Shift-Reduce Dependency DAG Parsing

Kenji Sagae[†]
Institute for Creative Technologies
University of Southern California
13274 Fiji Way
Marina del Rey, CA 90292
sagae@ict.usc.edu

Jun'ichi Tsujii
Department of Computer Science
University of Tokyo
School of Computer Science
University of Manchester
National Center for Text Mining
tsujii@is.s.u-tokyo.ac.jp

Abstract

Most data-driven dependency parsing approaches assume that sentence structure is represented as trees. Although trees have several desirable properties from both computational and linguistic perspectives, the structure of linguistic phenomena that goes beyond shallow syntax often cannot be fully captured by tree representations. We present a parsing approach that is nearly as simple as current data-driven transition-based dependency parsing frameworks, but outputs directed acyclic graphs (DAGs). We demonstrate the benefits of DAG parsing in two experiments where its advantages over dependency tree parsing can be clearly observed: predicate-argument analysis of English and syntactic analysis of Danish with a representation that includes long-distance dependencies and anaphoric reference links.

1 Introduction

Natural language parsing with data-driven dependency-based frameworks has received an increasing amount of attention in recent years (McDonald et al., 2005; Buchholz and Marsi, 2006; Nivre et al., 2006). Dependency representations directly reflect word-to-word relationships in a dependency graph, where the words in a sentence are the nodes, and labeled edges correspond to head-dependent syntactic relations. In addition to being inherently lexicalized, dependency analyses can be generated efficiently and have been show to be useful in a variety of practical tasks, such as question answering (Wang et al., 2007), information extraction in biomedical text (Erkan et al., 2007; Saetre et al, 2007) and machine translation (Quirk and Corston-Oliver, 2006).

However, despite rapid progress in the development of parsers for several languages (Nivre et al., 2007) and algorithms for more linguistically adequate non-projective structures (McDonald et al., 2005; Nivre and Nilsson, 2006), most of the current data-driven dependency parsing approaches are limited to producing only dependency trees, where each word has exactly one head. Although trees have desirable properties from both computational and linguistic perspectives, the structure of linguistic phenomena that goes beyond shallow syntax often cannot be fully captured by tree representations. Well-known linguistically-motivated dependency-based syntactic frameworks, such as Hudson's Word Grammar (Hudson, 1984), recognize that to represent phenomena such as relative clauses, control relations and other long-distance dependencies, more general graphs are needed. Hudson (2005) illustrates his syntactic framework with the analysis shown in figure 1. In this example, the arcs above the sentence correspond to a typical dependency tree commonly used in dependency parsing. It is clear, however, that the entire dependency structure is not a tree, but a directed acyclic graph (DAG), where words may have one or more heads. The arcs below the sentence represent additional syntactic dependencies commonly ignored in current dependency parsing approaches that are limited to producing tree

[†] This work was conducted while the author was at the Computer Science Department of the University of Tokyo.

Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008), pages 753–760
Manchester, August 2008

structures. There are several other linguistic phenomena that cannot be represented naturally with dependency trees, but can easily be represented with dependency DAGs, including anaphoric reference and semantically motivated predicate-argument relations. Although there are parsing approaches (often referred to as *deep parsing* approaches) that compute DAG dependency structures, this is usually done through more complex lexicalized grammar formalisms (such as HPSG, CCG and LFG) and unification operations with tree-based parsing algorithms.

We introduce a new data-driven framework for dependency parsing that produces dependency DAGs directly from input strings, in a manner nearly as simple as other current transition-based dependency parsers (Nivre et al., 2007) produce dependency trees. By moving from tree structures to DAGs, it is possible to use dependency parsing techniques to address a wider range of linguistic phenomena beyond surface syntax. We show that this framework is effective and efficient in analysis of predicate-argument dependencies represented as DAGs, and in syntactic parsing using DAGs that include long-distance dependencies, gapping dependents and anaphoric reference information, in addition to surface syntactic dependents.

Our parsing framework, based on shift-reduce dependency parsing, is presented in section 2. Experiments and results are presented and discussed in section 3. We review related work in section 4, and conclude in section 5.

2 A shift-reduce parsing framework for dependency DAGs

One of the key assumptions in both graph-based (McDonald et al., 2005) and transition-based (Nivre, 2004; Nivre and Nilsson, 2006) approaches to data-driven dependency parsing is that the dependency structure produced by the parser is a tree, where each word has exactly one head (except for a single root word, which has no head in the sentence). This assumption, of course, has to be abandoned in dependency DAG parsing. McDonald et al. (2006) point out that, while exact inference is intractable if the tree constraints are abandoned in their graph-based parsing framework, it is possible to compute more general graphs (such as DAGs) using approximate inference, finding a tree first, and adding extra edges that increase the graph's overall score. Our approach, in contrast, extends shift-reduce (transition-based) approaches, finding a DAG directly. Because data-driven shift-reduce dependency parsing is based on local decisions (informed by rich a rich feature set), the additional computational cost of computing DAGs instead of trees is small in practice, as we will show.

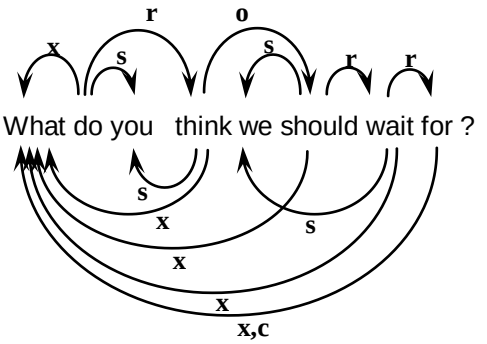

Figure 1: Word Grammar dependency graph (Hudson, 2005). Key for edge types: complement (c), object (o), sharer/xcomp (r), subject (s), and extractee (x).

We first describe how the basic shift-reduce bottom-up dependency parsing algorithm described by Nivre (2004) can be modified to allow multiple heads per word. We then explore the same type of modification to Nivre's arc-eager algorithm, which is a variant of the basic shift-reduce algorithm where arcs can be created at the first opportunity. Like their tree counterparts, our algorithms for dependency DAGs produce only projective structures, assuming that projectivity for DAGs is defined in much the same way as for trees. Informally, we define a projective DAG to be a DAG where all arcs can be drawn above the sentence (written sequentially in its original order) in a way such that no arcs cross and there are no covered roots (although a *root* is not a concept associated with DAGs, we borrow the term from trees to denote words with no heads in the sentence). However, non-projectivity is predictably more wide-spread in DAG representations, since there are at least as many arcs as in a tree representation, and often more, including arcs that represent non-local relationships. We then discuss the application of pseudo-projective transformations (Nivre and Nilsson, 2005) and an additional arc-reversing transform to dependency DAGs. Using a shift-reduce algorithm that allows multiple heads per word and pseudo-projective transformations to-

gether forms a complete dependency DAG parsing framework.

2.1 Basic shift-reduce parsing with multiple heads

The basic bottom-up left-to-right dependency parsing algorithm described by Nivre (2004) keeps a list of tokens (initialized to contain the input string) and a stack (initialized to be empty), and allows three types of actions: (1) *shift*, which removes the next item from the input list and pushes it onto the top of the stack; (2) *left-reduce*, which pops the top two items from the stack, creates a left-arc between the words they represent in the sentence, and push the top item (which is now the head of the item previously below it) back onto the stack; and (3) *right-reduce*, which works in the same way as *left-reduce*, but creates a right-arc instead, and pushes back onto the stack the item that was below the top item on the stack (which is now the head of the item previously on top of the stack)[1]. New dependency edges (or arcs) are only created by reduce actions, which are constrained so that they can only be applied to create a head-dependent pair where the dependent has already found all of its own dependents (if any). This is necessary because once a word is assigned a head it is popped from the stack and never visited again, since each word has only one head. This constraint, responsible for the parser's bottom-up behavior, should be kept in mind, as it is relevant in the design of the multiple-head parsing algorithm below.

To allow words to have multiple heads, we first need to create two new parser actions that create dependency arcs without removing the dependent from further consideration for being a dependent of additional heads. The first new action is *left-attach*, which creates a left dependency arc attaching the top two items on the stack, making the top item the head of the item immediately below, as long as a right arc between the two items does not already exist. This action is similar to *left-reduce*, except that neither item is removed from the stack (no reduction occurs). The second new action, *right-attach*, includes one additional final step: first, it creates a right dependency arc between the top two items on the stack (as long as a left arc between the two items does not already exist), making the top item a dependent of the item immediately below;

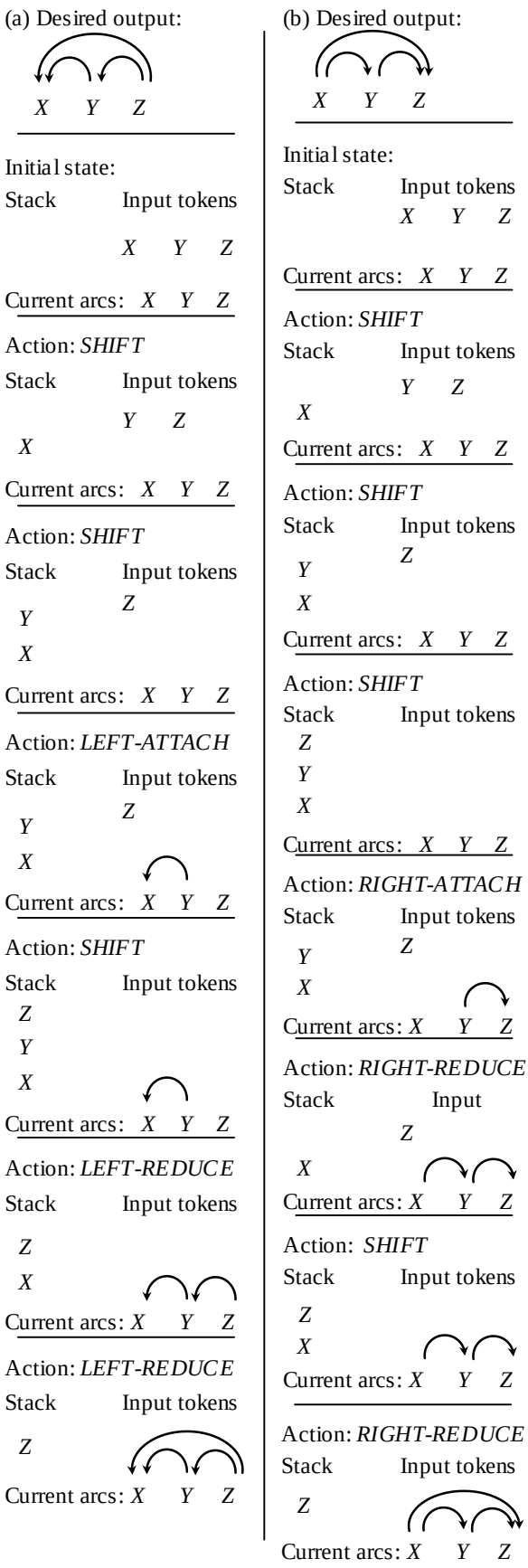

Figure 2: Example of how the basic algorithm builds dependencies with multiple heads.

[1] Like Nivre (2004), we consider the direction of the dependency arc to be from the head to the dependent.

and, as a second step, it pops the top item on the stack (newly made a dependent), and *places it back on the list of input words*. This second step is necessary because of the constraint that words can only be made dependents once all of its own dependents have been found. The behavior of the algorithm is illustrated in figure 2, where (a) shows an application of *left-attach*, and (b) shows an application of *right-attach*. In (b), we note that without placing the dependent in the *right-attach* action (*Z*) back on the input list, the dependency between *X* and *Y* could not be created. If we abandon the algorithm's bottom-up behavior, it is possible to modify the parser actions so that it is not necessary to place items back in the input list. This is discussed in section 2.2.

In summary, the algorithm has each of the three actions from the tree-based algorithm (*shift*, *right-reduce*, and *left-reduce*), and two additional actions that allow words to be dependents of more than one head (*right-attach* and *left-attach*). Although the algorithm as described so far builds unlabeled structures, the extension to labeled structures is straightforward: any action that results in a new arc being created must also choose a label for the arc. Another way to accomplish the same goal is to have a copy of each arc-producing action for each possible arc label. This is the same labeling extension as in the algorithm for trees. Finally, we note that the algorithm does not explicitly prevent multiple arcs (with the same direction) from being created between the same two words. In the unlabeled case, such a constraint can be easily placed on arc-producing actions. In the labeled case, however, it is useful to allow arcs with different labels to link the same two words[2].

2.2 Arc-eager shift-reduce parsing with multiple heads

Nivre's arc-eager algorithm was designed to build dependencies at the first opportunity, avoiding situations where items that form a chain of right arcs all have to be placed on the stack before any structure is built, as in figure 2(b) for example. This is done by creating dependencies not between the top two items on the stack, but between the single top item on the stack and the next word on the input list, resulting in a hybrid bottom-up/top-down strategy. A similar idea can result in an algorithm for dependencies that allow multiple heads per word, but in this case the resulting algorithm is not as similar to the arc-eager algorithm for trees as the algorithm in section 2.1 is to its tree-based counterpart.

The projective DAG arc-eager algorithm has four actions, each corresponding to one action of the tree-based algorithm, but only the shift action is the same as in the tree based algorithm. The four actions in the new algorithm are: (1) *shift*, which removes the next token from the input string and pushes it onto the top of the stack; (2) *reduce*, which pops the stack, removing only its top item, as long as that item has at least one head (unlike in the tree-based algorithm, however, the algorithm may not reduce immediately when an item that has a head is on the top of the stack); (3) *left-arc*, which creates a left dependency arc between the word on top of the stack and the next token in the input string, where the token in the input string is the head and the item on the stack is the dependent (the stack and input list are left untouched), as long as a right arc does not already exist between the two words; and (4) *right-arc*, which creates a right dependency arc between the word on top of the stack and the next token in the input list, where the item on the stack is the head and the token in the input list is the dependent (again, the stack and input list are left untouched), as long as a left arc does not already exist between the two words.

Like the algorithm in section 2.1, this algorithm can easily be extended to produce labeled structures, and it also allows multiple edges (with the same direction) between the same two words.

2.3 Graph transformations for DAGs

Although the algorithms presented in sections 2.1 and 2.2 can produce dependency structures where a word may have more than one head, they are of limited interest on their own, since they can only produce projective structures, and many of the interesting linguistic phenomena that can be represented with DAGs cannot be represented with projective DAGs. Fortunately, the pseudo-projective transformations (Nivre and Nilsson, 2006) used in tree-based dependency parsing can easily be applied to DAGs. These transformations consist of identifying specific non-projective arcs, and moving their heads up towards the root, making them projective. The process also involves creating markings on the labels of the edges involved, so that the transformations are (mostly) reversible. Because non-

[2] This means that the structures produced by the algorithm are technically not limited to projective DAGs, since they can also be projective labeled multi-digraphs.

projectivity is more common in linguistically interesting DAGs, however, the transform/detransform process may be more lossy than it is when applied to trees. This, of course, varies according to specific DAGs used for representing specific phenomena. For pseudo-transformations to work well, we must allow multiple differently labeled arcs between the same two words (which, as mentioned before, the algorithms do). Combining the algorithm in sections 2.1 or 2.2 with pseudo-projective parsing, we can use DAG training data and produce DAG output in the overall parsing framework.

An alternative to using pseudo-projective transformations is to develop an algorithm for DAG parsing based on the family of algorithms described by Covington (2001), in the same way the algorithms in sections 2.1 and 2.2 were developed based on the algorithms described by Nivre (2004). Although this may be straightforward, a potential drawback of such an approach is that the number of parse actions taken in a Covington-style algorithm is always quadratic on the length of the input sentence, resulting in parsers that are more costly to train and to run (Nivre, 2007). The algorithms presented here, however, behave identically to their linear runtime tree counterparts when they are trained with graphs that are limited to tree structures. Additional actions are necessary only when words with more than one head are encountered. For data sets where most words have only one head, the performance the algorithms described in sections 2.1 and 2.2 should be close to that of shift-reduce projective parsing for dependency trees. In data sets where most words have multiple heads (resulting in higher arc density), the use of a Covington-style algorithm may be advantageous, but this is left as an area of future investigation.

In addition to pseudo-projective transformations, an additional transformation that is useful in DAG parsing is *arc reversal*. This consists of simply reversing the direction of an edge, adding a special mark to its label to indicate that its direction has been reversed. Detransformation is trivial and can be done with perfect accuracy, since it can be accomplished by simply reversing the arcs marked as reversed. This transformation is useful in cases where structures are mostly in DAG form, but may sometimes contain cycles. Arc reversal can be used to change the direction of an arc in the cycle, making the previously cyclic structure a DAG, which can be handled in the framework presented here.

3 Experiments

To investigate the efficacy of our DAG parsing framework on natural language data annotated with dependency DAGs, we conducted two experiments. The first uses predicate-argument dependencies taken from the HPSG Treebank built by Miyao et al. (2004) from the WSJ portion of the Penn Treebank. These predicate-argument structures are, in general, dependency graphs that do contain cycles (although infrequently), and also contain a large number of words with multiple heads. Since the predicate-argument dependencies are annotated explicitly in the HPSG Treebank, extracting a corpus of gold-standard dependency graphs is trivial. The second experiment uses the Danish Dependency Treebank, developed by Kromann (2003). This treebank follows a dependency scheme that includes, in addition to standard grammatical relations commonly used in dependency parsing, long-distance dependencies, gapping dependents, and anaphoric reference links. As with the HPSG predicate argument data, a few structures in the data contain cycles, but most of the structures in the treebank are DAGs. In the experiments presented below, the algorithm described in section 2.1 was used. We believe the use of the arc-eager algorithm described in section 2.2 would produce similar results, but this is left as future work.

3.1 Learning component

The DAG parsing framework, as described so far, must decide when to apply each appropriate parser action. As with other data-driven dependency parsing approaches with shift-reduce algorithms, we use a classifier to make these decisions. Following the work of Sagae and Tsujii (2007), we use maximum entropy models for classification. During training, the DAGs are first projectivized with pseudo-projective transformations. They are then processed by the parsing algorithm, which records each action necessary to build the correct structure in the training data, along with their corresponding parser configurations (stack and input list contents). From each of these parser configurations, a set of features is extracted and used with the correct parsing action as a training example for the maximum entropy classifier. The specific features we used in both experiments are the same features described by Sagae and Tsujii, with the following two changes: (1) the addition of a feature that indicates whether an arc already exists between the top two items on the stack, or the top item on

the stack and the next item on the input list, and if so, what type of arc (direction and label); and (2) we did not use lemmas, morphological information or coarse grained part-of-speech tags. For the complete list of features used, please see (Sagae and Tsujii, 2007).

During run-time, the classifier is used to determine the parser action according to the current parser configuration. Like Sagae and Tsujii, we use a beam search instead of running the algorithm in deterministic mode, although we also report deterministic parsing results.

3.2 Predicate-argument analysis

The predicate-argument dependencies extracted from the HPSG Treebank include information such as extraction, raising, control, and other long-distance dependencies. Unlike in structures from PropBank, predicate-argument information is provided for nearly all words in the data. Following previous experiments with Penn Treebank WSJ data, or data derived from it, we used sections 02-21 as training material, section 22 for development, and section 23 for testing. Only the predicate-argument dependencies were used, not the phrase structures or other information from the HPSG analyses. Part-of-speech tagging was done separately using a maximum entropy tagger (Tsuruoka and Tsujii, 2005) with accuracy of 97.1%.

Cycles were eliminated from the dependency structures using the arc reversal transform in the following way: for each cycle detected in the data, the shortest arc in the cycle was reversed until no cycles remained. We applied pseudo-projective transformation and detransformation to determine how much information is lost in this process. By detransforming the projective graphs generated from gold-standard dependencies, we obtain labeled precision of 98.1% and labeled recall of 97.7%, which is below the accuracy expected for detransformation of syntactic dependency trees, but still within a range we considered acceptable. This represents an upper-bound for the accuracy of the DAG parser (including the arc-reversal and pseudo-projective transformations, and the algorithm described in section 2.1).

Table 1 shows the results obtained with our DAG parsing framework in terms of labeled precision, recall and F-score (89.0, 88.5 and 88.7, respectively). For comparison, we also show previously published results obtained by Miyao and Tsujii (2005), and Sagae et al. (2007), which used the same data, but obtained the predicate-argument analyses using an HPSG parser. Our results are very competitive, at roughly the same level as the best previously published results on this data set, but obtained with significantly higher speed. The parser took less than four minutes to process the test set, and pseudo-projective and arc-reversal detransformation took less than one minute in standard hardware (a Linux workstation with a Pentium 4 processor and 4Gb of RAM). Sagae et al. (2007) reported that an HPSG parser took about 20 minutes to parse the same data. Our results were obtained with a beam width of 150 parser states. Running the parser with a beam width of 1 (a single parser state), emulating the deterministic search used by Nivre (2004), resulted in numerous parse failures (the end of the input string is reached, and no further dependency arcs are created) in the development set, and therefore very low dependency recall (90.1 precision and 36.2 recall on development data). Finally, in table 1 we also show results obtained with standard bottom-up shift-reduce dependency parsing for trees, using the parser described in (Sagae and Tsujii, 2007). To train the dependency tree parser, we transformed the DAG predicate-argument structures into trees by removing arcs. Arcs were selected for removal as follows: for each word that had more than one head, only the arc between the word and its closest head (in linear distance in the sentence) was kept. Although this strategy still produces dependency analyses with relatively high F-score (87.0), recall is far lower than when DAG parsing is used, and the tree parser has no mechanism for capturing some of the structures captured by the DAG parser.

Parser	Precision	Recall	F-score
DAG-beam	**89.0**	**88.5**	**88.7**
Tree only	89.8	84.3	87.0
Sagae et al.	88.5	88.0	88.2
Miyao & Tsujii	85.0	84.3	84.6

Table 1: Results from experiments with HPSG predicate-argument dependencies (*labeled* precision, recall and F-score). Our results are denoted by *DAG-beam* and *tree only*, and others are previously published results using the same data.

3.3 Danish Dependency Treebank experiments

Our experiments with the Danish Dependency Treebank followed the same setup as described for the HPSG predicate-argument structures. The accuracy of pseudo-projective transforma-

tion and detransformation was higher, at 99.4% precision and 98.8% recall. To divide the data into training, development and test sections, we followed the same procedure as McDonald et al. (2006), who used the same data, so our results could be compared directly (a small number of graphs that contained cycles was discarded, as done by McDonald et al.).

Our results are shown in table 2 (unlabeled precision and recall are used, for comparison with previous work, in addition to labeled precision and recall), along with the results obtained by McDonald et al., who used an approximate inference strategy in a graph-based dependency parsing framework, where a dependency tree is computed first, and arcs that improve on the overall graph score are added one by one. As in the previous section, we also include results obtained with tree-only parsing. Obtaining tree structures from the Danish Dependency Treebank is straightforward, since anaphoric reference and long-distance dependency arcs are marked as such explicitly and can be easily removed.

In addition to overall results, we also measured the parser's precision and recall on long-distance dependencies and anaphoric reference. On long-distance dependencies the parser had 83.2 precision and 82.0 recall. On anaphoric reference links the parser has 84.9 precision and 84.4 recall. Although these are below the parser's overall accuracy figures, they are encouraging results. Finally, unlike with the HPSG predicate-argument structures, using a beam width of 1 reduces precision and recall by only about 1.5.

Parser	Precision	Recall	F-score
DAG-beam	**87.3**	**87.1**	**87.2**
Tree only	87.5	82.7	85.0
McDonald et al.	86.2	84.9	85.6
DAG-labeled	82.7	82.2	82.4

Table 2: Results from experiments with the Danish Dependency Treebank. Precision, recall and F-score for the first three rows are for *unlabeled* dependencies. The last row, *DAG-labeled*, shows our results in *labeled* precision, recall and F-score (not directly comparable to other rows).

4 Related work

The work presented here builds on the dependency parsing work of Nivre (2004), as discussed in section 2, on the work of Nivre and Nilsson (2006) on pseudo-projective transformations, and on the work of Sagae and Tsujii (2007) in using a beam search in shift-reduce dependency parsing using maximum entropy classifiers. As mentioned before, McDonald et al. (2006) presented an approach to DAG parsing (that could also easily be applied to cyclic structures) using approximate inference in an edge-factored dependency model starting from dependency trees. In their model, the addition of extra arcs to the tree was learned with the parameters to build the initial tree itself, which shows the power and flexibility of approximate inference in graph-based dependency models.

Other parsing approaches that produce dependency graphs that are not limited to tree structures include those based on linguistically-motivated lexicalized grammar formalisms, such as HPSG, CCG and LFG. In particular, Clark et al. (2002) use a probabilistic model of dependency DAGs extracted from the CCGBank (Hockenmeier and Steedman, 2007) in a CCG parser that builds the CCG predicate-argument dependency structures following the CCG derivation, not directly through DAG parsing. Similarly, the HPSG parser of Miyao and Tsujii (2005) builds the HPSG predicate-argument dependency structure following unification operations during HPSG parsing. Sagae et al. (2007) use a dependency parsing combined with an HPSG parser to produce predicate-argument dependencies. However, the dependency parser is used only to produce a dependency tree backbone, which the HPSG parser then uses to produce the more general dependency graph. A similar strategy is used in the RASP parser (Briscoe et al., 2006), which builds a dependency graph through unification operations performed during a phrase structure tree parsing process.

5 Conclusion

We have presented a framework for dependency DAG parsing, using a novel algorithm for projective DAGs that extends existing shift-reduce algorithms for parsing with dependency trees, and pseudo-projective transformations applied to DAG structures.

We have demonstrated that the parsing approach is effective in analysis of predicate-argument structure in English using data from the HPSG Treebank (Miyao et al., 2004), and in parsing of Danish using a rich dependency representation (Kromann, 2003).

Acknowledgements

We thank Yusuke Miyao and Takuya Matsuzaki for insightful discussions. This work was partially supported by Grant-in-Aid for Specially Promoted Research (MEXT, Japan).

References

Briscoe, T., Carroll, J. and Watson, R. 2006. The second release of the RASP system. In *Proceedings of the COLING/ACL-06 Demo Session.*

Buchholz, Sabine and Erwin Marsi. 2006. CoNLL-X Shared Task on Multilingual Dependency Parsing. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL-X) Shared Task session.*

Clark, Stephen, Julia Hockenmaier, and Mark Steedman. 2002. Building Deep Dependency Structures using a Wide-Coverage CCG Parser. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL).*

Covington, Michael A. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the Annual ACM Southeast Conference*, 95-102.

Erkan, Gunes, Arzucan Ozgur, and Dragomir R. Radev. 2007. Semisupervised classification for extracting protein interaction sentences using dependency parsing. In *Proceedings of CoNLL-EMNLP.*

Hudson, Richard. 1984. Word Grammar. Oxford: Blackwell.

Hudson, Richard. 2005. Word Grammar. In K. Brown (Ed.), *Encyclopedia of Language and Linguistics* (second ed., pp 633-642). Elsevier.

Hockenmaier, Julia and Mark Steedman. 2007. CCGbank: a corpus of CCG derivations and dependency structures extracted from the Penn Treebank. In *Computational Linguistics* 33(3), pp 355-396, MIT press.

Kromann, Matthias T. 2003. The Danish dependency treebank and the underlying linguistic theory. In *Proceedings of the Second Workshop on Treebanks and Linguistic Theories (TLT).*

McDonald, Ryan and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL).*

McDonald, Ryan, Fernando Pereira, Kiril Ribarov and Jan Hajic. 2005. Non-projective Dependency Parsing using Spanning Tree Algorithms. In *Proceedings of the Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT-EMNLP).*

Miyao, Yusuke, Takashi Ninomiya, and Jun'ichi Tsujii. 2004. Corpus-oriented grammar development for acquiring a Head-driven Phrase Structure Grammar from the Penn Treebank. In *Proceedings of the International Joint Conference on Natural Language Processing (IJCNLP).*

Miyao Yusuke and Jun'ichi Tsujii. 2005. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics.*

Nivre, Joakim. 2004. Incrementality in Deterministic Dependency Parsing. In *Incremental Parsing: Bringing Engineering and Cognition Together (Workshop at ACL-2004).*

Nivre, Joakim, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, Deniz Yuret. 2007. In *Proceedings of the CoNLL 2007 Shared Task in the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning.*

Nivre, Joakim. and Jens Nilsson. 2005. Pseudo-Projective Dependency Parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 99-106.

Nivre, Joakim. 2007. Incremental non-projective dependency parsing. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT'07).*

Saetre, R., Sagae, K., and Tsujii, J. 2007. Syntactic features for protein-protein interaction extraction. In *Proceedings of the International Symposium on Languages in Biology and Medicine (LBM short oral presentations).*

Sagae, Kenji., Yusuke Miyao Jun'ichi and Tsujii. 2007. HPSG Parsing with shallow dependency constraints. In *Proceedings of the 44th Meeting of the Association for Computational Linguistics.*

Sagae, K., Tsujii, J. 2007. Dependency parsing and domain adaptation with LR models and parser ensembles. In *Proceedings of the CoNLL 2007 Shared Task. in EMNLP-CoNLL.*

Tsuruoka, Yoshimasa and Tsujii, Jun'ichi. 2005. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *Proceedings of the Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT-EMNLP)*, pp. 523-530.

Wang, Mengqiu, Noah A. Smith, and Teruko Mitamura. 2007. What is the Jeopardy Model? A Quasi-Synchronous Grammar for QA. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL).*