Incorporating Compositional Evidence in Memory-Based Partial Parsing

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

Abstract

In this paper, a memory-based parsing method is extended for handling compositional structures. 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 embedded instances are being used simultaneously. The output is a partial parse in which instances of the target structures are marked.

1 Introduction

A variety of statistical methods were proposed over the recent years for learning to produce a full parse of free-text sentences (e.g., Bod (1992), Magerman (1995), Collins (1997), Ratnaparkhi (1997), and Sekine (1998)). In parallel, a lot of work is being done on *shallow parsing* (Abney, 1991; Greffenstette, 1993), focusing on partial analysis of sentences at the level of local phrases and the relations between them.

Shallow parsing tasks are often formulated as dividing the sentence into non-overlapping sequences of syntactic structures, a task called *chunking*. Most of the chunking works have concentrated on noun-phrases (NPs, e.g. Church (1988), Ramshaw and Marcus (1995), Cardie and Pierce (1998), Veenstra (1998)). Other chunking tasks involve recognizing subject-verb (SV) and verb-object (VO) pairs (Argamon et al., 1999; Munoz et al., 1999).

The output of shallow parsers is useful where a complete parse tree is not required (e.g., classification, summarization, bilingual alignment). The complexity of training full parsing algorithms may therefore be avoided in such tasks. On the other hand, full parsing has the advantages of producing compositional structures, finding multiple structures simultaneously, and using sub-structures for inference about higher-level ones. While shallow systems typically make use of single-word data, a full parser can use higher-level structures in a compositional manner, e.g., a NP for identifying a VP, or a conjunction of NPs in order to identify a longer NP.

A partial parser is typically concerned only with 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.

The approach presented here, of trainable *partial parsing*, attempts to reduce the gap between shallow and full parsing. It is an extension of shallow parsing towards handling composite and multiple patterns, while maintaining the local nature of the task, and simplicity of training material.

One approach to partial parsing was presented by Buchholz et al. (1999), who extended a shallow-parsing technique to partial parsing. The output of NP and VP chunking was used as an input to grammatical relation inference. The inferences process is cascaded, and a clear improvement was obtained by passing results across cascades.

Another approach for partial parsing was presented by Skut and Brants (1998). Their method is an extension of that of Church (1988) for finding NP's, achieved by extending the feature space to include structural information. Processing goes simultaneously for structures at all levels, from left to right. Since there are no cascades, the structural level of the output is limited by that of the feature set.

This paper presents an extension of the algorithm of Argamon et al. (1998, 1999, hereafter MBSL), which handles and exploits compositional structures. MBSL is a memory-based algorithm that uses raw-data segments for learning chunks. It works with POS tags, and combines segments of various lengths in order to decide whether part of the sentence may be an instance of a target pattern. As a memory-based algorithm, it does not abstract over the data during training, but makes the necessary abstractions during inference - for each particular instance.

In extending MBSL, which is a flat method, we have kept the structure of the inference mechanism and its local nature. This paper describes the extended version in a self-contained manner, while elaborating mostly on the extensions needed to handle compositional cases. Section 2 describes the partial parser. Results for NP and VP are presented in Sec. 3, followed by a short discussion in Sec. 4.

2 Algorithm

The training phase receives a list of target types of syntactic patterns that need to be identified (e.g., NP and VP), and a training corpus in which target pattern instances are marked with labeled brackets. It then stores raw-data statistics about these instances in a memory data structure. The parsing phase receives a sentence to be parsed and identifies instances of the target patterns based on the information stored in the memory. In the remainder of the paper we mostly adopt the terminology of the flat version.

2.1 An Illustrating Example

Suppose the training data contains the sentences as in Fig. 1, 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.

Every range of words in the sentence is considered a *candidate* for being a VP. The flat MBSL algorithm tries to support that hypothesis by matching POS subsequences of the candidate (and possibly some of its surrounding context) with subsequences of VPs that appeared in the training corpus. These subsequences, called *tiles*, should contain at least one VP boundary bracket.

In the example above, consider the range of words 4-8 as a candidate VP. That POS sequence does not appear as a complete VP in training, but some of its tiles do appear and can provide supporting evidence. The tile "`[VP RB VBZ`" provides a positive evidence because the only appearance of "`RB VBZ`" is at the beginning of a VP. On the other hand, "`NN [VP`" provides a weaker evidence because "`NN`" appears twice before a VP but three times in other positions. Accordingly, each tile has a positive count (*pos_count*), specifying the number of times the POS sequence appeared in training at the same position within the VP as it appears within the candidate, and a negative count (*neg_count*), specifying the number of times the sequence appeared in other positions. For "`[VP RB VBZ`" we have $pos\_count = 1$ and $neg\_count = 0$, while for "`NN [VP`" $pos\_count = 2$ and $neg\_count = 3$.

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

Some of the `VP` tiles used by the compositional algorithm are presented in Fig. 2. 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 range 4-8 being a VP because together they

```
1.     [NP NNS NP] [VP RB VBZ IN [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 1: An example training data

```
                                                                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]                          2    0
8.                           NP    VP]                          3    6
```

Figure 2: Some of the VP tiles derived from the data in Fig. 1

cover the whole range. Assuming the inner NP was detected, tiles 6-8 reflect the its presence within the composite VP. Tile 6 alone already provides a positive evidence for the entire VP, as it covers the whole range of the candidate words. 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 range of the candidates words. The composite algorithm could produce the evidence without this sentence, by realizing the compositional evidence.

2.2 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. Figure 3 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 3). In these terms, a tile t of the given instance 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, the tile is embedded 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).

2.3 Memory Structure

As illustrated in Section 2.1, the algorithm uses the training corpus statistics, *pos_count* and *total_count* for every tile it encounters. The memory encodes these statistics in a trie data structure: each path from the root of the trie to a certain node corresponds to a tile, storing the tile statistics in the corresponding node. The arc labels along the path are the POS, instance types and labeled brackets of the corresponding pattern. Tiles of candidate instances can be retrieved from the memory during parsing by following the corresponding path in the trie. The trie is created during the training phase by constructing the sentence graph for each sentence and generating all the tiles for each pattern instance.

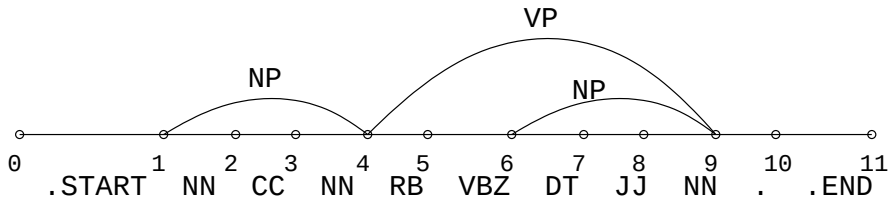

Figure 3: A sample sentence graph with context symbols, the target patterns are NP and VP

2.4 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.

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.

The following subsection describes the core part of the algorithm, scoring a candidate.

2.5 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 statistics of the corresponding tile.

The algorithm computes the following score function for each tile found in the trie :

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

Tiles satisfying

$$f_T(t) > \theta_T,$$

where θ_T is a pre-specified threshold, are considered as supporting evidence for the candidate and contribute to its score.

Once all supporting tiles are found, the algorithm tries to use them for covering the entire candidate. Tile information is stored in a *cover graph*, as in the flat version, to facilitate efficient scoring. A *cover* for a candidate is a set of tiles such that:

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

Often, a number of covers can be created 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:

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

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 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).

3 Evaluation

The system was trained on the Penn Treebank (Marcus et al., 1993) WSJ Sections 2-21 and tested on Section 23 (Table 1), 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 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.

Train Data, WSJ 02-21, 28884 sentences

	base	composite	base:all
NP	166242	61384	73%
VP	43377	28017	61%

Test Data, WSJ 23, 2416 sentences

	base	composite	base:all
NP	13524	5106	73%
VP	3496	2267	61%

Table 1: Sizes of training and test data, note the similar proportions of base instances

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 precision errors - 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 2 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 3), allowing embedded instances improved the recall in expense of precision. We have experimented with seperating NPs from base-NPs [1]. As Table 3 shows,

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

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.

	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 2: VP Results, $\theta_T = 0.6$, tile length≤ 5

	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

Table 3: NP Results, $\theta_T = 0.6$, tile length≤ 5. Rows 5–10 refer to experiments where base-NPs were distinguished from composite ones.

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. Our results are lower than those of full parsers, e.g., Collins (1997) – as might be expected since much less structural data, and no lexical data are being used.

4 Discussion

We have presented a memory-based learning method for partial parsing which can handle and exploit compositional information. Like other shallow-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 et al. (1999) provide a thorough comparison of the flat MBSL method with DOP. Considering POS data only, the compositional method resembles the DOP1 model in that it uses sub-constituent information in order to construct evidence for higher-level structures. There are, however, some differences:

- Consider the input `[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 ]`.
- 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 statistical model, whereas in MBSL, it is based on properties calculated from the cover graph.
- In MBSL it is possible to specify if tiles of a composite instance will be created

bottom-up or top-down, and limit the level. This can be useful in highly-nested instances, where going all the way top-down will create a lot of tiles.

- DOP can be naturally formulated to allow for wildcards, as in the DOP3 model, by allowing some of the leaves to be unknown. Allowing wildcards in MBSL would be more complicated because it relies on tiling of continuous sequences.

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, Apple Pie Parser) and Sekine (1998). The algorithm extracts grammar rules with `S` and `NP` (and possibly other structures) as non-terminals. Both Apple Pie (hereafter 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 when generating the non-terminal `S`. In MBSL, the context is consulted for each instance candidate.

- APP, like DOP, uses a probabilistic model. The probability of a grammar rule $\mathtt{X} \leftarrow \mathtt{Y}$ is $\mathrm{Freq}(\mathtt{X} \leftarrow \mathtt{y})/\mathrm{Freq}(\mathtt{X})$. Analogously, the denominator in MBSL would be $\mathrm{Freq}(\mathtt{Y})$.

The presented method concerns primarily with phrases, which can be represented by a tree structure. It is not aimed at handling dependencies, which require heavy use of lexical information (Hindle and Rooth, 1993, for PP attachment). As (Daelemans et al., 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 providing instance *candidates*, 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 prepositions like 'of' which attaches mainly to nouns (Sekine and Grishman, 1995).

In a similar vain to Skut and Brants (1998) and Buchholz et al. (1999), the method extends an existing flat shallow-parsing method to handle composite structures. 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.

Acknowledgements Y. K. thanks Jorn Veenstra, Sabine Buchholz, and Khalil Sima'an for thorough and helpful discussions, as well as Walter Daelemans and Antal van der Bosch for helpful comments.

References

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

S. Argamon, 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.

S. Argamon, 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.

R. Bod. 1992. A computational model of language performance: Data oriented parsing. In *Coling*, pages 855–859, Nantes, France.

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

C. Cardie 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.

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

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

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

Gregory Greffenstette. 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.

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

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

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

M. Munoz, 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.

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

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

Remko Scha, 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.

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

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

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

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

J. Veenstra. 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.