

# A Memory-Based Approach to Learning Shallow Natural Language Patterns

Shlomo Argamon-Engelson      Ido Dagan  
Yuval Krymolowski  
Department of Mathematics and Computer Science  
Bar-Ilan University  
52900 Ramat Gan, Israel  
{argamon,dagan,yuvalk}@cs.biu.ac.il

May 23, 1999

## **Abstract**

Recognizing shallow linguistic patterns, such as basic syntactic relationships between words, is a common task in applied natural language and text processing. The common practice for approaching this task is by tedious manual definition of possible pattern structures, often in the form of regular expressions or finite automata. This paper presents a novel memory-based learning method that recognizes shallow patterns in new text based on a bracketed training corpus. The examples are stored as-is, in efficient data structures. Generalization is performed on-line at recognition time by comparing subsequences of the new text to positive and negative evidence in the corpus. This way, no information in the training is lost, as can happen in other learning systems that construct a single generalized model at the time of training. The paper presents experimental results for recognizing noun phrase, subject-verb and verb-object patterns in English.

# 1 Introduction

Identifying local patterns of syntactic sequences and relationships is a fundamental task in natural language processing (NLP). Such patterns may correspond to syntactic phrases, like noun phrases, or to pairs of words that participate in a syntactic relationship, like the heads of a verb-object relation. Such patterns have been found useful in various application areas, including information extraction, text summarization, and bilingual alignment. Syntactic patterns are useful also for many basic computational linguistic tasks, such as statistical word similarity and various disambiguation problems.

One approach for detecting syntactic patterns is to obtain a full parse of a sentence and then extract the required patterns. However, obtaining a complete parse tree for a sentence is difficult in many cases, and may not be necessary at all for identifying most instances of local syntactic patterns.

An alternative approach is to avoid the complexity of full parsing and instead analyse a sentence at the level of phrases and the relations between them, this is the task of *shallow parsing* (Abney, 1991; Greffentette, 1993). In contrast to full parsing, shallow parsing can be achieved using *local* information. Previous works (cf. section 2.3) have shown that it is possible to identify most instances of shallow syntactic patterns by rules that examine only the pattern itself and its nearby context. Often, the rules are applied to sentences that were tagged by part-of-speech (POS) and are phrased by some form of regular expressions or finite state automata.

Manual writing of local syntactic rules has become a common practice for many applications. However, writing rules is often tedious and time consuming. Furthermore, extending the rules to different languages or sub-language domains can require substantial resources and expertise that are often not available. As in many areas of NLP, a *learning* approach is appealing.

Abney (1991) introduced the notion of *chunks*, denoting sequences of words with a certain syntactic function (more in section 2.1). The task of dividing a sentence into meaningful sequences of words is thus referred to as *chunking*. The most studied chunking task is that of identifying noun phrases (NPs, e.g. Church (1988), Ramshaw and Marcus (1995), Cardie and Pierce (1998), Veenstra (1998)), naturally due to their central role in a sentence.

This paper presents a novel general learning approach for recognizing local sequential patterns, that falls within the memory-based learning paradigm. The method learns from a POS tagged training corpus in which all instances of the target pattern are marked (bracketed). Subsequences of the training examples are stored as-is in a trie, thereby facilitating a linear-time search for subsequences in the corpus. While the presented algorithm is oriented at recognizing sequences, it is useful also for grammatical *relations* such as subject-verb, and verb-object. We present these applications, in which the relations are represented as sequences encompassing the relevant words. The algorithm is therefore suitable for learning to perform tasks involved in shallow parsing.

The memory-based nature of the presented algorithm stems from its induction strategy: a sequence is recognized as an instance of the target pattern by examining the *raw* training corpus, searching for relevant positive and negative evidence. No model is created for storing the training corpus, and the raw data are not converted to any other representation.

Modelling lies in the choice of the kinds of data retrieved from the stored corpus, that is, in the substrings being searched in the memory. We believe this choice to be highly relevant for linguistic data.

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 NNP = plural noun. As an illustrating example, Suppose we want to decide whether the candidate sequence

DT ADJ ADJ NN NNP

is an NP using information from the training corpus. Finding the entire sequence as-is several times in the corpus would yield an exact match. Due to data sparseness, however, that cannot always be expected.

A somewhat weaker match may be obtained if we consider *tiles*, sub-parts of the candidate sequence. For example, suppose the corpus contains noun phrases with the following structures:

- (1) DT ADJ ADJ NN NN
- (2) DT ADJ NN NNP

The first structure provides positive evidence that the sequence ‘DT ADJ ADJ NN’ is a possible NP prefix while the second structure provides evidence for ‘ADJ NN NNP’ being an NP suffix. Together, these two training instances provide positive evidence that *covers* the entire candidate. Considering evidence for sub-parts of the pattern enables us to generalize over the exact structures present in the corpus. Similarly, the algorithm considers the negative evidence for such sub-parts by noting where they occur in the corpus without being a corresponding part of a target-pattern instance. Surrounding context and evidence overlap are considered as well.

Other implementations of the memory-based paradigm for NLP tasks include Daelemans et al. (1996), for POS tagging; Cardie (1993), for syntactic and semantic tagging; and Stanfill and Waltz (1986), for word pronunciation. In all these works, examples are represented as sets of features and the induction is carried out by finding the most similar cases. The memory-based works of Bod (1992, DOP) for parsing, and Yvon (1996) for pronunciation use the *raw* form of the data, rather than encode it as features. The method presented here is similar in that it makes use of *raw sequential* data, and generalizes by reconstructing test examples from different pieces of the training data.

Previous related work is described in section 2. Section 3 describes the inference algorithm formally; section 4 presents experimental results for three target syntactic patterns in English, along with comparison with related results.

## 2 Background

We present here a brief overview of the state-of-the-art in shallow parsing, covering both hand-crafted and learnable parsers. In addition, since shallow parsing can be viewed as a sub-task of full parsing, we also discuss relevant methods for learning full parsing. Section 2.1

presents some of the current hand-crafted shallow parsers, while sections 2.2 and 2.3 present methods for learning full and shallow parsing respectively.

## 2.1 Shallow Parsers with Hand-Written Rules

Much work to date on shallow parsing has been based on hand-crafted sets of rules, generally using a probabilistic model to choose between parse alternatives.

Abney (1991, 1996) has pioneered work on shallow parsing. His work was motivated by psycholinguistic evidence (Gee and Grosjean, 1983), indicating that language processing involves dividing the sentence into *performance structures* — corresponding to pauses and intonation changes during speech. Abney introduced the concept of *chunks*, defined as consisting of ‘a single content word surrounded by a constellation of function words, matching a fixed template’. His chunk parser operates in two phases: a *chunker* which offers potential chunks, and an *attacher* which resolves attachment ambiguities and selects the final chunks. The chunker makes use of POS data, whereas the attacher requires lexical information. That way, lexical information is used only for the tasks for which it is more important, and simpler POS information is used for the more basic task of chunking. Both parts of Abney’s chunk parser are implemented as non-deterministic LR parsers. The distinction between chunking and attachment is common to all the systems presented in this section.

Aït-Mokhtar and Chanod (1997) presented a sequence of finite-state transducers for extracting subjects and objects from POS-tagged French texts. Extraction is incremental, each processing phase is carried out by a dedicated transducer, which prepares the input for the next phase. The system relies only on POS information, that is, it does not require lexical information. For various corpora, the recall and precision for subjects were above 90.5% and 86.5% respectively, whereas for objects these figures exceeded 84.4% and 79.9% .

Much shallow parsing effort has been motivated by information-extraction (IE) tasks such as MUC. For example, the FASTUS (Appelt et al., 1993) system uses cascaded, non-deterministic finite-state automata for extracting noun groups, verb groups, and particles. As an IE system, FASTUS is built for cases where only part of the text is relevant, and the target patterns can be represented in a simple rigid fashion.

Schiller (1996) presented a finite-state multilingual system for NP detection. This system is also intended for simple phrases, without coordinations or relative clauses.

The SPARKLE (Shallow PARsing for acquisition of Knowledge for Language Engineering, <http://www.ilc.pi.cnr.it/sparkle/sparkle.htm>) project aims mainly in developing ‘robust and portable tools leading to commercial applications devoted to the management of multilingual information in electronic form’. Phrasal-level syntactic analysis is essential for the objective of the project; the resulting parses will serve for acquiring lexical information. Part of the SPARKLE project is thus devoted to developing shallow parsers.

- The shallow parser for English is developed in Cambridge and Sussex universities. Parsing is carried out by a generalized LR parser, which uses a unification-based phrasal grammar of POS tags. The parser performs disambiguation based on a probabilistic model, that is, the rules are fixed, but their probabilities are being learned. The reported recall and precision are 82.5%/83% for phrases (without lexical information) and 88.1%/88.2% for grammatical relations (including lexical information).

- In the system for German, developed at the University of Stuttgart, the grammar rules are written in a bottom-up fashion. The parser is a standard chart parser, extended to include head-markings for lexicalization. The parser is also enhanced to carry out EM estimation of hand-written CFG rule probabilities. That way, both parsing and lexical acquisition are integrated in a single process. Final evaluation data were not available.
- The system developed for French, at Rank-Xerox Research Centre (RXRC) is a collection of finite-state transducers. The input is a POS-tagged text, and each transducer inserts markup symbols for the corresponding pattern. Parsing is carried out by invoking the transducers bottom-up, starting from the easier tasks, using the transducer output as an input to the next one. Once the text is marked, another transducer identifies the head words. That information is used in the last phase, by a transducer which identifies syntactic functions.

## 2.2 Learning: Full Parses

Full parsing learning methods are applicable in general to shallow parsing, where they can be used for extracting partial parses.

Some methods aim at estimating probabilities of hand-written grammar rules, based on annotated corpora. These include the works of Pereira and Schabes (1992) for CFG, Chitrao and Grishman (1990) for context-sensitive grammar, Eisner (1996) for dependency parsing, Magerman and Marcus (1991) for bottom-up chart parsing, and Briscoe and Carroll (1993) for attribute-value grammars. In the rest of the section we present works which do not use a certain set of grammar rules, but induce the parsing from statistical features extracted from the training data.

Brill (1993) used transformation-based-learning for learning to produce an unlabeled parse tree. Given a POS-tagged sentence, the learned transformations manipulated insertion and deletion of left and right parenthesis depending on the current POS tag and, possibly, the POS tag of the previous or next word.

SPATTER (Magerman, 1995), is a learning algorithm for parsing which makes use only of hierarchical structure information. It learns a decision-tree model for tagging and parsing, where parsing decisions build a hierarchical structure in a bottom-up manner. The questions at the nodes of the tree take into account neighboring as well as child nodes; the result is a complete parse tree. It scores 84% recall and 84.3% precision on Penn TreeBank WSJ data.

The parser of Collins (1997) is based on statistics of lexical dependencies, subcategorization and wh-movement. His parser scored 88.1% recall and 87.5% precision on Penn TreeBank WSJ data, currently the best result on that dataset.

Recently, Sekine (1998) built a system based on learning parse rules for five non-terminals: sentence, subordinate sentence, infinitive sentence, NP, and base NP; the system also uses lexical dependency information. In addition his system is capable of performing ‘fitted parsing’ — combining partial trees or chunks built by these rules, producing a complete parse. With combined chunks restricted to S nodes or punctuation marks, less than 1% of the test sentences required fitting. 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, best-first search and Viterbi search were used for optimization.

The maximum-entropy parser by Ratnaparkhi (1997), operates in three passes: tagging, chunking, and building hierarchical structure. A maximum-entropy model is employed at each phase. The features which the models test are ‘contextual predicates’ - word, POS or chunk-tag of neighboring (up to a distance of 2) words, or features which represent hierarchical dependency. Each pass prepares information for the next one. The performance of the system on the data on which SPATTER was tested is 86.3% recall and 87.5% precision.

Data Oriented Parsing (DOP), presented by Bod (1992), is another example of an algorithm based purely on hierarchical structure information. It relies on the idea of producing a parse-tree by combining sub-trees from a memory of previous parses. For example, suppose the training data contained the parsed sentences:

```
(S (NP (DT The) (JJ pretty) (NN bird)) (VP (VB sings)))  
(S (NP (DT The) (NN airplane)) (VP (VB flies)))
```

The system memory would contain all the sub-trees of these parse trees; some of them are:

```
(S (NP) (VP))  
(VP (VB))  
(NP (DT) (NN))  
(NP (DT) (JJ) (NN))  
(DT the)  
(NN bird)  
(VB flies)
```

Given the new sentence: ‘The bird flies’, the parse tree

```
(S (NP (DT the) (NN bird)) (VP (VB flies)))
```

can now be built by combining the sub-trees from the memory.

In the general case, each sub-tree is scored according to its frequency relative to other sub-trees with the same root. Then, the various parse alternatives are scored according to the scores of their building blocks. Finding the best parse is NP-hard (Sima’an, 1996), therefore the DOP parsing algorithm uses a Monte-Carlo approximation and the resulting parse is an estimation of the best one.

Note that DOP does not rely on a given set of parsing rules (nor does it build such a set); instead, it reads the parsing trees as raw data and uses them as building blocks. The algorithm presented in this paper is similar to DOP in that it tries to build evidence for bracketing an input sequence based on matching subsequences of the target pattern stored in the memory.

## 2.3 Learning: Shallow Parses

A number of systems have been developed for learning to perform shallow parsing. Most of these systems learn to identify chunks such as noun phrases, while some others learn to identify relationships between chunks (or representative words thereof).

Church (1988) uses a simple model for finding base (non-recursive) NPs in a sequence of POS tags. Viewing the task as a bracketing problem, he calculated the probability of inserting open/close brackets between POS tags. A sequence of POS tags, representing a sentence, may be chunked into base NPs in various ways. Each chunking alternative is scored using the probabilities, the best alternative is then chosen. The algorithm was tested on the Brown corpus; while exact results are not reported, they are described as encouraging. In particular, that motivates using POS-tag information alone for NP detection. That is important because the few dozens of POS tags require much less resources than any collection of lexical information.

Ramshaw and Marcus (1995, hereafter RM95) viewed the problem as classification of words. Each word was assigned a chunk-tag: ‘I’ or ‘O’ for words inside or outside a chunk, and ‘B’ for words which stand at the beginning of a base NP that immediately follows another base NP. Thus, in contrast to (Church, 1988), where the sentence was chunked based on a global criterion (the chunking with the higher score), here the decision is *local*. RM95 used transformation-based learning (Brill, 1992), with rule-templates referring to neighboring words, POS tags, and chunk tags (up to a distance of 3 for words or POS tags, and 2 for chunk tags). Their work is the first one to present large-scale testing of base NP learning. Training on 950K words from the Penn TreeBank WSJ data tagged by Brill’s POS tagger (Brill, 1992), with a test set of 50K words, they achieved a recall of 93.5% and precision of 93.1%. RM95 demonstrate that the contribution of the lexical information is about 1%, the main information sources were thus the neighboring POS and chunk tags. They also report a chunk-tagging accuracy of 97.8%.

Veenstra (1998) recently presented an application of the IGTREE (Daelemans, van den Bosch, and Weijters, 1996) memory-based learning algorithm for NP chunking. Using the data of RM95, POS tagged by Memory-Based Tagger (Daelemans et al., 1996), he also assigned a chunk tag (‘I’, ‘O’, ‘B’) to each word. The features included lexical and POS information at a distance of up to two words. Some variants have generally yielded higher recall and lower precision than RM95, see table 4 for details. While both works used information about the near context as features, a further correction phase was required in order to make sure that the chunk tagging yields a proper chunking (e.g. a ‘B’ tag cannot follow an ‘O’ tag). That phase essentially makes use of information which may result from decisions made for quite far words.

Another memory-based approach was presented by Cardie and Pierce (1998). They created a set of grammar rules from the NP training inventory, and pruned it using a separate corpus. In the inference phase, the longest matching rule was applied. The system accepts POS-tag strings, that is, it does not handle lexical information. Direct comparison with the results of RM95 was not presented, but cross-validation results on NPs created in a similar fashion yielded 91.1% recall and 90.7% precision – similar to what RM95 obtained ignoring lexical data. Cardie and Pierce have used a pruning methodology which discards the ten worst rules until precision drops, as well as local repair heuristics which improved the precision by 1% without harming the recall. Their method works better, as they note, on simpler NPs.

Skut and Brants (1998) worked also at the word-level, but with features which include hierarchical structure information as well. They used the maximum-entropy method for

learning to assign words with a triple structural tag: a POS tag, syntactic category, and a tag which represents the relation between dominating nodes of the current and the preceding words. The triple nature of the features makes it possible to use only some of the three tags - thereby obtaining a more general and, possibly, meaningful feature. Weights of the structural features are evaluated using the maximum-entropy method, then incorporated into a trigram model. The reported results for NP and PP chunking of a German text are 88.9% recall and 87.6% precision.

### 3 The Algorithm

The input to the Memory-Based Sequence Learning (MBSL) algorithm is a sentence represented as a sequence of POS tags, and its output is a *bracketed sentence*, indicating which subsequences of the sentence are to be considered instances of the target pattern (*target instances*). The training corpus consists of pre-bracketed sentences, it is used for creating the memory. MBSL (see figure 1) determines the bracketing by first considering each subsequence of the sentence as a *candidate* to be a target instance. For each candidate  $c$ , it computes a likelihood score  $f_C(c)$  by searching the memory, keeping  $c$  if its score is above a threshold  $\theta_C$ . The algorithm then finds a consistent bracketing for the input sentence, giving preference to subsequences whose score was high. In the remainder of this section we will describe the components of the algorithm in more detail.

insert figure 1 here

#### 3.1 Scoring candidates

We first describe the mechanism for scoring candidates. The input is a candidate subsequence, along with its *context*, i.e. surrounding tags in the input sentence. We derive a score for the candidate by searching substrings of the candidate and its context in the training corpus.

Consider first the case when the candidate subsequence as a whole occurs in the training corpus as an actual target instance. For example, consider the candidate ADJ NN NN, where the training corpus contains the bracketed sentence

[ NN ] VB [ ADJ NN NN ] RB PP [ NN ] .

Here the candidate sequence is a target instance in the training sentence. This fact constitutes a *positive* evidence that the candidate is indeed a target instance. The evidence would be even stronger if the candidate's context in the input sentence was the same as that in the training sentence. However, if the candidate (with or without context) also occurs in the training *not* as a target instance, we would also have *negative* evidence indicating that the candidate might not be a target instance. For example, the candidate NN NN RB occurs negatively in the training sentence above.

We may generalize this notion by considering positive and negative evidence to include substrings of the candidate and its context. For example, if the sequence ADJ NN often

occurs at the beginning of a noun phrase in the training, we have some evidence that a candidate beginning with that sequence is a noun phrase. For example, although the candidate ADJ NN NN NN does not occur as a whole in the training sentence above, the sentence gives positive evidence for the candidate, in the form of the prefix subsequence ADJ NN NN and the suffix subsequence NN NN.

The basic idea, then, is to find the set of subsequences of the candidate and its context which provide positive evidence. This set is then used to compute the candidate score.

### 3.1.1 Candidates and tiles

The MBSL scoring algorithm works by considering *situated candidates*. A situated candidate is a POS sequence containing a pair of brackets ('[[ $\dots$ ]']), indicating a candidate for a target instance. The portion of the sentence between the brackets is the *candidate* (as above), while the portions to the left and to the right of the candidate are its *context*.

The idea of the MBSL scoring algorithm is to construct a *tiling* of subsequences of a situated candidate which covers the entire candidate. We consider as *tiles* all subsequences of the situated candidate which contain at least a left or right bracket. Thus we only consider tiles within or adjacent to the candidate that also include a candidate boundary. This constraint reduces the computational complexity of the algorithm (and the run-time in practice), while taking into account the primary evidence for a target instance at the instance boundaries.

Although in principle our 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  $cn$ , the total number of tiles for a candidate of length  $l$  is  $n_{\text{tiles}} = 2 \cdot cn \cdot (l + 2) + 2l + cn^2 + 1$ . For a fixed maximal context,  $n_{\text{tiles}} = O(l)$ , whereas for a fixed length the number of tiles grows as  $cn^2$ . A large  $cn$  will therefore yield an unmanageable number of tiles. That is the reason for considering a fixed maximal context, we have used values of 2 or 3 in our experiments.

Each tile is assigned a score based on its occurrences in the training memory. Since brackets correspond to boundaries of potential target instances, it is important to consider how the bracket positions in the tile correspond to those in the training memory.

For example, consider again the training sentence

[ NN ] VB [ ADJ NN NN ] RB PP [ NN ] .

We may now examine the occurrence in this sentence of several possible tiles:

VB [ ADJ NN occurs positively in the sentence, and

NN NN ] RB also occurs positively, while

NN [ NN RB occurs negatively in the training sentence, since the bracket is in a different position.

The positive evidence for a tile is measured by its *positive count*, the number of times the tile occurs in the training memory with corresponding brackets. Similarly, the negative evidence for a tile is measured by its *negative count*, the number of times that the POS sequence of the tile occurs in the training memory with non-corresponding brackets (either brackets in

the training are in a different position than in the tile, or without brackets). The *total count* of a tile is its positive count plus its negative count. The score  $f_T(t)$  of a tile  $t$  is defined as the ratio of its positive and total counts (other functions could also easily be considered).

$$f_T(t) = \frac{\text{pos\_count}(t)}{\text{total\_count}(t)}$$

We say that a tile whose score  $f_T(t)$  exceeds a given threshold  $\theta_T$ , and thus has ‘sufficient’ positive evidence, is a *matching tile*. Each matching tile gives supporting evidence that a part of the candidate should be considered part of a target instance. In order to combine this evidence, we try to cover the entire candidate by a set of matching tiles, with no gaps. Such a covering constitutes evidence that the entire candidate is a target instance. For example, consider the matching tiles shown for the candidate in figure 2. The set of matching tiles 2, 4, and 5 covers the candidate, as does the set of tiles 1 and 5. Also note that tile 1 constitutes a cover on its own.

insert figure 2 here

To make this precise, we first say that a tile  $t_1$  *connects* to a tile  $t_2$  if (i)  $t_2$  starts after  $t_1$ , (ii) there is no gap between the end of  $t_1$  and the start of  $t_2$  (there may be some overlap), and (iii)  $t_2$  ends after  $t_1$  (neither tile includes the other). For example, tiles 2 and 4 in the figure connect, while tiles 2 and 5 do not, and neither do tiles 1 and 4 (since tile 1 includes tile 4 as a subsequence).

### 3.1.2 Cover statistics

A *cover* for a situated candidate  $c$  is a sequence of matching tiles which collectively cover the entire candidate, including the boundary brackets and possibly some context, such that each tile connects to the following one. A cover thus provides positive evidence for the entire sequence of tags in the candidate.

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 a function of statistics of 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, **num**( $c$ ),
- The least number of tiles constituting a 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**( $c$ ).

Each of these items indicates the strength of the evidence which the covers provide for the candidate.

In order to compute a candidate’s statistics efficiently, we construct the *cover graph* of the candidate. The cover graph is a directed acyclic graph (DAG) whose nodes represent tiles of the candidate, such that an arc exists between nodes  $v(t_1)$  and  $v(t_2)$  whenever tile  $t_1$  connects to  $t_2$ . Two special nodes, START and END, are added to the cover graph. START is connected to every node whose tile contains an open bracket, and similarly, every node representing a tile containing a close bracket is connected to END.

It is easy to see that each path from START to END constitutes a cover, and that every cover gives such a path. Therefore the statistics of all the covers may be efficiently computed by a depth-first traversal of the cover graph. An algorithm for constructing the cover graph is given in figure 3.

insert figure 3 here

A graph whose structure is similar to that of the cover graph was used by Dedina and Nusbaum (1991) and Yvon (1996) for representing alternative pronunciations of a word. In the application of Yvon (1996), the nodes are labelled with phoneme sequences and arcs connect nodes whose labels overlap. START and END nodes are connected to prefix and suffix nodes respectively; each possible pronunciation thus corresponds to a path from START to END. Nevertheless, the scoring mechanism is different as the aim is to find a best path rather than calculating statistics of a graph. Similar with candidate score, though, the path scoring function of Yvon (1996) also prefers more overlap and short paths.

### 3.1.3 Computing the candidate score

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

$$f_C(c) = \alpha \mathbf{num}(c) - \beta \mathbf{minsize}(c) + \gamma \mathbf{maxcontext}(c) + \delta \mathbf{maxoverlap}(c)$$

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. The candidate scoring algorithm is given in figure 4.

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. We plan to investigate in the future a data-driven approach (based on the Winnow algorithm (Littlestone, 1988)) for optimal selection and weighting of statistical features of the score.

insert figure 4 here

### 3.1.4 Complexity

We analyse the worst-case complexity of creating the cover graph and computing the score of a situated candidate in terms of the candidate length  $l$ . The steps of that process are:

**MBSL**(*sentence*, *memory*,  $\theta_T$ ,  $\theta_C$ )

1. Consider each subsequence of *sentence* as a candidate:
2. Construct a situated candidate *s* from the candidate by prepending left context followed by '[' and appending ']' followed by right context;
3. Evaluate  $f_C(c) = \mathbf{CandidateScore}(s, \textit{memory}, \theta_T)$ ;
4. If  $f_C(c) > \theta_C$ , add *c* to *candidate set*;
5. **SelectCandidates**(*candidate set*).

Figure 1: The top-level MBSL bracketing algorithm.

```
Candidate: NN VB [ ADJ NN NN ] RB
MTile 1:   VB [ ADJ NN NN ]
MTile 2:   VB [ ADJ
MTile 3:   [ ADJ NN
MTile 4:           NN NN ]
MTile 5:           NN ] RB
```

Figure 2: A candidate subsequence, and 5 matching tiles found in the training corpus.

**CoverGraph**(*situated-candidate*,  $\theta_T$ )

1.  $V \leftarrow \{\text{START}\}$ ;
2. For each subsequence *t* (potential tile) of *situated-candidate* including either '[' or ']':
3. **SearchTile**(*memory*, *t*) to get positive and total counts;
4. If  $f_T(t) > \theta_T$
5. add a vertex  $v(t)$  to *V*;
6.  $E \leftarrow \emptyset$ ;
7. For each pair of vertices  $v(t_1), v(t_2) \in V$  such that  $t_1$  connects to  $t_2$ :
8. Add the arc  $(v(t_1), v(t_2))$  to *E*;
9. Return the graph  $(V, E)$ .

Figure 3: Constructing the cover graph.

**Create vertices:** Since there are  $O(l)$  potential tiles  $t$  in the situated candidate, and searching for a tile in the memory takes linear time (see below in section 3.4), this step could take  $O(l^2)$ .

**Create edges:** There are at most  $O(l^2)$  possible edges in the graph, so this step takes  $O(l^2)$ .

**Computing statistics:** The DFS takes  $O(|V| + |E|) = O(l^2)$ .

**Computing the score:** This takes a constant time.

Hence computing the candidate score for a situated candidate takes  $O(l^2)$  in the worst case. In practice, however, this worst case is rarely reached, since usually only a small fraction of the tiles are matching.

## 3.2 Selecting candidates

In order to select a bracketing for the input sentence, we assume that target instances are non-overlapping (this is the case for the types of linguistic patterns with which we experimented). 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 5. Other methods for determining the preferred bracketing may also be applied; this remains a topic for future research.

insert figure 5 here

We analyse the complexity of candidate selection by first noting that the number of candidates in the candidate set is at most  $O(n^2)$  where  $n$  is the length of the input sentence. The first step in selection is to sort the candidate set by  $f_C$ , which takes  $O(n^2 \log n^2) = O(n^2 \log n)$ . Since there are at most  $O(n)$  candidates in the final bracketing, with proper indexing removing overlapping candidates takes a total of at most  $O(n^2)$  time, giving a total time of  $O(n^2 \log n)$ .

## 3.3 Worst-case complexity of MBSL

The overall complexity of the bracketing algorithm (figure 1) may now be easily computed. For a sentence of length  $n$ , there are  $O(n^2)$  candidates, each of length at most  $O(n)$ . Hence, since scoring a single candidate of length  $l$  takes  $O(l^2)$ , scoring all of the candidates takes at most  $O(n^4)$ . Since the selection step takes  $O(n^2 \log n)$ , the worst case complexity of the MBSL bracketing algorithm is  $O(n^4)$ . In practice, however, this worst-case is rarely reached.

## 3.4 Implementing the training memory

The MBSL scoring algorithm above 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 possible tiles for each target instance (positive examples) in the training corpus using a trie data structure. A trie is a tree whose arcs are labeled with characters such that each path from the root represents

a distinct tile in the training. A 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 counts for each tile at its associated node in the trie.

Given a trie as described, determining the positive and total counts of a given potential tile is done by searching the trie for the tile, and simply returning the counts stored at the node found. This takes time linear in the length of the potential tile. If tile  $t_1$  is a prefix of  $t_2$ , then searching for  $t_2$  after searching for  $t_1$  may be done incrementally, by starting from  $t_1$ 's node. Hence searching for a set of tiles all starting at the same point in the text is performed incrementally in linear (rather than quadratic) time.

The memory trie itself is built in two passes. Their worst-case complexity depends on  $k$ , the number of sentences in the corpus, and  $n$ , the maximum sentence length.

In the first pass, all the tiles of each target instance in the corpus are inserted into the trie. For example, considering a maximum context of 1, given the NP instance

VB [ NN ] IN

the possible tiles are:

VB [  
 VB [ NN  
 VB [ NN ]  
 VB [ NN ] IN  
 [ NN  
 [ NN ]  
 [ NN ] IN  
 NN ]  
 NN ] IN  
 ] IN

For each possible tile  $t$ , if  $t$  is not in the trie, new nodes required for constructing a path representing  $t$  are added, and the positive count of the final node of the path is set to 1. Otherwise, if  $t$  is already in the trie, the positive count of its associated node is increased by 1. This search is performed incrementally for tiles sharing a prefix and so adding all tiles starting at a given point in a target instance takes time linear in the length of the tile (at most  $O(n)$ ). Thus, since there are at most  $O(n)$  tile starting points in a sentence, the first pass takes time at most  $O(kn^2)$ .

In the second pass, we compute the total counts for each node by considering all the subsequences of every sentence in the corpus. For each such subsequence  $s$ , we find each node in the trie whose associated subsequence is identical to  $s$  except possibly for the addition of brackets. We then increment the total count for each such matching node. Since each tile is a substring of a sentence, there are at most  $O(n^2)$  ways to construct a possible tile for  $s$ . Therefore, there are at most  $O(n^2)$  matching nodes in the trie. Here too we search the trie for all subsequences that share a starting point incrementally, and so the search takes time  $O(n^3)$  for each starting point, and  $O(kn^4)$  for the entire corpus. Therefore the overall worst-case complexity of building the memory is  $O(kn^4)$ . Although this complexity looks

large, in practice the worst case is not really reached. The reason is that pattern instances, hence tiles, are commonly much shorter than the sentence in which they are embedded. The actual time required for building the memory is not long, for example: in the experiments on NPs, the training data contain 8936 sentences, 54760 patterns, and 229,598 words. With a maximum context of 3, it took 16 seconds to build the memory on a 300 MHz Pentium II running Linux. The bracketing rate was 1000 sentences per minute, and the trie took up 100MB of memory.

In a previous implementation (Argamon, Dagan, and Krymolowski, 1998) we encoded the entire corpus in two suffix trees. Inspired by Satta and Henderson (1997), one suffix tree encoded the positive examples and the other encoded the entire corpus, and was used for obtaining the total counts of tiles. Following Roth (1998b) we have decided to use a trie of positive tiles only. While using suffix trees gives a better worst-case time complexity for building the memory, using a trie improves performance by 25-30%. This is mainly because the trie is used for storing positive examples only.

## 4 Evaluation

### 4.1 The Data

We have tested our algorithm in recognizing three syntactic patterns: noun phrase sequences (NP), verb-object (VO), and subject-verb (SV) relations. The training and testing data were derived from the Penn TreeBank.

We used the NP data prepared by RM95, these data were tagged by Brill's POS tagger. The SV and VO data were extracted using T (TreeBank's search script language) scripts (available at <http://www.cs.biu.ac.il/~yuvalk/MBSL>). Table 1 summarizes the sizes of the training and test data sets and the number of examples in each. The T scripts did not attempt to match dependencies over very complex structures, since we are concerned with shallow, or local, patterns.

insert table 1 here

For VO patterns, we put the starting delimiter just before the main verb and the ending delimiter just after the object head. For example:

```
... investigators started to  
[ view the lower price levels ]  
as attractive ...
```

We used a similar policy for SV patterns, defining the start of the pattern at the start of the subject noun phrase and the end right at the first verb encountered. For example:

```
... argue that  
[ the U.S. should regulate ]  
the class ...
```

The subject and object noun-phrase boundaries were those specified by the Penn TreeBank annotators, and phrases containing conjunctions or appositives were not further analysed.

**CandidateScore**(*situated-candidate*, *memory*,  $\theta_T$ )

1. Let  $G = \mathbf{CoverGraph}$ (*situated-candidate*,  $\theta_T$ );
2. Compute **num**, **minsize**, **maxcontext**, and **maxoverlap** by performing DFS on  $G$ ;
3. Return the candidate score  $f_C$  as a function of these statistics.

Figure 4: The candidate scoring algorithm

**SelectCandidates**(*candidate-set*)

1. Examine each candidate  $c \in \textit{candidate-set}$  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 5: Candidate selection algorithm.

Train Data:			
	sentences	words	patterns
NP	8936	229598	54760
VO	16397	454375	14271
SV	16397	454375	25024

Test Data:			
	sentences	words	patterns
NP	2012	51401	12335
VO	1921	53604	1626
SV	1921	53604	3044

Table 1: Sizes of training and test data

By terminating the SV pattern right after the first verb encountered we get seemingly incorrect relation descriptions such as:

```
[ the bird which stands ] there sings
```

While the actual SV pair is `bird - sings`, the T script extracts `bird - stands`. This pair of words is not a properly grammatical SV, but it contains important information and may be regarded as grammatically meaningful in its own right. Hence we do not believe that it impinges on the evaluation of our learning method.

Table 2 shows the distribution of pattern length in the train data. We did not attempt to extract passive-voice VO relations. We have extracted SVs from 92.4% of the sentences and VOs from 58.5% (these figures refer to the overall train and test data). Almost all (99.8%) of the sentences in the NP data contain a noun-phrase. Table 3 shows the number of examples for corpus sizes of 50K, 100K, 150K, and 200K.

insert table 2 here

insert table 3 here

## 4.2 Testing Methodology

The test procedure has two parameters: (a) maximum context size of a candidate, which limits *what* queries are performed on the memory, and (b) the threshold  $\theta_T$  used for establishing a matching tile, which determines *how* to make use of the query results. (The candidate score threshold,  $\theta_C$  was set to 0, such that every candidate with at least one cover was considered.)

Recall and precision figures were obtained for various parameter values.  $F_\beta$ , a common measure in information retrieval (van Rijsbergen, 1979), was used as a single-figure measure of performance:

$$F_\beta = \frac{(\beta^2 + 1) \cdot P \cdot R}{\beta^2 \cdot P + R}$$

We use  $\beta$  which gives no preference to either recall or precision.

Performance may be measured also on a word-by word basis, counting as a success any word which was identified correctly as being part of the target pattern. That method was employed, along with recall/precision, by RM95. We preferred to measure performance by recall and precision for complete patterns. Most errors involved identifications of slightly shifted, shorter or longer sequences. Given a pattern consisting of five words, for example, identifying only a four-word portion of this pattern would yield both a recall and precision errors. Tag-assignment scoring, on the other hand, will give it a score of 80%. We hold the view that such an identification is an error, rather than a partial success.

## 4.3 Results

Table 4 summarizes the optimal parameter settings and results for NP, VO, and SV on the test set. In order to find the optimal values of the context size and threshold, we tried

Len	NP	%	VO	%	SV	%
1	16959	31				
2	21577	39	3203	22	7613	30
3	10264	19	5922	41	7265	29
4	3630	7	2952	21	3284	13
5	1460	3	1242	9	1697	7
6	521	1	506	4	1112	4
7	199	0	242	2	806	3
8	69	0	119	1	592	2
9	40	0	44	0	446	2
10	18	0	20	0	392	2
>10	23	0	21	0	1817	8
total	54760		14271		25024	
avg. len	2.2		3.4		4.5	

Table 2: Distribution of pattern lengths, total number of patterns and average length in the training data.

Words	NP	VO	SV
50K	12100	1603	2764
100K	23864	3180	5385
150K	35799	4798	8237
200K	47730	6313	11034

Table 3: Number of pattern instances within 50K to 200K words

$0.1 \leq \theta_T \leq 0.95$ , and maximum context sizes of 1, 2, and 3. Our experiments used 5-fold cross-validation on the training data to determine the optimal parameter settings.

insert table 4 here

In experimenting with the maximum context size parameter, we found that the difference between the values of  $F_\beta$  for context sizes of 2 and 3 is less than 0.5% for the optimal threshold. Scores for a context size of 1 yielded  $F_\beta$  values smaller by more than 1% than the values for the larger contexts.

Figure 6 shows recall/precision curves for the three data sets, obtained by varying  $\theta_T$  while keeping the maximum context size at its optimal value. The difference between  $F_{\beta=1}$  values for different thresholds was always less than 2%.

insert figure 6 here

Figure 7 shows the learning curves by amount of training examples and number of words in the training data, for particular parameter settings.

insert figure 7 here

The results of RM95 are shown in table 4 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% ( $F_\beta=93.3\%$ ); 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 the results of our algorithm, because it also does not consider lexical information. All results for NPs 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 4 at the line marked NP<sup>†</sup>.

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 5. 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 -

	Con.	Thresh.	Break Even	Recall (%)	Precision (%)	$F_{\beta=1}$
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 <sup>†</sup>	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

Table 4: Results with optimal parameter settings for context size and threshold, and breakeven points. The NP<sup>†</sup> line presents results obtained when removing adverbs and prepositions from the beginning of NPs (section 4.3). 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. The optimal parameters were obtained by 5-fold cross-validation. Note that the results for SV and VO were obtained from Penn TreeBank data which include trace markers. In table 5 we present results without the trace markers. These results are degraded, especially for VO.

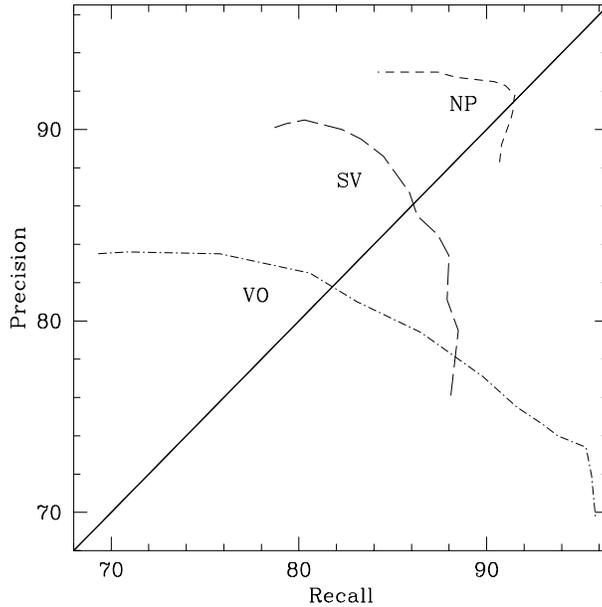


Figure 6: Recall-Precision curves for NP, VO, and SV;  $0.1 \leq \theta \leq 0.99$

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.

## 4.4 Common Errors

This section contains a brief account of typical error-sources for the NP, SV, and VO learning tasks. In the examples shown [ ] bracket the true pattern instance, whereas [[ ]] bracket the one found by the algorithm.

The NP data were tagged automatically, that certainly introduced some errors. Since the tagging errors are consistent throughout the train and test data, their effect is to increase data sparseness. Most of the errors in the NP learning task were due to:

- Coordinations and punctuation marks: Depending on the context, some NPs which contain a coordination were split and some NPs separated by a coordination were detected as a single NP.

```
, , [ [[ DT a JJ local NN lawyer ]] CC and
[[ JJ human-rights NN monitor ]] ], ,
```

```
NN relationship IN between [[ [ NNP Mr. NNP Noriega ]
CC and NNP [ Washington ] ]] IN that
```

In order to estimate the effect of coordinations we made a test in which all NPs containing CC's were split to their more basic components. That increased the number of phrases by 2%, and the recall/precision to 93.4% and 92.9% respectively. However, we are then faced with the new problem of deciding whether or not to create a single NP from two NPs connected by a coordination.

- Adjacent phrases: Cases where two adjacent NPs were detected as a single phrase

```
VBZ recalls [[ [ PRP$ his NN close NN ally ]
[ NNP Jose NNP Blandon ] ]] , ,
```

- Ambiguity of verbs: Words tagged VBG or VBN may be verbs as well as modifiers. The way these words are handled depends on the context

, , [VBN fixed [[ VBG leading NNS edges]] ] IN for

Note that `leading` was part of the detected NP, probably due to the VBN context, vs.

PRP it VBD had [VBG operating [[ NN profit ]] ] IN of \$ \$ CD million

where the context is different. A proper treatment of this issue has to take into account lexical information, not only POS data.

- Quotation marks: Many of the errors involved phrases which contain quotation marks

IN by [ ‘ ‘ ‘ ‘ [[ DT a RB very JJ meaningful ’ ’ ’ ’  
NN increase ]] ] IN in

A test with all quotation marks completely removed have not yielded any significant improvement in the results.

In the SV and VO learning tasks, the text was taken directly from Penn TreeBank, hence the number of tagging errors was smaller. We have made tests with quotation marks removed for these tasks as well, using the optimal parameters. The result was a degradation of the precision by 1% for SV and 0.5% for VO, along with a slight increase of the recall (0.2% for SV and 0.4% for VO). These effects are insignificant.

The VO patterns did not take in account cases where the verb is a be-verb, such as the pair `is-chairman` in

NNP Mr. NNP Vinken VBZ is NN chairman IN of

Using POS tags only, it is impossible to distinguish these verbs. We therefore conducted an experiment in which `be`, `am`, `is`, `are`, `were`, and `was` (and their abbreviations) were tagged with the new POS tag `VBE`. That was a rudimentary way of introducing lexical knowledge. The results of this experiment are presented in table 5. The incorporated knowledge caused an improvement of 1.5% in the recall and a more significant precision improvement of 4.3%.

A significant part of the remaining VO recognition errors were results of verbs appearing as modifiers rather than actions:

NN asbestos [[ VBG including NN crocidolite ]] RBR more RB stringently

DT all [[ VBG remaining NNS uses ]] IN of

TO to [ VB indicate [[ VBG declining NN interest NNS rates ]] ]

Regarding SV learning, most of the recall errors involve very long pattern instances. As table 2 shows, about 8% of the SV patterns are more than 10 words long. With such length, the diversity is large and it is harder to model all the cases. Similarly, many of the precision mistakes are a result of erroneously detecting relatively short instances, or sub-parts of a pattern. The variety of possible tiles increases the chance of false detections.

insert table 5 here

## 5 Conclusions

We have presented a novel general algorithm for learning sequential patterns. Applying the method to three syntactic patterns in English yielded positive results, suggesting its applicability for recognizing syntactic chunks. The results for noun-phrase learning are compatible with current state of the art. The algorithm achieved good results also when applied to patterns which constitute a relation between words (subject-verb, verb-object), rather than a chunk.

Many of the errors can be attributed to the lack of lexical and semantic information, as the input consists only of POS tags. These errors point out sub-problems (e.g. ambiguity of gerunds) which cannot be handled using POS information alone.

The presented algorithm is part of a more comprehensive learnable shallow parser under plan. The planned shallow parser will handle sequential patterns (chunks) as well as attachment problems (e.g. prepositional phrase attachment). Attachment problems, like many of the cases where POS data are not enough, require lexical information. We plan to use SNOW (Roth, 1998a), a feature-based network learning architecture based on Winnow, for learning to handle such problems. SNOW can also be used within our algorithm, for a principled candidate scoring through weighting the statistical features of candidates.

## 6 acknowledgements

The authors wish to thank Yoram Singer for his collaboration in an earlier phase of this research project, and Giorgio Satta for helpful discussions. We also thank Jorn Veenstra, Min Yen Kan, and the anonymous reviewers for their instructive comments. This research was supported in part by grant 498/95-1 from the Israel Science Foundation, and by grant 8560296 from the Israeli Ministry of Science.

## 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, pp. 257-278.
- Aït-Mokhtar, S., and Chanod, J.-P., 1997, Subject and object dependency extraction using finite-state transducers, In *ACL'97 Workshop on Information Extraction and the Building of Lexical Semantic Resources for NLP Applications*, Madrid.

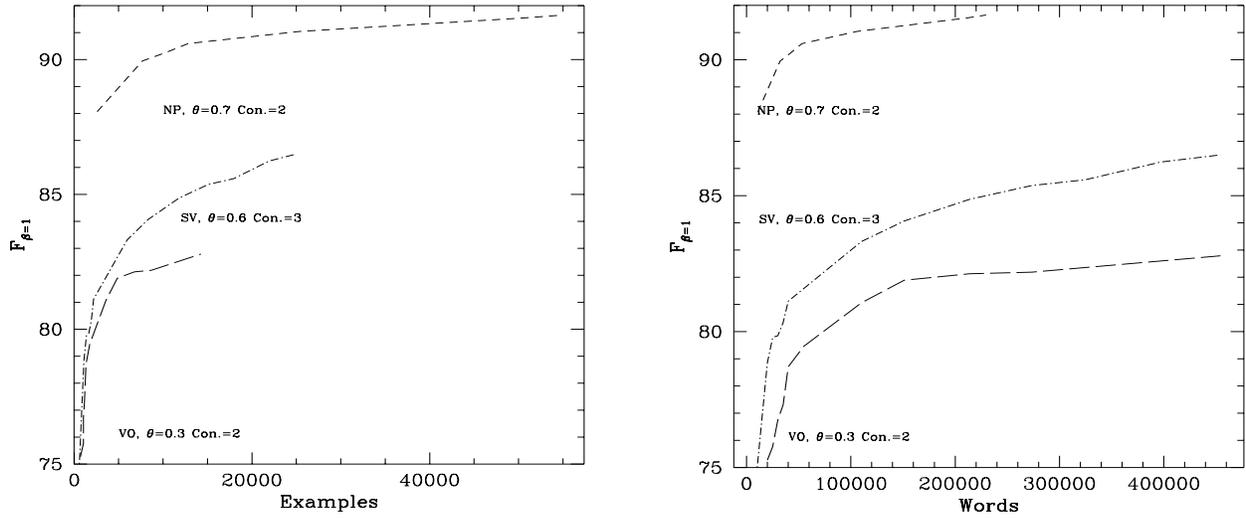


Figure 7: Learning curves for NP, VO, and SV by number of examples (left) and words (right)

	Con.	Thresh.	Recall (%)	Precision (%)	$F_{\beta=1}$
VO	2	0.5	87.6	69.2	77.3
VO <sup>Be</sup>	2	0.5	89.2	73.6	80.6
SV	3	0.6	84.0	87.5	85.7

Table 5: SV and VO results without trace markers, notice the degradation compared with the original data. The line labelled VO<sup>Be</sup> presents results obtained with all be-verbs tagged by a special POS tag.

- Appelt, D. E., Hobbs, J. R., Bear, J., Israel, D., and Tyson, M., 1993, Fastus: A finite-state processor for information extraction from real-world text, In *Proc. of the 13th IJCAI*, pp. 1172–1178, Chambery, France.
- Argamon, S., Dagan, I., and Krymolowski, Y., 1998, A memory-based approach to learning shallow natural language patterns, In *Proc. of COLING/ACL*, pp. 67–73, Montreal, Canada.
- Bod, R., 1992, A computational model of language performance: Data oriented parsing, In *Coling*, pp. 855–859, Nantes, France.
- Brill, E., 1992, A simple rule-based part of speech tagger, In *proc. of the DARPA Workshop on Speech and Natural Language*.
- Brill, E., 1993, Automatic grammar induction and parsing free text: A transformation-based approach, In *Proc. of the Annual Meeting of the ACL*.
- Briscoe, T., and Carroll, J., 1993, Generalized probabilistic LR parsing of natural language corpora with unification-based grammars, *Computational Linguistics*, 19(1):25–60.
- Cardie, C., 1993, A case-based approach to knowledge acquisition for domain-specific sentence analysis, In *Proceedings of the 11th National Conference on Artificial Intelligence*, pp. 798–803, Menlo Park, CA, USA, AAAI Press.
- Cardie, C., and Pierce, D., 1998, Error-driven pruning of treebank grammars for base noun phrase identification, In *Proc. of COLING/ACL*, pp. 218–224, Montreal, Canada.
- Chitrao, M., and Grishman, R., 1990, Statistical parsing of messages, In *Proceedings of DARPA Speech and Natural Language Processing*, Morgan Kaufman: New York.
- Church, K. 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*, Madrid, Spain.
- Daelemans, W., Zavrel, J., Berck, P., and Gillis, S., 1996, MBT: A memory-based part of speech tagger generator, In Eva Ejerhed and Ido Dagan, editors, *Proceedings of the Fourth Workshop on Very Large Corpora*, pp. 14–27, ACL SIGDAT.
- Daelemans, W., van den Bosch, A., and Weijters, T., 1996, IGTrees: Using trees for compression and classification in lazy learning algorithms, *D. Aha ed., Artificial Intelligence Review, special issue on Lazy Learning*.
- Dedina, M. J., and Nusbaum, H. C., 1991, PRONOUNCE: a program for pronunciation by analogy, *Computer Speech and Langage*, 5:55–64.
- Eisner, J., 1996, Three new probabilistic models for dependency parsing: An exploration, In *Proc. of COLING*, pp. 340–346, Copenhagen, Denmark.

- Gee, J. P., and Grosjean, F., 1983, Performance structures: A psycholinguistic and linguistic appraisal, *Cognitive Psychology*, 15:411–458.
- Greffentette, G., 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.
- Littlestone, N., 1988, Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm, *Machine Learning*, 2:285–318.
- Magerman, D., and Marcus, M., 1991, Pearl: A probabilistic chart parser, In *Fourth DARPA Workshop on Speech and Natural Language*, Pacific Grove, CA.
- Magerman, D. M., 1995, Statistical decision-tree models for parsing, In *Proc. of the 33<sup>rd</sup> Annual Meeting of the Association for Computational Linguistics. Cambridge, MA*, 26–30.
- Marcus, M. P., Santorini, B., and Marcinkiewicz, M., 1993, Building a large annotated corpus of English: The Penn Treebank, *Computational Linguistics*, 19(2):313–330.
- Pereira, F., and Schabes, Y., 1992, Inside-outside reestimation from partially bracketed corpora, In *Proc. of the Annual Meeting of the ACL*.
- Ramshaw, L. A., and Marcus, M. P., 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.
- Ron, D., Singer, Y., and Tishby, N., 1995, On the learnability and usage of acyclic probabilistic finite automata, In *Proceedings of the 8th Annual Conference on Computational Learning Theory (COLT'95)*, pp. 31–40, New York, NY, USA, ACM Press.
- Roth, D., 1998a, Learning to resolve natural language ambiguities: A unified approach, In *proc. of the Fifteenth National Conference on Artificial Intelligence*, pp. 806–813, Menlo Park, CA, USA, AAAI Press.
- Roth, D., 1998b, Private Communications.
- Satta, G., and Henderson, J. C., 1997, String transformation learning, In *Proc. of the ACL/EACL Annual Meeting*, pp. 444–451, Madrid, Spain.
- Schiller, A., 1996, Multilingual finite-state noun phrase extraction, In *ECAI '96 Workshop on Extended Finite State Models of Language*, pp. 65–69, Budapest.
- Sekine, S., 1998, *Corpus-Based Parsing and Sublanguage Studies*, Ph.D. thesis, New York University.

- Sima'an, K., 1996, Computational complexity of probabilistic disambiguation by means of tree grammars, In *Proceedings of the seventeenth International Conference on Computational Linguistics (COLING'96)*, pp. 1175–1180, Copenhagen, Denmark.
- Skut, W., and Brants, T., 1998, A maximum-entropy partial parser for unrestricted text, In *Proc. of the sixth Workshop on Very Large Corpora*, Montreal, Canada.
- Stanfill, C., and Waltz, D., 1986, Toward memory-based reasoning, *Communications of the ACM*, 29(12):1213–1228.
- van Rijsbergen, C. J., 1979, *Information Retrieval*, Butterworth.
- Veenstra, J., 1998, Fast np chunking using memory-based learning techniques, In F. Verdenius and W. van den Broek, editors, *Proceedings of Benelearn*, pp. 71–79, Wageningen, the Netherlands.
- Yvon, F., 1996, Grapheme-to-phoneme conversion using multiple unbounded overlapping chunks, In *Proceedings of the conference on New Methods in Natural Language Processing (NeMLaP II)*, pp. 218–228, Ankara, Turkey.