Linköping: Cubic-Time Graph Parsing with a Simple Scoring Scheme

Marco Kuhlmann
Dept. of Computer and Information Science
Linköping University, Sweden
marco.kuhlmann@liu.se

Abstract

We turn the Eisner algorithm for parsing to projective dependency trees into a cubic-time algorithm for parsing to a restricted class of directed graphs. To extend the algorithm into a data-driven parser, we combine it with an edge-factored feature model and online learning. We report and discuss results on the SemEval-2014 Task 8 data sets (Oepen et al., 2014).

1 Introduction

This paper describes the system that we submitted to the closed track of the SemEval-2014 Task on Broad-Coverage Semantic Dependency Parsing (Oepen et al., 2014).[1] However, the main contribution of the paper is not the system as such (which had the lowest score among all systems submitted to the task), but the general approach for which it is a proof of concept.

Graphs support natural representations of linguistic structure. For this reason, algorithms that can learn, process and transform graphs are of central importance to language technology. Yet, most of the algorithms that are used in natural language processing today focus on the restricted case of trees, and do so for a reason: Computation on general graphs is hard or even intractable, and efficient processing is possible only for restricted classes (cf. Courcelle and Engelfriet (2012)). The task then is to identify classes of graphs that are both expressive enough to cover the linguistic data, and restricted enough to facilitate efficient processing.

This work is licensed under a Creative Commons Attribution 4.0 International Licence. Page numbers and proceedings footer are added by the organisers. Licence details: http://creativecommons.org/licenses/by/4.0/

[1] https://github.com/liu-nlp/gamma

This paper shows that there are graphs that satisfy both of these desiderata. Our system is based on a new algorithm for parsing to a restricted class of directed graphs (Section 2). This class is restricted in so far as our algorithm runs in cubic time with respect to the length of the sentence; it thus has the same asymptotic complexity as parsing with context-free phrase structure grammars. The class of graphs defined by our algorithm is also expressive, in so far that it covers more than 98% of the SemEval data.

To demonstrate that our parsing algorithm can be turned into a practical system, we combine it with two techniques taken straight from the literature on data-driven syntactic dependency parsing:

- an edge-factored scoring model, as it has been used as the core of practical parsers since the seminal work of McDonald et al. (2005), and
- online learning using the structured perceptron, in the style of Collins (2002).

State-of-the-art parsers use considerably more advanced (and computationally more demanding) techniques, and therefore our system cannot be expected to deliver competitive results. (Its results on the SemEval data are reported in Section 4.) Instead, the main point of our contribution to the SemEval Task is to provide evidence that research on classes of graphs that balance linguistic coverage and parsing efficiency holds a lot of potential.

2 Parsing Algorithm

We start the description of our system with the description of our cubic-time parsing algorithm. The remaining components of our system will be described in Section 3.

Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014), pages 395–399,
Dublin, Ireland, August 23-24, 2014.

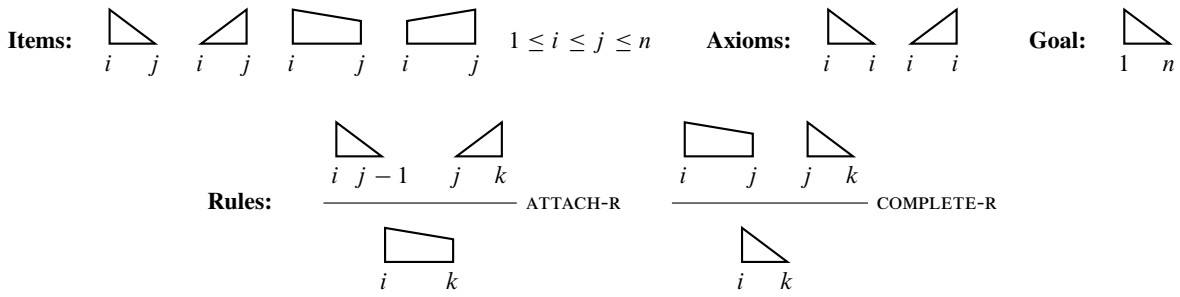

Figure 1: The Eisner algorithm for building the packed forest of all projective dependency trees with n nodes. Only the rightward versions of ATTACH and COMPLETE are shown here.

2.1 The Eisner Algorithm

We recall the algorithm for projective dependency parsing by Eisner and Satta (1999). The declarative specification of this algorithm in terms of a deduction system (Shieber et al., 1995) is given in Figure 1. The algorithm uses four types of items, ◺, ◿, ▭, and ▱, and two types of inference rules called ATTACH and COMPLETE. These rules can be interpreted as operations on graphs: An ATTACH rule *concatenates* two graphs and adds one of two possible edges—from the left endpoint of the first graph to the right endpoint of the second graph, or vice versa. Similarly, a COMPLETE rule *fuses* two graphs by unifying the right endpoint of the first with the left endpoint of the second. The algorithm by Eisner and Satta (1999) produces a compact representation of the set of all dependency graphs over the input sentence that can be built using these operations. This is exactly the set of projective dependency trees for the input sentence.

2.2 The Graph-Parsing Algorithm

To parse to dependency graphs rather than trees, we modify the Eisner algorithm as follows:

- We give up the distinction between ▭ and ▱. This distinction is essential for ensuring that the parser builds a tree. Since our goal is to parse to graphs, we do not need it.
- We allow ATTACH to add one, zero, or several edges. This modification makes it possible to parse graphs with reentrancies (several incoming edges) and isolated nodes.

To implement the first modification, we introduce a new type of items, ▭, that subsumes ▭ and ▱.

To implement the second modification, we parametrize the ATTACH rule by a set ω that specifies the edges that are added during the concatenation. We refer to the left and right endpoints of a graph as its *ports* and number the ports of the antecedents of the ATTACH rule left-to-right from 1 to 4. A set ω then takes the form

$$\omega \subseteq \big(\{1,2\} \times \{3,4\}\big) \cup \big(\{3,4\} \times \{1,2\}\big) .$$

The rule ATTACH_ω adds an edge $u \rightarrow v$ if and only if u and v are nodes corresponding to ports s and t, respectively, and $(s,t) \in \omega$. For example, the ATTACH rule in Figure 1 is specified by the set $\omega = \{(1,4)\}$: it adds one edge, from the left endpoint of the graph corresponding to the first antecedent to the right endpoint of the other graph.

The complete parsing algorithm is specified in Figure 2, where the rule CONC (for *concatenation*) corresponds to the two conflated ATTACH rules and FUSE corresponds to the two COMPLETE rules. Inspecting the specification, we find that the algorithm runs in time $O(mn^3)$ where n is the number of nodes and m is the number of concatenation rules. Note that, because each concatenation rule is determined by a set ω as defined above, each parser in our framework can use at most $2^8 = 256$ different CONC rules.[2]

3 Data-Driven Parsing

We now extend our parsing algorithm into a simple parser for data-driven parsing. We cast parsing as an optimization problem over a parametrized scoring function: Given a sentence x we compute

$$\hat{y} = \arg\max_{y \in \mathcal{Y}(x)} s(x, y) \qquad (1)$$

where $\mathcal{Y}(x)$ is the set of candidate graphs for x and the scoring function is decomposed as $s(x, y) = \theta \cdot f(x, y)$. The function f returns a high-dimensional feature vector that describes characteristic properties of the sentence–graph pair (x, y), and the vector θ assigns to each feature a weight.

[2]This is because a set ω specifies up to $2 \times 2 + 2 \times 2 = 8$ different concatenation operations.

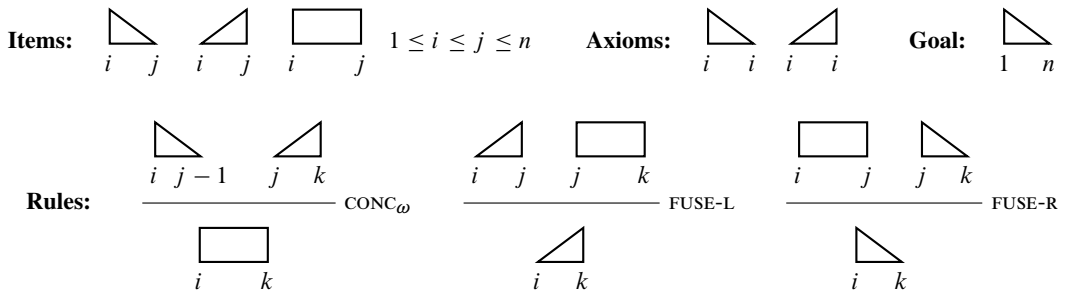

Figure 2: The parsing algorithm used in this paper. The concatenation rules (CONC) are parametrized with respect to an edge specification ω (see Section 2.2).

3.1 Candidate Graphs

Our set of candidate graphs is the set of all graphs that can be built using the operations of our parsing algorithm. The size of this set and hence the maximal coverage of our parser is determined by the set of CONC rules: The more different concatenation operations we use, the more graphs we can build. At the same time, increasing the number of operations also increases the runtime of our parser. This means that we need to find a good trade-off between coverage and parsing efficiency.

To obtain upper bounds on the coverage of our parser we compute, for each graph G in the SemEval test data, a graph $\tilde{G}$ that maximizes the set of edges that it has in common with G. This can be done using a Viterbi-style variant of our parsing algorithm that scores an item by the number of edges that it has in common with G. The results are reported in Table 1. As we can see, our approach has the potential to achieve more than 98% labelled recall (LR) on all three representation types used in the task. This figure is obtained for the full set of concatenation operations. For our submission we chose to optimize for parsing speed and used a parser with a reduced set of only three operations:

$$\omega_1 = \{(1,4)\}\,, \quad \omega_2 = \{(4,1)\}\,, \quad \omega_3 = \{\}\,.$$

These are the two operations that correspond to the ATTACH rules of the algorithm by Eisner and Satta (ω_1, ω_2), together with the operation that concatenates two graphs without adding any edges at all (ω_3). The latter is required to produce graphs where a node has no incoming edges. As can be seen in Table 1, the upper bounds for the reduced set of operations are still surprisingly high when measured in terms of LR: 95.70% for DM, 93.06% for PAS, and 93.51% for PCEDT. However, there is a significant loss when coverage is measured in terms of labelled exact match (LM).

	DM	PAS	PCEDT
full	98.25 / 75.74	98.13 / 69.81	98.19 / 83.23
reduced	95.70 / 52.15	93.06 / 23.66	93.51 / 54.75

Table 1: Upper bounds for recall (LR/LM) on the test data for two different sets of operations.

3.2 Scoring Function

We use the same features as in the first-order model implemented in the MSTParser system for syntactic dependency parsing (McDonald et al., 2005).[3] Under this model, the feature vector for a dependency graph is the sum of the feature vectors of its edges, which take into account atomic features such as the word forms and part-of-speech tags of the tokens connected by the edge, the length of the edge, the edge label, as well as combinations of those atomic features. To set the feature weights we use averaged perceptron training in the style of Collins (2002).

3.3 Top-Node Tagger

The final component in our system is a simple tagger that is used to annotate the output of our parser with information about *top nodes* (as defined in the task's data format). It is based on Matthew Honnibal's part-of-speech tagger[4] and uses features based on the word form and part-of-speech of the node to be tagged, as well as the labels of the edges incident to that node; these features were selected based on tagging accuracy with the recommended development train/dev-split. The tagger is a sequence model without global constraints; in particular, it does not enforce unique top nodes. Tagging accuracy on the final test set was 98.50% for DM, 99.21% for PAS, and 99.94% for PCEDT.

[3] http://sourceforge.net/projects/mstparser/
[4] http://honnibal.wordpress.com/

	DM			PAS			PCEDT		
	LP	LR	LF	LP	LR	LF	LP	LR	LF
Baseline	83.20%	40.73%	54.68%	88.34%	35.74%	50.89%	74.82%	62.08%	67.84%
Linköping	78.54%	78.05%	78.29%	76.16%	75.55%	75.85%	60.66%	64.35%	62.45%
Task average	84.21%	81.29%	82.69%	87.95%	83.57%	85.65%	72.17%	68.44%	70.21%
Peking	90.27%	88.54%	89.40%	93.44%	90.69%	92.04%	78.75%	73.96%	76.28%

Table 2: Labelled precision (LP), labelled recall (LR), and labelled F1 (LF) scores of our own system (Linköping) and three points of comparison on the SemEval-2014 Task 8 test data: baseline, task average, and the best-performing system from Peking University (Du et al., 2014).

4 Experiments

We report experimental results on the SemEval data sets (closed track). We trained one parser for each representation (DM, PAS, PCEDT). Averaged perceptron training can be parametrized by the number N of iterations over the training data; to determine the value of this parameter, for each representation type and each $1 \leq N \leq 10$ we trained a development system using the recommended development train/dev-split and selected that value of N which gave the highest accuracy on the held-out data. The selected values and the number of (binary) features in the resulting systems are reported in Table 3. Training took around 8 minutes per iteration on an iMac computer (Late 2013, 3,4 GHz Intel Core i5) with a 6 GB Java heap size.

4.1 Results

Table 2 reports the labelled precision (LP) and labelled recall (LR) of our system on the final test data. Compared to the tree-based baseline, our system has substantially lower precision (between 4.66 and 14.16 points) but substantially higher recall (between 2.27 and 39.81 points). Compared to the top-scoring system, our system is way behind in terms of both scores (11.11–16.19 points). The scores of our system are also substantially below the task average, which resulted in it being ranked last of all six systems participating in the closed track. Given these results, we have refrained from doing a detailed error analysis. It may be interesting to note, however, that our system is the only one in the task for which labelled F1 is higher on the DM data than on the PAS data.

	DM	PAS	PCEDT
# iterations	4	1	9
# features	7.3M	8.7M	8.1M

Table 3: Characteristics of the trained models.

4.2 Discussion

The comparatively low scores of our system do not come unexpected. Our parser uses a very simple scoring model and learning method, whereas even the baseline relies on a state-of-the-art syntactic dependency parser (Bohnet, 2010). Also, we did not do any feature engineering (on the parser), but just used the feature extraction procedure of MSTParser. Regarding both of these points, the potential for improving the system is apparent. Finally, our post-hoc prediction of top nodes is extremely simplistic. It would have been much more desirable to integrate this prediction into the parser, for example by adding virtual incoming dependencies to all top nodes. However, preliminary experiments showed that this particular strategy had a severely negative impact on coverage.

5 Conclusion

We have presented a new algorithm for parsing to a restricted class of digraphs and shown how to extend this algorithm into a system for data-driven dependency parsing. Our main goal was to show that it is possible to develop algorithms for direct parsing to directed graphs that are both efficient and achieve good coverage on practical data sets: Our algorithm runs in cubic time in the length of the sentence, and has more than 98% coverage on each of the three data sets.

Our future work will address both theoretical and practical issues. On the theoretical side, we feel that it is important to obtain a better understanding of the specific graph-structural properties that characterise the linguistic data. Our parser provides an operational definition of a class of graphs (those graphs that can be built by the parser); it would be more satisfying to obtain a declarative characterisation that does not depend on a specific algorithm. Such a characterisation would be interesting even for a restricted set of operations.

On the practical side, we would like to extend our approach into a more competitive system for semantic dependency parsing. In particular, we would like to use a more powerful scoring function (incorporating second- and third-order features) and a more predicative learning method (such as max-margin training).

Acknowledgements

We thank the two anonymous reviewers of this paper for their detailed and constructive comments.

References

Bernd Bohnet. 2010. Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING)*, pages 89–97, Beijing, China.

Michael Collins. 2002. Discriminative training methods for Hidden Markov Models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1–8, Philadelphia, USA.

Bruno Courcelle and Joost Engelfriet. 2012. *Graph Structure and Monadic Second-Order Logic*, volume 138 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press.

Yantao Du, Fan Zhang, Weiwei Sun, and Xiaojun Wan. 2014. Peking: Profiling syntactic tree parsing techniques for semantic graph parsing. In *Proceedings of the Eighth International Workshop on Semantic Evaluation (SemEval 2014)*, Dublin, Republic of Ireland.

Jason Eisner and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and Head Automaton Grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 457–464, College Park, MD, USA.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 91–98, Ann Arbor, USA.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajič, Angelina Ivanova, and Yi Zhang. 2014. SemEval 2014 Task 8: Broad-coverage semantic dependency parsing. In *Proceedings of the Eighth International Workshop on Semantic Evaluation (SemEval 2014)*, Dublin, Republic of Ireland.

Stuart M. Shieber, Yves Schabes, and Fernando Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1–2):3–36.