Transition-based Dependency DAG Parsing Using Dynamic Oracles

Alper Tokgöz
Istanbul Technical University
Department of Computer Engineering
Istanbul, Turkey
tokgoza@itu.edu.tr

Gülşen Eryiğit
Istanbul Technical University
Department of Computer Engineering
Istanbul, Turkey
gulsen.cebiroglu@itu.edu.tr

Abstract

In most of the dependency parsing studies, dependency relations within a sentence are often presented as a tree structure. Whilst the tree structure is sufficient to represent the surface relations, deep dependencies which may result to multi-headed relations require more general dependency structures, namely Directed Acyclic Graphs (DAGs). This study proposes a new dependency DAG parsing approach which uses a dynamic oracle within a shift-reduce transition-based parsing framework. Although there is still room for improvement on performance with more feature engineering, we already obtain competitive performances compared to static oracles as a result of our initial experiments conducted on the ITU-METU-Sabancı Turkish Treebank (IMST).

1 Introduction

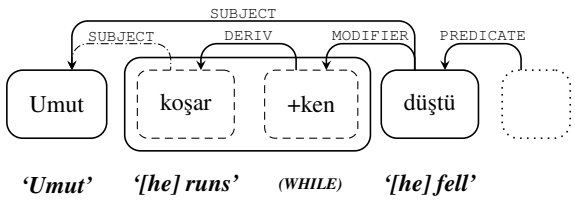
Syntactic parsing is the process of determining the grammatical structure of a sentence as conforming to the grammatical rules of the relevant natural language. The structure of the sentence is determined according to the grammar formalism that the parser is built upon. Phrase structure parsers, also known as constituency parsers, parse a sentence by splitting it into its smaller constituents. On the other hand, in dependency parsers, the structure of the sentence is represented as dependency trees consisting of directed dependency links between a dependent and a head word.

Data-driven dependency parsing frameworks have gained increasing popularity in recent years and been used in a wide range of applications such as machine translation (Ding and Palmer, 2005), textual entailment (Yuret et al., 2013) and question answering (Xu et al., 2014). Most data-driven dependency parsers achieve state-of-the art parsing performances with a language agnostic approach on the different syntactic structures of different languages (Buchholz and Marsi, 2006). Modern data-driven dependency parsers can be categorized into two groups: graph-based and transition-based parsers. Graph-based parsers rely on the global optimization of models aiming to find spanning trees over dependency graphs. On the other hand, transition-based parsers work basically with greedy local decisions that are deterministically selected by oracles, which are generic machine learning models trained to make decisions about the next transition action. In a recent study, Zhang and Nivre (2012) propose a new approach on transition-based parsing that aims to provide global learning instead of greedy local decisions.

Despite the high performances of both graph-based and transition-based dependency parsers, these are generally bounded by the constraint that each dependent may not have multiple heads. Therefore, the resulting parsing output is a tree where words correspond to nodes and dependency relations correspond to edges. Although dependency trees yield satisfactory performances, they are inadequate in capturing dependencies at different levels of semantic interpretations or more complicated linguistic phenomena (e.g. relative clauses, anaphoric references) which could result in multi-head dependencies together with existing surface dependency relations. An example is given in Figure 1 which is taken from the Turkish IMST Treebank (Sulubacak and Eryiğit, 2015). In Figure 1, the dependent token "Umut" depends

Proceedings of the ACL-IJCNLP 2015 Student Research Workshop, pages 22–27,
Beijing, China, July 28, 2015.

on more than one head token with SUBJECT relations: 1) the verb "*koşmak*" (to run) and 2) the verb "*düşmek*" (to fall). Adding a second relation (emphasized with a dash-dotted line in the figure) to the token "Umut" breaks the condition that each token may have at most one head, and renders existing dependency tree parsers incompatible for this setup. It is also worth mentioning that the deep dependencies in the IMST are not discriminated from surface dependencies by the use of different labels.



"Umut fell as [he was] running."

Figure 1: Example for Turkish multi-head dependencies.

In this paper, for the first time in the literature, we investigate the impact of using dynamic oracles for parsing multi-head dependency structures by extending the approach of Goldberg and Nivre (2012). We provide comparisons with the replication of the basic shift-reduce DAG parsing algorithm of Sagae and Tsujii (2008) and a first time implementation of their proposed arc-eager parsing algorithm. The remainder of the paper first gives a background information about the topic, then introduces the DAG parsing framework and the proposed algorithms together with experiments and results.

2 Background

Although it is possible to discover the syntactic relations with a two stage model by first finding the regular surface dependencies and then finding the deep relations with post-processing as in Nivre et al. (2010), it is not always straightforward to decide which dependencies should be treated as surface relations or deep relations as in the case of Turkish. Thus, in this study, we focus on single stage models and aim to discover the entire set of relations in a single pass. McDonald and Pereira (2006) use graph-based algorithms for DAG parsing simply using approximate interference in an edge-factored dependency model starting from dependency trees. On the other hand, Sagae and Tsujii (2008) propose a transition-based counterpart for DAG parsing which made available for parsing multi-headed relations. They modified the existing shift-reduce bottom-up dependency parsing algorithm of Nivre and Nilsson (2005) to allow multiple heads per token by the use of cycle removal and pseudo-projectivization as a preprocessing stage. They report higher performance scores on the Danish treebank compared to McDonald and Pereira (2006).

A standard way of determining transition actions in a shift-reduce dependency parser is using static oracles. During the training stage, the learning instances for the oracle are prepared by the use of manually annotated gold-standard parse trees and the current parsing configuration. During the parsing stage, the already trained oracle decides on the next transition operation. One of the problems with static oracles lays beneath the spurious ambiguity, which implies there might be more than one transition sequence for a given sentence and the sequence proposed by an oracle may not be the easiest to learn. The second problem occurs when the parser makes a parsing error which leads to a parser configuration from which the correct parse tree is not derivable. The algorithm does not provide any solutions for dealing with the error propagation caused by such situations. The idea of dynamic oracles introduced by Goldberg and Nivre (2012) rises for handling the aforementioned handicaps of static oracles. Rather than returning a unique transition for a given configuration, a dynamic oracle returns a set of valid transitions regarding the current configuration, which would allow the algorithm to explore non-optimal configurations during the training procedure.

3 Transition-Based Solutions for Dependency DAG Parsing

Transition-based parsing frameworks consider the transition system to be an abstract machine that processes input sentences and produces corresponding parsing graphs. The tokens of the input sequence and the partially created dependency structures are kept within the following data structures:

1. a buffer β which includes the remaining unprocessed tokens in the input sequence in a queue,

2. a stack σ which consists of the tokens being processed,

3. a set A of assigned dependency arcs.

The transition actions explained in the following subsections are basic stack and queue operations that correspond to parsing actions marking dependency relations. The algorithm starts with a buffer β initialized with all tokens of a sentence preserving their sequence, and an empty stack σ. The parsing process finishes when there are no nodes left in β and only the artificial root in σ.

3.1 Basic shift-reduce parsing with multiple heads

The first algorithm that is capable of parsing DAG structures is the standard algorithm of Sagae and Tsujii (2008). The complete list of the transitions of this algorithm is as follows:

- Shift: Pops the first item of the buffer and pushes it onto the top of the stack.
- Left-Reduce: Pops the top two items of the stack, creates a left arc between them where the top item is assigned as the head of the item below, and pushes the head token back onto the stack.
- Right-Reduce: Pops the top two items of the stack, creates a right arc between them, where the item below is assigned as the head of the top item, and pushes the head token back onto the stack.
- Left-Attach: Creates a left arc between the top two items of the stack, where the top item is assigned as the head of the one below. The stack and the buffer remain unchanged.
- Right-Attach: Creates a right dependency arc between the two top items on the stack and assigns the top token as the dependent of the token below. As the second step, it pops the top of the stack and places it into the buffer β.

3.2 Multi-Head Arc-Eager Parsing Algorithm

The second transition algorithm introduced but not implemented by Sagae and Tsujii (2008) is a variation of the Arc-Eager algorithm of Nivre et al. (2007) and has the following transition operations:

- Shift: Pops the first item of the buffer and pushes it onto the top token of the stack.
- Left-Arc: Creates a left dependency arc between the top token of the stack and the first token of the input buffer, where the first token in the buffer becomes the head and the one at the top of the stack becomes the dependent. It is also worth noticing that the stack and the input buffer remains unchanged.
- Right-Arc: Creates a right dependency arc between the top token of the stack and the first token on the input buffer, where the token in the stack becomes the head, and the token which is in front of the buffer becomes the dependent. It is also worth noticing that the stack and the input buffer remains unchanged.
- Reduce: Pops the top item of the stack if and only if it was previously associated with at least one head.

3.3 Multi-Head Arc Eager Parsing with a Dynamic Oracle

In order to design a dynamic oracle with the capability of parsing multi-head dependencies, we need an efficient method for computing the cost of each transition. To this end, we extend the dynamic oracle defined by Goldberg and Nivre (2012), considering DAG parsing arc-eager system of Sagae and Tsujii (2008). Extended arc-eager transition system will operate in the same way as previously defined in Section 3.2, within a dynamic oracle system whose cost function is defined with a transition operation, the current configuration $c = (\sigma|s, b|\beta, A)$[1] and the gold parse of the sentence (G_{gold}). Differing from Goldberg and Nivre (2012), for ease of computation, we prefer marking transitions as zero cost or costly instead of computing the exact cost:

- $Cost(LeftAttach, c, G_{gold})$ Attaching s to b with a left arc is costly, if there is a right arc between s and b, or it is already attached with a left arc.
- $Cost(RightAttach, c, G_{gold})$ Attaching s to b by creating right arc is costly, if there is a left arc between s and b, or it is already attached with a right arc.

[1] In $c = (\sigma|s, b|\beta, A)$, s denotes the top token of the stack σ, b denotes first item of buffer β, A denotes revealed arcs

- $Cost(Reduce, c, G_{gold})$ Popping s from the stack means it will be no longer possible to associate it with any head or dependent from buffer β, therefore it is costly if it has heads or dependents in the β.

- $Cost(Shift, c, G_{gold})$ Pushing b onto the stack means it will be no longer possible to associate it with any heads or dependents in stack σ, therefore it is costly if it has a head or dependent token in the σ.

Since left attach and right attach operations do not change the parser configuration (i.e. these operations cannot lead to a parser configuration from which the gold tree is not reachable), their cost is measured according to the validity of the attachment. The only difference of our multi-head variant from the single head arc-eager transition system is that the left and right arc operations do not change the parser state. As such, it is essentially a relaxed version of the single-head system. Therefore, since the arc-decomposition property holds for the single-head system (as proven by Goldberg and Nivre (2013)), it also holds for our variant.

We use the same online training procedure (with the perceptron algorithm) as Goldberg and Nivre (2012) given in Algorithm 1.

Algorithm 1 Online training with dynamic oracle

```
1:  procedure TRAIN
2:      w ← 0
3:      for I ← 1, ITERATIONS do
4:          c ← c_s(x)
5:          for sentence x do
6:              while c is not terminal do
7:                  t_p ← argmax_t w.Φ(c, t)
8:                  ZC ← {t|o(t; c; G_gold) = true}
9:                  t_o ← argmax_{t∈ZC} w.Φ(c, t)
10:                 if t_p ∉ ZC then
11:                     w ← w + Φ(c, t_o) − Φ(c, t_p)
12:                 t_n ←NEXT(I, t_p, ZC)
13:                 c ← t_n(c)
14: procedure NEXT(I, t, ZC)
15:     if t ∈ ZC then
16:         return t
17:     else
18:         RANDOM_ELEMENT(ZC)
```

The proposed oracle will return a set of zero cost transition operations (denoted as ZC at line 8) where the costs are calculated according to the cost function defined above. Feature weights will be updated only if the perceptron model makes a transition prediction that does not belong to the zero cost transitions (lines 10 and 11). After that, the next transition operation is chosen by the function NEXT, which returns the transition that is predicted by the model if it belongs to zero cost transitions; if not, it returns a random transition which belongs to the zero cost transition set.

4 Experiments

In order to apply the specified DAG parsing algorithm to non-projective sentences, a pseudo-projective transformation operation is applied to the IMST. For that aim, we apply Head scheme[2] described by Nivre (2005). Moreover, before the application of this pseudo-projective transformation, the cyclic dependency paths are handled as described by Sagae and Tsujii (2008), by reversing the shortest arc within the cyclic dependency path until no cyclic path remains. 99.3% precision and 99.2% recall are acquired on IMST by applying the pseudo-projective transformation and detransformation operations. As a learning component, we follow the work of Sagae and Tsujii (2008) and use a Maximum Entropy model for the classification with the greedy search algorithm. For the dynamic oracle experiment, we use an averaged perceptron algorithm iterating 15 times over the training data.

The following features are used in all of the experiments:

- The POS tag, lemma and morphological features of the top 3 tokens on the stack and the first 3 tokens in the buffer.

- The POS tag and dependency relations of the rightmost and leftmost modifiers of the top 2 items on the stack.

- The number of heads and dependents of the top item of the stack and the first item of the buffer.

- The dependency relations of the top of the stack.

[2] Although other schemes could be tried for DAGs for better performance, this is left for future work due to time constraints.

- Whether the top 2 tokens on the stack have a dependency relation between them or not.
- Whether the top token of the stack and the first of the buffer have a dependency relation between them or not, and if so the direction and the type of the relation.
- Combination of the surface form of the top token of the stack and its number of left and right modifiers.
- Combination of the surface form of the first token of the buffer and its number of left and right modifiers.
- The surface forms and POS tags of heads of the top token of the stack and the first token of the buffer.
- The previous two parsing actions.

For training and evaluation purposes, we use the IMST with ten-fold cross validation. Experiment results are given in Table 4.

Table 1: Unlabeled scores of experiments with using IMST.

Experiment	Precision	Recall	F1
Static-Standard	79.42	77.56	78.50
Static-Eager	78.90	76.79	77.83
Dynamic-Eager	79.68	81.17	80.42

As shown in Table 4, the static arc-eager DAG implementation for Turkish performs slightly worse than the arc-standard algorithm. This is not surprising in the light of previous studies (Nivre, 2008; Eryiğit et al., 2008) reporting that the arc standard algorithm performs better in tree parsing due to the smaller number of classes to be learned by the oracle. However, the proposed multi-head arc-eager algorithm with dynamic oracle (referred to as Dynamic-Eager) yields the best precision, recall and F1 scores among the three experiments.[3]

In this study, although there is still room for improvement on performance with more feature engineering, we obtain better results on Turkish IMST treebank between static and dynamic oracles with our newly proposed method for parsing DAGs. This encourages us to test our system with different languages as future work with the expectation that the ameliorations will be much higher than the reported ones in the single-head scenario.

[3]The difference of this model from the runner-up models are found to be statistically significant according to McNemar's test ($p < 0.0001$)

5 Conclusion and Future Works

In this paper, we experimented with three different transition-based algorithms for DAG parsing which eliminate the single-head constraint of traditional algorithms and allows multi-head dependency relations to represent more complicated linguistic phenomena along with surface relations. We present the first results for arc-eager DAG parsing with static oracles and propose a new arc-eager DAG parsing algorithm using dynamic oracles. Our initial experiments conducted on Turkish pave the way for future research on the usage of the dynamic arc-eager DAG parsing for other languages. For future work, we will first conduct experiments on how well the Dynamic-Eager algorithm performs on different treebanks, including multi-head dependencies (such as the Danish treebank (Kromann, 2003)). Secondly, we will conduct experiments on previously described static-oracle parsing algorithms by using different classifiers such as Support Vector Machines.

Acknowledgments

We hereby acknowledge that this study is part of a research project named "Parsing Web 2.0 Sentences" that is supported by TÜBİTAK (Turkish Scientific and Technological Research Council) 1001 program (grant number 112E276) and part of the ICT COST Action IC 1207.

References

Sabine Buchholz and Erwin Marsi. 2006. Conll-x shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 149–164. Association for Computational Linguistics.

Yuan Ding and Martha Palmer. 2005. Machine translation using probabilistic synchronous dependency insertion grammars. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 541–548. Association for Computational Linguistics.

Gülşen Eryiğit, Joakim Nivre, and Kemal Oflazer. 2008. Dependency parsing of Turkish. *Computational Linguistics*, 34(3):357–389.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *COLING*, pages 959–976.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *Transactions of the association for Computational Linguistics*, 1:403–414.

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

Ryan T McDonald and Fernando CN Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *EACL*. Citeseer.

Joakim Nivre and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 99–106. Association for Computational Linguistics.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(02):95–135.

Joakim Nivre, Laura Rimell, Ryan McDonald, and Carlos Gomez-Rodriguez. 2010. Evaluation of dependency parsers on unbounded dependencies. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 833–841. Association for Computational Linguistics.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Kenji Sagae and Jun'ichi Tsujii. 2008. Shift-reduce dependency DAG parsing. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 753–760. Association for Computational Linguistics.

Umut Sulubacak and Gülşen Eryiğit. 2015. A redefined Turkish dependency grammar and its implementations: A new Turkish web treebank & the revised Turkish treebank. under review.

Kun Xu, Sheng Zhang, Yansong Feng, and Dongyan Zhao. 2014. Answering natural language questions via phrasal semantic parsing. In *Natural Language Processing and Chinese Computing*, pages 333–344. Springer.

Deniz Yuret, Laura Rimell, and Aydın Han. 2013. Parser evaluation using textual entailments. *Language resources and evaluation*, 47(3):639–659.

Yue Zhang and Joakim Nivre. 2012. Analyzing the effect of global learning and beam-search on transition-based dependency parsing. In *COLING (Posters)*, pages 1391–1400.