithm tries to use them for covering the entire candidate. Often, a number of covers can be cre-

ated from a set of tiles, each may contain overlapping tiles or tiles which cover different parts of the context. The scoring function uses the following quantities for a given candidate c:

- The ratio of grand-total positive to grand-total positive+negative counts of tiles in all the covers, **totalratio**(c),
- The total number of different covers, **num**(c),
- The length of the shortest cover, **minsize**(c),
- The maximum amount of total context in any cover (left plus right context),
  maxcontext(c), and
- The maximum over all covers of the total number of tile elements that overlap between connecting tiles, **maxoverlap**(c).

The score of the candidate is a linear function of its cover statistics:

$$\begin{aligned} f_C(c) \quad = \quad & \alpha_1 \, \mathbf{totalratio}(c) + \alpha_2 \, \mathbf{num}(c) \\ & -\alpha_3 \, \mathbf{minsize}(c) \\ & +\alpha_4 \, \mathbf{maxcontext}(c) \\ & +\alpha_5 \, \mathbf{maxoverlap}(c) \end{aligned}$$

If candidate c has no covers, $f_C(c) = 0$. The weights in the current implementation were chosen so as to give a lexicographic ordering on the features, preferring first candidates with a higher grand-total ratio, then according to the following order: candidates with more covers, with covers containing fewer tiles, with larger covered contexts, and when all else is equal, candidates whose covers have more overlap between connecting tiles.

The flat version used a similar function without using **totalratio**, hence **num** was the most important quantity. In the composite case, inner instances increase the number of possible covers to the extent that it no longer becomes a good measure of reliability (at least not at face value).

1.6 Evaluation of the Compositional Algorithm

The compositional system was trained on the Penn Treebank WSJ Sections 02-21 and tested on Section 23 (Table 3), same as used by Magerman (1995), Collins (1997), and Ratnaparkhi (1997), and became a common testbed.

The tasks were selected so as to demonstrate the benefit of using internal structure data for learning composite structures. We have studied the effect of noun-phrase information on learning verb phrases by setting limits on the number of embedded instances, n_{emb} in a tile. A limit of zero emulates the flat version since learning takes place from POS tags

TABLE 3 Data set sizes, note the similar proportions of base instances

	Training Data: WSJ 02-21, 28884 sentences			Test Data: WSJ 23, 2416 sentences		
	base	composite	base:all	base	composite	base:all
NP	166242	61384	73%	13524	5106	73%
VP	43377	28017	61%	3496	2267	61%

only. The final output, however, may include embedded instances since instances may be composite. Results indicate that $n_{\text{emb}} > 0$ allows for NP information to contribute to VP learning.

A minimal context of one word and up to two words on each side was used, and tile threshold was set to $\theta_T = 0.6$ following results of the flat version. A minimal context was not necessary in the flat version, but here the additional evidence from embedded instances gives rise to more false detections - a phenomenon compensated by setting a minimal context. The maximal tile length was set to 5; higher values gave a very small improvement which did not justify the additional memory.

Table 4 presents results for simultaneous NP and VP learning, and for learning VP without NP. For $n_{\text{emb}} = 0$, NPs do not contribute to inference; the small performance difference results from NP influence on conflict resolution. The effect of NPs is clearly visible when $n_{\text{emb}} = 1$ is allowed, yielding 24% more recall and an improvement of 10% in F_β.

For NPs only (Table 5), allowing embedded instances improved the recall in expense of precision. We have experimented with seperating NPs from base-NPs [3]. As Table 5 shows, seperating the tasks improved the overall precision for NPs. When $n_{\text{emb}} = 0$, there was a 2.5% recall decrease, whereas when $n_{\text{emb}} = 1$, the recall decrease was only 0.4%. The effect of modeling the internal NP structure ($n_{\text{emb}} = 1$) clearly shows for composite NP's. The table also shows that VP information in did not have a significant impact on NP learning.

There are currently no other partial parsers on these tasks to compare the combined VP and NP results to. Tjong Kim Sang (1999) presented result for composite NP, obtained by repeated cascading, similar to our results with seperate base and composite NPs and no internal structure.

1.7 Discussion

We have presented a memory-based learning method for sequences. The method can learn non-recursive chunk structures, as well as exploit compositional information and producde a partial parse. Like other shallow-

[3] when scoring non-recursive patterns, **num** was the leading quantity and **total-ratio** the second, based on flat-version experience

TABLE 4 VP Results, $\theta_T = 0.6$, tile length≤ 5

	max. n_{emb}	rec.	prec.	F_β
VP only	0	47.1	76.2	58.2
VP only	1	58.9	62.8	60.7
VP with NP	0	45.4	77.4	57.2
VP with NP	1	82.6	61.5	70.5

TABLE 5 NP Results, $\theta_T = 0.6$, tile length≤ 5. Rows 5–10 refer to experiments where base-NPs were distinguished from composite ones.

	max. n_{emb}	rec.	prec.	F_β
NP only	0	81.8	76.8	79.2
	1	87.6	65.8	75.2
NP with VP	0	81.7	77.1	79.3
	1	86.0	66.0	74.7
base NP	0	93.4	93.6	93.5
composite		42.0	67.1	51.7
all NP		79.3	88.5	83.7
base NP	1	93.2	93.5	93.3
composite		71.4	49.0	58.1
all NP		87.2	77.7	82.2
NP (TKS99)		76.1	91.3	83.0

parsing systems, it is most useful when the number of target patterns is small. In particular, the method does not require fully-parsed sentences as training, unlike trainable full parsing methods.

The training material has to contain only bracketing of the target patterns, implying much simpler training material when the parsing task is limited. This and similar methods are, accordingly, attractive in cases where a fully-parsed corpus is not available for training, or when a full parse is not necessary for handling the problem.

Scha, Bod, and Sima'an (1999) provide a thorough comparison of the flat MBSL method with DOP. Considering POS data only, the compositional MBSL method resembles the DOP1 model in that it uses subconstituent information in order to construct evidence for higher-level structures. There are, however, some differences:

- Consider the phrase "`[VP A [NP B C NP] VP]`". In DOP, `B` and `C`

will be leaves in the tree whose root is NP, and will contribute to VP via that inner NP. In MBSL, these tags will participate in NP tiles as well as in VP tiles. That is, VP would be considered as comprised of "[A NP]" as well as "[A B C]". In general, MBSL relies on local information even for high-level structures. In DOP, all hierarchical structures has to be composed from sub-structures.

- Even supposing B and C do not contribute to evidence for VP, the MBSL score of VP will be calculated with NP regarded as a terminal. That is, the internal structure and score of NP is not taken into account. In DOP, the probability of the VP node would be calculated from that of its constituents.
- The scoring process of DOP is based on a probabilistic tree substitution model. Flat MBSL evaluates the evidence quality using the *combinatorial* property of the number of covers, as well as tile properties like context, overlap, and shortest cover available. With this scheme, Flat MBSL does not aim to estimate any probabilities or rely on a probabilistic model. It would be interesting to explore in future research alternative scoring mechanisms for MBSL that are based on probabilistic models.
- During MBSL training, it is possible to specify if tiles of a composite instance will be created bottom-up or top-down, and limit the nesting level. This can be useful in highly-nested instances, where going all the way top-down will create a lot of tiles.

A property both methods share is that they will work harder when there is plenty of evidence for a candidate (Sima'an, p.c.). This contradicts the intuition that more "straightforward" candidates would be identified faster. We plan to tackle this problem in the future.

Another related algorithm was presented by Sekine and Grishman (1995, hereafter APP) and Sekine (1998). The algorithm extracts grammar rules with S and NP (and possibly other structures) as non-terminals. Both APP and MBSL use raw-data examples for parsing, and can be restricted to specified target non-terminals or patterns. The differences lie in:

- MBSL recombines fragments of instances for generalizations, while APP uses rules derived from complete instances.
- The grammar rules of APP do not include context, which is taken into account only when generating the non-terminal S. In MBSL, the context can add evidence for each instance candidate.
- APP, like DOP, uses a probabilistic model. The probability of a grammar rule Y $\rightarrow$ X is Freq(Y $\rightarrow$ X)/Freq(X). Analogously, the denominator for a tile score in MBSL is Freq(Y).

MBSL differs from these two hierarchical methods as it does not try to produce a global phrase structure, but only to find pattern instances, detecting each instant independently.

The compositional method concerns primarily with hierarchical phrase structure. It is not aimed at handling dependencies, which require heavy use of lexical information (Hindle and Rooth, 1993, for PP attachment). As (Daelemans, Buchholz, and Veenstra, 1999) show, lexical information improves on NP and VP chunking as well. Since our method uses raw data, representing lexical entries will require a lot of memory.

In a future work, we plan to use the system for learning functional head-modifier relations, and disambiguate them using an algorithm more suitable for handling lexical information. An additional possibility is to use word-types, such as a special tag for be-verbs, or for a preposition like "of" which attaches mainly to nouns (Sekine and Grishman, 1995). In determining the final parse, the MBSL system would be able to incorporate information from relations as well as phrases.

In a similar vain to Skut and Brants (1998b) and Buchholz, Veenstra, and Daelemans (1999), the method extends an existing flat shallow-parsing method to handle composite structures. Nevertheless, the simplicity of the flat method is retained, and only local information is being used. It yields a significant improvement over the flat method, especially for long and more complex structures.

As can be expected, the performance of the partial method is still lower than that of full parsers, which exploit (and require) much richer information. The results of this line of research enrich the space of alternative parsing approaches, aiming to reduce the gap between shallow and full parsing.

References

Abney, S. P. 1991. Parsing by chunks. In R. C. Berwick, S. P. Abney, and C. Tenny, editors, *Principle-Based Parsing: Computation and Psycholinguistics*. Kluwer, Dordrecht, pages 257–278.

Abney, S. P. 1996. Part-of-speech tagging and partial parsing. In K. Church, S. Young, and G. Bloothooft, editors, *Corpus-Based Methods in Language and Speech*. Kluwer, Dordrecht. An ELSNET book.

Argamon, S., I. Dagan, and Y. Krymolowski. 1998. A memory-based approach to learning shallow natural language patterns. In *Proc. of COLING/ACL*, pages 67–73, Montreal, Canada.

Argamon, S., I. Dagan, and Y. Krymolowski. 1999. A memory-based approach to learning shallow natural language patterns. *Journal of*

Experimental and Theoretical AI, 11:369–390. CMP-LG/9806011.

Brill, E. 1992. A simple rule-based part of speech tagger. In *proc. of the DARPA Workshop on Speech and Natural Language.*

Brill, Eric. 1993. Automatic grammar induction and parsing free text: A transformation-based approach. In *Proceedings of the 31 th ACL*, pages 259–265, Unknown.

Buchholz, S., J. Veenstra, and W. Daelemans. 1999. Cascaded grammatical relation assignment. In *Proeedings of EMNLP/VLC-99*, pages 239–246, University of Maryland, USA, June.

Cardie, C. and D. Pierce. 1998. Error-driven pruning of treebank grammars for base noun phrase identification. In *Proc. of COLING/ACL*, pages 218–224, Montreal, Canada.

Church, Kenneth W. 1988. A stochastic parts program and noun phrase parser for unrestricted text. In *proc. of ACL Conference on Applied Natural Language Processing.*

Collins, M. 1997. Three generative, lexicalised models for statistical parsing. In *Proc. of the ACL/EACL Annual Meeting*, pages 16–23, Madrid, Spain, July.

Daelemans, Walter, Sabine Buchholz, and Jorn Veenstra. 1999. Memory-based shallow parsing. In *Proeedings of CoNLL-99*, Bergen, Norway, June.

Greffenstette, Gregory. 1993. Evaluation techniques for automatic semantic extraction: Comparing syntactic and window based approaches. In *ACL Workshop on Acquisition of Lexical Knowledge From Text*, Ohio State University, June.

Hindle, D. and M. Rooth. 1993. Structural ambiguity and lexical relations. *Computational Linguistics*, 19(1):103–120.

Krymolowski, Y. and I. Dagan. 2000. Incorporating compositional evidence in memory-based partial parsing. In *Proc. of the 38th Annual Meeting of the ACL*, pages 45–52, Hong Kong.

Magerman, David M. 1995. Statistical decision-tree models for parsing. In *Proc. of the 33rd Annual Meeting of the Association for Computational Linguistics. Cambridge, MA, 26–30.*

Marcus, M. P., B. Santorini, and M. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, June.

Muñoz, M., V. Punyakanok, D. Roth, and D. Zimak. 1999. A learning approach to shallow parsing. In *EMNLP-VLC'99, the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 168–178, June.

Ramshaw, L. A. and M. P. Marcus. 1995. Text chunking using transformation-based learning. In *Proceedings of the Third Workshop on Very Large Corpora*.

Ratnaparkhi, A. 1997. A linear observed time statistical parser based on maximum entropy models. In *EMNLP2*, Providence, RI, March.

Ron, D., Y. Singer, and N. Tishby. 1995. On the learnability and usage of acyclic probabilistic finite automata. In *Proceedings of the 8th Annual Conference on Computational Learning Theory (COLT'95)*, pages 31–40, New York, NY, USA, July. ACM Press.

Scha, Remko, Rens Bod, and Khalil Sima'an. 1999. A memory-based model of syntactic analysis: Data-oriented parsing. *Journal of Experimental and Theoretical AI*, 11:409–440.

Sekine, Satoshi. 1998. *Corpus-Based Parsing and Sublanguage Studies*. Ph.D. thesis, New York University.

Sekine, Satoshi and Ralph Grishman. 1995. A corpus-based probabilistic grammar with only two non-terminals. In *Fourth International Workshop on Parsing Technology – IWPT*, Prague.

Skut, W. and T. Brants. 1998a. Chunk tagger, statistical recognition of noun phrases. In *ESSLLI Workshop on Automated Acquisition of Syntax and Parsing*, Saarbrücken, Germany.

Skut, W. and T. Brants. 1998b. A maximum-entropy partial parser for unrestricted text. In *Proc. of the sixth Workshop on Very Large Corpora*, Montreal, Canada.

Tjong Kim Sang, E. F. 1999. Noun phrase detection by repeated chunking. In *Proeedings of CoNLL-99*, Bergen, Norway, June.

Tjong Kim Sang, Erik. F. and Sabine Buchholz. 2000. Introduction to the conll-2000 shared task: Chunking. In *Proceedings of CoNLL-2000 and LLL-2000*, pages 127–132, Lisbon, Portugal.

van Rijsbergen, C. J. 1979. *Information Retrieval*. Buttersworth.

Veenstra, J. 1998. Fast NP chunking using memory-based learning techniques. In F. Verdenius and W. van den Broek, editors, *Proceedings of Benelearn*, pages 71–79, Wageningen, the Netherlands.