Learning to Transform Linguistic Graphs

Valentin Jijkoun and Maarten de Rijke
ISLA, University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
jijkoun,mdr@science.uva.nl

Abstract

We argue in favor of the the use of labeled directed graph to represent various types of linguistic structures, and illustrate how this allows one to view NLP tasks as graph transformations. We present a general method for learning such transformations from an annotated corpus and describe experiments with two applications of the method: identification of non-local depenencies (using Penn Treebank data) and semantic role labeling (using Proposition Bank data).

1 Introduction

Availability of linguistically annotated corpora such as the Penn Treebank (Bies et al., 1995), Proposition Bank (Palmer et al., 2005), and FrameNet (Johnson et al., 2003) has stimulated much research on methods for automatic syntactic and semantic analysis of text. Rich annotations of corpora has allowed for the development of techniques for recovering deep linguistic structures: syntactic non-local dependencies (Johnson, 2002; Hockenmaier, 2003; Dienes, 2004; Jijkoun and de Rijke, 2004) and semantic arguments (Gildea, 2001; Pradhan et al., 2005; Toutanova et al., 2005; Giuglea and Moschitti, 2006). Most state-of-the-art methods for the latter two tasks use a cascaded architecture: they employ syntactic parsers and re-cast the corresponding tasks as pattern matching (Johnson, 2002) or classification (Pradhan et al., 2005) problems. Other methods (Jijkoun and de Rijke, 2004) use combinations of pattern matching and classification.

The method presented in this paper belongs to the latter category. Specifically, we propose (1) to use a flexible and expressive graph-based representation of linguistic structures at different levels; and (2) to view NLP tasks as graph transformation problems: namely, problems of transforming graphs of one type into graphs of another type. An example of such a transformation is adding a level of the predicate argument structure or semantic arguments to syntactically annotated sentences. Furthermore, we describe a general method to automatically learn such transformations from annotated corpora. Our method combines pattern matching on graphs and machine learning (classification) and can be viewed as an extension of the Transformation-Based Learning paradigm (Brill, 1995). After describing the method for learning graph transformations we demonstrate its applicability on two tasks: identification of non-local dependencies (using Penn Treebank data) and semantic roles labeling (using Proposition Bank data).

The paper is organized as follows. In Section 2 we give our motivations for using graphs to encode linguistic data. In Section 3 we describe our method for learning graph transformations and in Section 4 we report on experiments with applications of our method. We conclude in Section 5.

2 Graphs for linguistic structures and language processing tasks

Trees and graphs are natural and common ways of encoding linguistic information, in particular, syn-

TextGraphs-2: Graph-Based Algorithms for Natural Language Processing, pages 53–60,
Rochester, April 2007

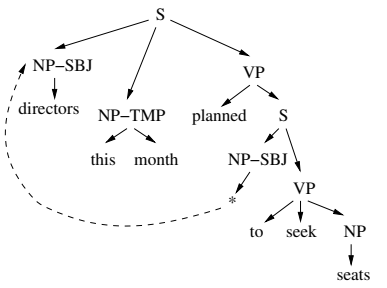

Figure 1: Local and non-local syntantic relations.

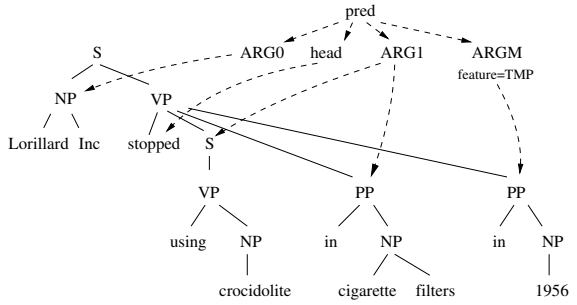

Figure 2: Syntactic structure and semantic roles.

tactic structures (phrase trees, dependency structures). In this paper we use node- and edge-labeled directed graphs as our representational formalism. Figures 1 and 2 give informal examples of such representations.

Figure 1 shows a graph encoding of the Penn Treebank annotation of the local (solid edges) and non-local (dashed edges) syntantic structure of the sentence *directors this month planned to seek more seats*. In this example, the co-indexing-based implicit annotation of the non-local dependency (subject control) in the Penn Treebank (Bies et al., 1995) is made explicit in the graph-based encoding.

Figure 2 shows a graph encoding of linguistic structures for the sentence *Lorillard Inc stopped using crocodolite in sigarette filters in 1956*. Here, solid lines correspond to surface syntactic structure, produced by Charniak's parser (Charniak, 2000), and dashed lines are an encoding of the Proposition Bank annotation of the semantic roles with respect to the verb *stopped*.

Graph-based representations allow for a uniform view on the linguistic structures on different layers. An advantage of such a uniform view is that apparently different NLP tasks can be considered as

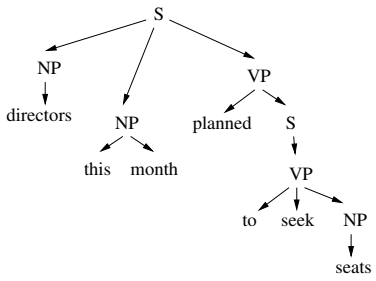

Figure 3: Output of a syntactic parser.

manipulations with graphs, in other words, as graph transformation problems.

Consider the task of recovering non-local dependencies (such as control, WH-extraction, topicalization) in the surface syntactic phrase trees produced by the state-of-the-art parser of (Charniak, 2000). Figure 3 shows a graph-based encoding of the output of the parser, and the task in question would consist in transforming the graph in Figure 3 into the graph in Figure 1. We notice that this transformation can be realised as a sequence of independent and relatively simple graph transformations: adding nodes and edges to the graph or changing their labels (e.g., from NP to NP-SBJ).

Similarly, for the example in Figure 2, adding a semantic layer (dashed edges) to the syntactic structure can also be seen as transforming a graph.

In general, we can view NLP tasks as adding additional linguistic information to text, based on the information already present: e.g., syntactic parsing taking part-of-speech tagged sentences as input (Collins, 1999), or anaphora resolution taking sequences of syntactically analysed and named-entity-tagged sentences. If both input and output linguistic structures are encoded as graphs, such NLP tasks become graph transformation problems.

In the next section we describe our general method for learning graph transformations from an annotated corpus.

3 Learning graph transformations

We start with a few basic definitions. Similar to (Schürr, 1997), we define ıemphgraph as a relational structure, i.e., a set of objects and relations between them; we represent such structures as sets of first-order logic atomic predicates defining nodes,

directed edges and their attributes (labels). Constants used in the predicates represent *objects* (nodes and edges) of graphs, as well as attribute names and values. Atomic predicates $\mathsf{node}(\cdot)$, $\mathsf{edge}(\cdot,\cdot,\cdot)$ and $\mathsf{attr}(\cdot,\cdot,\cdot)$ define nodes, edges and their attributes. We refer to (Schürr, 1997; Jijkoun, 2006) for formal definitions and only illustrate these concepts with an example. The following set of predicates:

$$\mathsf{node}(n_1), \mathsf{node}(n_2), \mathsf{edge}(e, n_1, n_2),$$
$$\mathsf{attr}(n_1, \mathsf{label}, \mathsf{Src}), \mathsf{attr}(n_2, \mathsf{label}, \mathsf{Dst})$$

defines a graph with two nodes, n_1 and n_2, having labels Src and Dst (encoded as attributes named label), and an (unlabelled) edge e going from n_1 to n_2.

A *pattern* is an arbitrary graph and an *occurence* of a pattern P in graph G is a total injective homomorphism Ω from P to G, i.e., a mapping that associates each object of P with one object G and preserves the graph structure (relations between nodes, edges, attribute names and values). We will also use the term *occurence* to refer to the graph $\Omega(P)$, a subgraph of G, the image of the mapping Ω on P.

A *graph rewrite rule* is a triple $r = \langle lhs_r, C_r, rhs_r \rangle$: the left-hand side, the constraint and the right-hand side of r, respectively, where lhs_r and rhs_r are graphs and C_r is a function that returns 0 or 1 given a graph G, pattern lhs_r and its occurence in G (i.e., C_r specifies a constraint on occurences of a pattern in a graph).

To *apply* a rewrite rule $r = \langle lhs_r, C_r, rhs_r \rangle$ to a graph G means finding all occurences of lhs_r in G for which C_r evaluates to 1, and replacing such occurences of lhs_r with occurences of rhs_r. Effectively, objects and relations present in lhs_r but not in rhs_r will be removed from G, objects and relations in rhs_r but not in lhs_r will be added to G, and common objects and relations will remain intact. Again, we refer to (Jijkoun, 2006) for formal definitions.

As will be discussed below, our method for learning graph transformations is based on the ability to compare pairs of graphs, identifying where the two graphs are similar and where they differ. An *alignment* of two graphs is a partial one-to-one homomorphism between their nodes and edges, such that if two edges of the two graphs are aligned, their respective endpoints are aligned as well. A *maximal alignment* of two graphs is an alignment that maximizes the sum of (1) the number of aligned objects (nodes and edges), and (2) the number of matching attribute values of all aligned objects. In other words, a maximal alignment identifies as many similarities between two graphs as possible. Given an alignment of two graphs, it is possible to extract a list of rewrite rules that can transform one graph into another. For a maximal alignment such a list will consist of rules with the smallest possible left- and right-hand sides. See (Jijkoun, 2006) for details.

As stated above, we view NLP applications as graph transformation modules. Our supervised method for learning graph transformation requires two corpora: input graphs $In = \{In_k\}$ and corresponding output graphs $Out = \{Out_k\}$, such that Out_k is the desired output of the NLP module on the input In_k.

The result of the method is an ordered list of graph rewrite rules $R = \langle r_1, \ldots r_n \rangle$, that can be applied in sequence to input graphs to produce the output of the NLP module.

Our method for learning graph transformations follows the structure of Transformation-Based Learning (Brill, 1995) and proceeds iteratively, as shown in Figure 4. At each iteration, we compare and align pairs of input and output graphs, identify possible rewrite rules and select rules with the most frequent left-hand sides. For each selected rewrite rule r, we extract all occurences of its left-hand side and use them to train a two-class classifier implementing the constraint C_r: the classifier, given an encoding of an occurence of the left-hand side predicts whether this particular occurence should be replaced with the corresponding right-hand side. When encoding an occurence as a feature vector, we add as features all paths and all attributes of nodes and edges in the one-edge neighborhood from the nodes of the occurence. For the experiments described in this paper we used the SVM Light classifier (Joachims, 1999) with a standard linear kernel. See (Jijkoun, 2006) for details.

4 Applications

Having presented a general method for learning graph transformations, we now illustrate the method at work and describe two applications to concrete

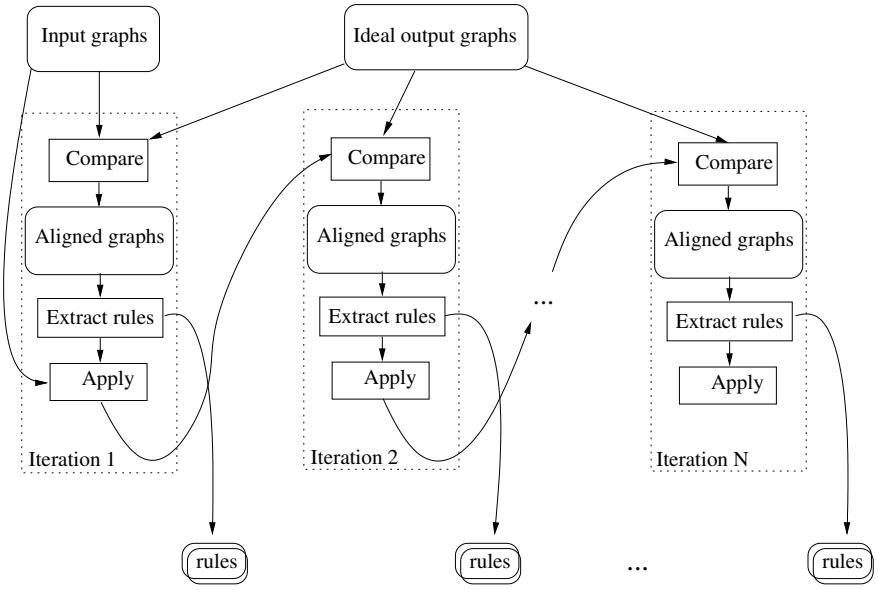

Figure 4: Structure of our method for learning graph transformations.

NLP problems: identification of non-local dependencies (with the Penn Treebank data) and semantic role labeling (with the Proposition Bank data).

4.1 Non-local dependencies

State-of-the-art statistical phrase structure parsers, e.g., Charniak's and Collins' parsers trained on the Penn Treebank, produce syntactic parse trees with bare phrase labels, (NP, PP, S, see Figure 3), i.e., providing surface grammatical analysis of sentences, even though the training corpus, the Penn Treebank, is richer and contains additional grammatical and semantic information: it distinguishes various types of modifiers, complements, subjects, objects and annotates non-local dependencies, i.e., relations between phrases not adjacent in the parse tree (see Figure 1). The task of recovering this information in the parser's output has received a good deal of attention. (Campbell, 2004) presents a rule-based algorithm for empty node identification in syntactic trees, competitive with the machine learning methods we mention next. In (Johnson, 2002) a simple pattern-matching algorithm was proposed for inserting empty nodes into syntactic trees, with patterns extracted from the Penn Treebank. (Dienes, 2004) used a preprocessor that identified surface location of empty nodes and a syntactic parser incorporating non-local dependencies into its probabilistic model. (Jijkoun and de Rijke, 2004) described an extension of the pattern-matching method with a classifier trained on the dependency graphs derived from the Penn Treebank data.

In order to apply our graph transformation method to the task of identifying non-local dependencies, we need to encode the information provided in the Penn Treebank annotations and in the output of a syntactic parser using directed labeled graphs. We used a straightforward encoding of syntactic trees, with nodes representing terminals and non-terminals and edges defining the parent-child relationship. For each node, we used the attribute type to specify whether it is a terminal or a non-terminal. Terminals corresponding to Penn empty nodes were marked with the attribute empty $= 1$. For each terminal (i.e., each word), the values of attributes pos, word and lemma provided the part-of-speech tag, the actual form and the lemma of the word. For non-terminals, the attribute label contained the label of the corresponding syntactic phrase. The co-indexing of empty nodes and non-terminals used in the Penn Treebank to annotate non-local dependencies was encoded using explicit edges with a distinct type attribute, connecting empty nodes with their antecedents (e.g., the dashed edge in Figure 1). For each non-terminal node, its head child was marked by attaching attribute head with value 1 to the corre-

sponding parent-child edge, and the lexical head of each non-terminal was explicitly indicated using additional edges with the attribute type = lexhead. We used a heuristic method of (Collins, 1999) for head identification.

When Penn Treebank sentences and the output of the parser are encoded as directed labeled graphs as described above, the task of identifying non-local dependencies can be formulated as transforming phrase structure graphs produced by a parser into graphs of the type used in Penn Treebank annotations.

We parsed the strings of the Penn Treebank with Charniak's parser and then used the data from sections 02–21 of the Penn Treebank for training: encoding of the parser's output was used as the corpus of input graphs for our learning method, and the encoding of the original Penn annotations was used as the corpus of output graphs. Similarly, we used the data of sections 00–01 for development and section 23 for testing. Using the input and output corpora, we ran the learning method as described above, at each iteration considering 20 most frequent left-hand sides of rewrite rules. At each iteration, the learned rewrite rules were applied to the current training and development corpora to create a corpus of input graphs for the next iteration (see Figure 4) and to estimate the performance of the system at the current iteration. The system was evaluated on the development corpus with respect to non-local dependencies using the "strict" evaluation measure of (Johnson, 2002): the F_1 score of precision and recall of correctly identified empty nodes and antecedents. If the absolute improvement of the F_1 score for the evaluation measure was smaller than 0.1, the learning cycle was terminated, otherwise a new iteration was started.

The learning cycle terminated after 12 iterations. The resulting sequence of $12 \times 20 = 240$ graph rewrite rules was applied to the test corpus of input graphs: Charniak's parser output on the strings of section 23 of the Penn Treebank. The result was evaluated against the original annotations of the Penn Treebank.

The results of the evaluation of the system on empty nodes and non-local dependencies and the PARSEVAL F_1 score on local syntactic phrase structure against the test corpus at each iteration are shown in Table 1.

Stage	P	R	F_1	PARSEVAL F_1
Initial	0.0	0.0	0.0	88.7
1	88.2	38.6	53.7	88.4
2	87.2	48.6	62.5	88.4
3	87.5	51.9	65.2	88.4
4	86.7	52.1	65.1	88.4
5	86.1	56.3	68.1	88.3
6	86.0	57.2	68.7	88.4
7	86.3	61.3	71.7	88.4
8	86.6	63.4	73.2	88.4
9	86.7	64.6	74.0	88.4
10	86.7	64.9	74.2	88.4
11	86.6	65.1	74.3	88.4
12	86.7	65.2	74.4	88.4

Table 1: Evaluation of our method for identification of empty nodes and their antecedents (12 first iterations).

As one can expect, at each iteration the method extracts graph rewrite rules that introduce empty nodes and non-local relations into syntactic structures, increasing the recall. The performance of the final system (P/R/F_1 = 86.7/65.2/74.4) for the task of identifying non-local dependencies is comparable to the performance of the best model of (Dienes, 2004): P/R/F_1=82.5/70.1/75.8. The PARSEVAL score for the present system (88.4) is, however, higher than the 87.3 for the system of Dienes.

Another effect of the learned transformations is changing node labels of non-terminals, specifically, modifying labels to include Penn functional tags (e.g., changing NP in the input graph in Figure 3 to NP-SBJ in the output graph in Figure 1). In fact, 17% of all learned rewrite rules involved only changing labels of non-terminal nodes. Analysis of the results showed that the system is capable of assigning Penn function tags to constituents produced by Charniak's parser with $F_1 = 91.4$ (we use here the evaluation measure of (Blaheta, 2004): the F_1 score of the precision and recall for assigning function tags to constituents with surface spans correctly identified by Charniak's parser). Comparison to the evaluation results of the function tagging method presented in (Blaheta, 2004) is shown in Table 2.

The present system outperforms the system of Blaheta on semantic tags such as -TMP or -MNR marking temporal and manner adjuncts, respectively, but performs worse on syntactic tags such as -SBJ or -PRD marking subjects and predicatives,

Type	Count	(Blaheta, 2004) P / R / F_1	Here P / R / F_1
All tags	8480	-	93.3 / 89.6 / 91.4
Syntactic	4917	**96.5** / 95.3 / **95.9**	95.4 / **95.5** / 95.5
Semantic	3225	86.7 / 80.3 / 83.4	**89.7** / **82.5** / **86.0**

Table 2: Evaluation of adding Penn Treebank function tags.

respectively. Note that the present method was not specifically designed to add functional tags to constituent labels. The method is not even "aware" that functional tags exists: it simply treats NP and NP-SBJ as different labels and tries to correct labels comparing input and output graphs in the training corpora.

In general, of the 240 graph rewrite rules extracted during the 12 iterations of the method, 25% involved only one graph node in the left-hand side, 16% two nodes, 12% three nodes, etc. The two most complicated extracted rewrite rules involved left-hand sides with ten nodes.

We now switch to the second application of our graph transformation method.

4.2 Semantic role labeling

Put very broadly, the task of semantic role labeling consists in detecting and labeling simple predicates: *Who did what to whom, where, when, how, why*, etc. There is no single definition of a universal set of semantic roles and moreover, different NLP applications may require different specificity of role labels. In this section we apply the graph transformation method to the task of identification of semantic roles as annotated in the Proposition Bank (Palmer et al., 2005), PropBank for short. In PropBank, for all verbs (except copular) of the syntactically annotated sentences of the Wall Street Journal section of the Penn Treebank, semantic arguments are marked using references to the syntactic constituents of the Penn Treebank. For the 49,208 syntactically annotated sentences of the Penn Treebank, the PropBank annotated 112,917 verb predicates (2.3 predicates per sentence on average), with a total of 292,815 semantic arguments (2.6 arguments per predicate on average).

PropBank does not aim at cross-verb semantically consistent labeling of arguments, but rather at annotating the different ways arguments of a verb can be realized syntactically in the corpus, which resulted in the choice of theory-neutral numbered labels (e.g., *Arg0, Arg1,* etc.) for semantic arguments. Figure 2 shows an example of a PropBank annotation (dashed edges).

In this section we address a specific NLP task: identifying and labeling semantic arguments in the output of a syntactic parser. For the example in Figure 2 this task corresponds to adding "semantic" nodes and edges to the syntactic tree.

As before, in order to apply our graph transformation method, we need to encode the available information using graphs. Our encoding of syntactic phrase structure is the same as in Section 4.1 and the encoding of the semantic annotations of PropBank is straightforward. For each PropBank predicate, a new node with attributes type = propbank and label = pred is added. Another node with label = head and nodes for all semantic arguments of the predicate (with labels indicating PropBank argument names) are added and connected to the predicate node. Argument nodes with label ARGM (adjunct) additionally have a feature attribute with values TMP, LOC, etc., as specified in PropBank. The head node and all argument nodes are linked to their respective syntactic constituents, as specified in the PropBank annotation. All introduced semantic edges are marked with the attribute type = propbank.

As before, we used section 02–21 of the PropBank (which annotates the same text as the Penn Treebank) to train our graph transformation system, section 00-01 for development and section 23 for testing. We ran three experiments, taking three different corpora of input graphs:

1. the original syntactic structures of the Penn Treebank containing function tags, empty nodes, non-local dependencies, etc.;

2. the output of Charniak's parser (i.e., bare syntactic trees) on the strings of sections 02–21; and

3. the output of Charniak's parser processed with the graph transformation system described in 4.1.

For all three experiments we used the gold standard syntactic and semantic annotations from the

Iter.	Penn Treebank P	Penn Treebank R	Charniak P	Charniak R	Charniak + P	Charniak + R
1	90.0	70.7	79.5	58.6	79.9	59.1
2	90.7	76.5	81.2	63.9	81.0	64.2
3	90.7	78.1	81.3	65.6	81.1	65.8
4	90.6	78.9	81.4	66.5	81.2	66.7
5	90.5	80.4	81.4	67.0	81.2	68.3
6	90.4	81.2	81.4	68.3	81.1	68.8
7	90.3	81.9	81.3	68.9	81.0	69.3
8	90.3	82.2	81.3	69.3	81.0	69.8
9	90.3	82.5	81.3	69.6	81.0	70.1
10	90.3	82.8	81.4	69.8	81.0	70.3
11	90.3	83.0	81.3	69.9	81.0	70.4
12	90.3	83.2				

Table 3: Evaluation of our method for semantic role identification with Propbank: with Charniak parses and with parses processed by the system of Section 4.1.

Penn Treebank and PropBank as the corpora of output graphs (for the experiment with bare Charniak parses, we dropped function tags, empty nodes and non-local dependencies from the syntactic annotation of the output graphs: we did not want our system to start recovering these annotations, but were interested in the identification of PropBank information alone).

For each of the experiments, we used the corpora of input and output graphs as before, at each iteration extracting 20 rewrite rules with most frequent left-hand sides, applying the rules to the development data to measure the current performance of the system. We stopped the learning in case the performance improvement was less than a threshold and, otherwise, continued the learning loop. As our performance measure we used the F_1 score of precision and recall of the correctly identified and labeled non-empty constituents—semantic arguments.

In all experiments, the learning stopped after 11 or 12 iterations. The results of the evaluation of the system at each iteration on the test section of PropBank are shown in Table 3.

As one may expect, the performance of our semantic role labeler is substantially higher on the gold Penn Treebank syntactic structures than on the parser's output. Surprisingly, however, adding extra information to the parser's output (i.e., processing it with the system of Section 4.1) does not significantly improve the performance of the resulting system.

In Table 4 we compare our system for semantic roles labeling with the output of Charniak's parser to the state-of-the-art system of (Pradhan et al., 2005).

System	P	R	F_1
(Pradhan et al., 2005)	80.9	76.8	78.8
Here	81.0	70.4	75.3

Table 4: Evaluation of our methods for semantic role identification with Propbank (12 first iterations).

While showing good precision, our system performs worse than state-of-the-art with respect to recall. Taking into account the iterative nature of the method and imperfect rule selection criteria (we simply take the most frequent left-hand sides), we believe that it is the rule selection and learning termination condition that account for the relatively low recall values. Indeed, in all three experiments described above the learning loop stops while the recall is still on the rise, albeit very slowly. It seems that a more careful rule selection mechanism and loop termination criteria are needed to address the recall problem.

5 Conclusions

In this paper we argued that encoding diverse and complex linguistic structures as directed labeled graphs allows one to view many NLP tasks as graph transformation problems. We proposed a general method for learning graph transformation from annotated corpora and described experiments with two NLP applications.

For the task of identifying non-local dependencies and for function tagging our general method demonstrates performance similar to the state-of-the-art systems, designed specifically for these tasks. For the PropBank semantic role labeling the method shows a relatively low recall, which can be explained by our sub-optimal "rule of thumb" heuristics (such as selecting 20 most frequent rewrite rules at each iteration of the learning method). We see two ways of avoiding such heuristics. First, one can define and fine-tune the heuristics for each specific application. Second, one can use more informed rewrite rule selection methods, based on graph-based relational learning and frequent subgraph detection algorithms (Cook and Holder, 2000; Yan and Han, 2002). Furthermore, more experiments are required

to see how the details of encoding linguistic information in graphs affect the performance of the method.

Acknowledgements

This research was supported by the Netherlands Organization for Scientific Research (NWO) under project numbers 017.001.190, 220-80-001, 264-70-050, 354-20-005, 600.065.120, 612-13-001, 612.000.106, 612.066.302, 612.069.006, 640.001.501, 640.002.501, and by the E.U. IST programme of the 6th FP for RTD under project MultiMATCH contract IST-033104.

References

Ann Bies, Mark Ferguson, Karen Katz, and Robert MacIntyre. 1995. Bracketing guidelines for Treebank II style Penn Treebank project. Technical report, University of Pennsylvania.

Don Blaheta. 2004. *Function Tagging*. Ph.D. thesis, Brown University.

Eric Brill. 1995. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21(4):543–565.

Richard Campbell. 2004. Using linguistic principles to recover empty categories. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, pages 645–653.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st Meeting of NAACL*, pages 132–139.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Diane J. Cook and Lawrence B. Holder. 2000. Graph-based data mining. *IEEE Intelligent Systems*, 15(2):32–41.

Péter Dienes. 2004. *Statistical Parsing with Non-local Dependencies*. Ph.D. thesis, Universität des Saarlandes, Saarbrücken, Germany.

Daniel Gildea. 2001. *Statistical Language Understanding Using Frame Semantics*. Ph.D. thesis, University of California, Berkeley.

Ana-Maria Giuglea and Alessandro Moschitti. 2006. Semantic role labeling via framenet, verbnet and propbank. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 929–936.

Julia Hockenmaier. 2003. Parsing with generative models of predicate-argument structure. In *Proceedings of the 41st Meeting of ACL*, pages 359–366.

Valentin Jijkoun and Maarten de Rijke. 2004. Enriching the output of a parser using memory-based learning. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 311–318, Barcelona, Spain, July.

Valentin Jijkoun. 2006. *Graph Transformations for Natural Language Processing*. Ph.D. thesis, University of Amsterdam.

Thorsten Joachims. 1999. Making large-scale svm learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT-Press.

Christopher R. Johnson, Miriam R. L. Petruck, Collin F. Baker, Michael Ellsworth, Josef Ruppenhofer, and Charles J. Fillmore. 2003. FrameNet: Theory and Practice. `http://www.icsi.berkeley.edu/~framenet`.

Mark Johnson. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of the 40th meeting of ACL*, pages 136–143.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1).

Sameer Pradhan, Wayne Ward, Kadri Hacioglu, Jim Martin, and Dan Jurafsky. 2005. Semantic role labeling using different syntactic views. In *Proceedings of ACL-2005*.

A. Schürr. 1997. Programmed graph replacement systems. In Grzegorz Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, chapter 7, pages 479–546.

Kristina Toutanova, Aria Haghighi, and Chris Manning. 2005. Joint learning improves semantic role labeling. In *Proceedings of the 43rd Meeting of the Association for Computational Linguistics (ACL)*.

Xifeng Yan and Jiawei Han. 2002. gspan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM)*.