A new parsing algorithm

Rudolf Rosa

Charles University in Prague, Faculty of Mathematics and Physics, Prague, Czechia.

Abstract. We propose a new dependency parsing algorithm, designed to allow for any features while maintaining tractability. The main idea is to start with a baseline parse tree rather than with no tree at all, and to transform that tree into the correct one by repeated application of simple transformation operations. We focus on inference, discussing anticipated issues and possible remedies. Suggestions towards the training procedure and the feature set are also briefly presented.

1. Introduction

In many of today's state-of-the-art dependency parsers, such as MST parser [McDonald et al., 2005] or Malt parser [Nivre et al., 2006], features must be local with regard to the tree context, being able to take only one edge or a small part of the parse tree into account. Because of that, many phenomena that require a larger context, such as verb valency or coordination structures, cannot easily be captured, let alone explicitly modelling the overall "niceness" of the dependency tree.[1]

An existing way of introducing large-context and global features into parsing is n-best list reranking [Charniak and Johnson, 2005; Collins and Koo, 2006]. In this approach, a parser produces an n-best list of parse trees, and a reranker then uses additional features to choose the best parse tree from these candidates. The reranker has access to the whole tree, and therefore can use any features. There are some issues with reranking though. It may be hard to produce an n-best list from a parser – the rise of complexity often makes this approach impractical. Also, it may simply be better to do everything in one step by designing a parser that allows global features to avoid error cumulation.

A well-known observation is that many languages are primarily left-branching or right-branching;[2] probably for all languages, a left branching or a right branching is generally a very strong baseline parse tree, which is sometimes even hard to beat, especially under limited resources scenarios. It is further well known that typically the most common edge length is 1, i.e. edges are very often part of either the left or the right branching.

In this work, we try to combine the aforementioned observations and approaches in a novel way into a parser that allows employment of any features. The main idea is that instead of starting with an unparsed sentence, we initialize our parser with a baseline parse tree (a left or right branching). The final parse tree is then obtained by performing transformations on the tree. Thus, a whole parse tree is available at each moment, and thus any features, including high-order or global ones, can be used. Moreover, the parser has a baseline parse tree already at the time of initialization, and keeps having a reasonable tree all the time, which we believe to be beneficial.

We present the suggested inference procedure in Section 2, extensively analyzing and discussing it in Section 3. We also briefly discuss possible ways of training the model (Section 4) and envision features to be used (Section 5).

[1]Intuitively, this should be possible, as an experienced researcher can often easily tell that an incorrect dependency tree looks bad, even without inspecting the actual words or their part-of-speech tags – there are some expectations we have about the shape of the tree, its depth, node fertilities, non-projective edges, edge lengths... However, current parsers can be seen as being capable of only a very limited generalization in this respect.

[2]We save ourselves the discussion of to what extent this is a property of the language itself or rather of its treebank; in this work, a language is represented by its treebank to us.

Please note that this paper is a theoretical one. We only state our ideas, the experiments have not been performed yet – this is the obvious and vital future work necessary to validate or reject the ideas drafted in this paper.

2. Inference

We first describe our suggested greedy inference procedure. The goal of the inference procedure is to efficiently find the best parse tree – according to a scoring function, described in Section 2.1. We analyze the suggested inference procedure and suggest possible improvements in Section 3.

1. The inference procedure starts by initializing the *current tree* with a baseline parse tree – a left or right branching, depending on the prevalent characteristic of the language.[3] As usually done, we assume an artificial root node.

2. In each step, we try to transform the current tree by performing one of the following operations:

rotate(*child*) a rotation; rotate the edge between *child* and its parent (the original grandparent of *child* becomes its parent, and the original parent of *child* becomes its child)

attach(*child,parent*) a reattachment; attach *child* as a new child of *parent*

Each operation changes only 1 (attach) or 2 (rotate) edges (i.e. the nodes are moved together with their children). Note that necessarily, not all operations all *valid* – the *parent* node in the *attach* operation cannot be a descendant of the *child* node (as this would lead to a cycle), and a rotation cannot be performed on a child of the artificial root node,

Each valid operation transforms the current tree into a *candidate tree.* If any of the candidate trees is judged to be better than the current tree by the scoring function, the algorithm proceeds to a next step with the highest scoring candidate tree as the new current tree.

3. If no better tree is found, the algorithm stops, returning the current tree as the result.

2.1. Scoring

We propose a scoring function in the general form of $score(tree) = \vec{W} \cdot \vec{F}$, where $\vec{F}$ is the vector of values of the binary features and $\vec{W}$ is a vector of their weights. $\vec{W}$ is to be obtained by training a model using machine learning techniques, see Section 4; $\vec{F}$ is to be defined appropriately to the given language and tuned according to performance, as discussed in Section 5.

We distinguish three types of features:

local computed for each edge, its value depends on the current tree only locally (e.g. number of children of the parent node)

semilocal computed for each edge, its value depends on the current tree globally (e.g. depth of the child node in the tree)

global computed only once for the whole tree (e.g. average node depth)

Thus, the score can be decomposed as follows:

- $score(tree) = score_{local}(tree) + score_{semilocal}(tree) + score_{global}(tree)$
- $score_{local}(tree) = \sum_{j=1}^{N} \sum_{i=1}^{L} w_{l_i} l_i(node_j)$
- $score_{semilocal}(tree) = \sum_{j=1}^{N} \sum_{i=1}^{S} w_{s_i} s_i(node_j)$

[3]The prevalent characteristic of each language can be easily determined from the treebank beforehand.

- $score_{global}(tree) = \sum_{i=1}^{G} w_{g_i} g_i(tree)$

where $l_1, \dots l_L$ are the local features, $s_1, \dots s_S$ are the semilocal features, $g_1, \dots g_G$ are the global features, w are their weights and $node_1, \dots node_N$ are the nodes corresponding to tokens of the sentence (each node uniquely identifies the edge to its parent node).

3. Inference analysis and discussion

In this section, we analyze the algorithm, its strengths and its weaknesses, we discuss several design decisions, and we provide several possible improvements that may be necessary if issues that we forsee as possible are encountered in practice.

3.1. Initialization

The initialization step is simple and efficient, ensuring that the parser starts with an already reasonably good tree. It is conceivable to use an even better baseline parse tree, such as one where the first verb becomes the root, all preceding tokens create a left branching and all following tokens create a right branching. However, the more this is tuned, the more it becomes a rule-based (and also language-specific) parser, which has not been intended and would require a lot of manual work (while the prevalence of left or right branching can be determined automatically from a treebank).

Alternatively, a baseline parser could be used to provide the initial parse tree, thus using our proposed parser rather as a parse post-processor, similarly to the reranking approach (but without the need of the baseline parser to produce n-best parses). The proposed parser is well suited for being initialized with any reasonable parse tree, leading to a proportional reduction of steps needed to achieve the final parse.[4] However, in this case, we expect it would be fruitful to add a feature indicating whether a given edge was part of the baseline parse tree or not.

Of course, the exact way in which the parser will be initialized should be reflected already in the training.

3.2. Operations

In each step, there are at most N^2 possible operations that may be performed on the current tree. While the number of steps is theoretically $O(N^N)$ if the algorithm was to subsequently reach all possible trees, this is very unlikely to happen in practice. In the optimal case, there will be at most N steps (each step will transform the tree into one that has at least one more correct edge), and we believe that if the algorithm is trained properly, the average number of steps will be about N.

It is easy to show that the attach() operation would by itself be sufficient to obtain the correct parse tree in at most N steps (although more steps can be taken in practice, since the parser is allowed to rethink its decisions) – in each step, there is always at least one node that has an incorrect parent and can be attached to its correct parent (i.e. its correct parent is not among its descendants), or the parse tree is correct. However, we propose the "additional" rotate() operation for several reasons:

- it is a one-step transformation of an edge between the two possible branching directions; in many languages, edges corresponding to both of the branching types are quite common, so it seems reasonable to have an easy way of going from one to the other
- it reduces the number of necessary steps (a rotate() operation can be achieved by two attach() operations)

[4]We believe this approach could also be used for transforming sentence parse trees from one annotation style to another.

- it may help avoid the algorithm being stuck in a local optimum (see Section 3.4)
- it has little effect on time complexity, as there are at most $N-1$ possible rotations in each step, while there can be as many as $N \cdot (N-1)$ possible reattachments

As we introduce the rotations as a way of switching between left and right branching, we believe that it may be sufficient to allow only rotations of edges of length 1.[5]

Our original suggestion is to consider all possible $O(N^2)$ reattachments in each step. However, this is not in accord with the observation that most edges are short, and thus short-range reattachments will be much more common than long-range reattachments; moreover, the benefit of starting with a reasonable baseline parse tree would not be fully exploited. Therefore, we propose an alternative approach, which leads to an increased number of steps and an increased number of reconsiderations (i.e. nodes reattached multiple times), but the number of operations considered in each step is much lower.

The general idea behind the suggested approach is to try out the a priori most likely operations first, and only continue with the less likely ones once no further improvement can be gained. Thus, run the algorithm as defined, but allowing only rotations and reattachments of length 1. When a local optimum is reached, increase the maximum allowed reattachment length to 2.[6] Keep running the algorithm and incrementing the maximum reattachment length until it reaches N, then stop.

We expect that most of the steps will happen in the earlier phases and the later slower phases will only take a few steps, thus leading to a lower number of total transformations considered.

3.3. Scoring

Typical parsers only allow local features. We allow two other feature types (see Section 2.1), which is a strength of our parser, but it has implications on complexity: While computing $score_{global}$ is $O(G)$ and computing $score_{local}$ can be easily reduced from $O(N \cdot L)$ to $O(L)$ by dynamic programming, computing $score_{semilocal}$ is $O(N \cdot S)$ and cannot be easily reduced. Thus, computing the score of each candidate tree is $O(N \cdot S + L + G)$. This may be rather costly, and it is therefore preferable not to use a large number of semilocal features.

Still, the need to compute the score for possibly very many candidate trees in each step is expected to make the parser significantly slower than other existing parsers, and it may be of interest to try to reduce the number of steps and/or the number of candidate trees considered in those steps, as discussed in the previous section. Further smaller optimizations may be performed as well, such as disregarding candidate trees corresponding to a reattachment of a node that has just been reattached.

3.4. Search and stopping

Our original proposition is to perform a simple hill climbing, stopping whenever a local optimum is reached. While it may be that with a good training procedure, such an approach will prove to be sufficient, we anticipate possible issues and suggest remedies.

One of the problems of simple hill climbing is the danger of taking an incorrect step. However, we believe that this will not be a grave issue with our approach, as our algorithm is able to reconsider its decisions.[7] Still, if this were to become an issue, beam search can be used.

[5]However, this would not be true if the baseline parse tree were not a branching.

[6]Note that we need to always allow all shorter-range reattachments in each step, as the algorithm may reconsider some of its decisions after long-range reattachments are performed.

[7]Only going back to a tree that had already been the current tree is not even theoretically possible; however, this would be a score error, not a search error.

A more serious threat is the danger of being caught in a local optimum different from the global optimum. This is easily imaginable, as the locally optimal parse tree might look rather good by itself, but to reach the correct one, the algorithm might have to perform a set of operations that will make the tree worse at first, only to make it even better after a few steps. If this turns out to be an issue, we suggest trying to perform a deeper search, e.g. trying out a sequence of two operations instead of one operation to reach the next tree instead of stopping once a local optimum is reached. This change is not expected to slow down the algorithm significantly, as we expect the inference to land in a local optimum only very few times.

4. Training

We suggest to treat the task as a classification problem. In this approach, we closely follow the inference in learning the scoring function. Each step is a separate classification task, in which we try to classify each candidate tree into one of two classes: better or not better than the current tree. As the inference procedure actually considers only the highest scoring candidate tree, it may be sufficient to also take only the currently highest scoring tree into account in the training phase. For space reasons, we do not discuss that here in more detail, nor do we present reasons for which we do not treat the task as a regression problem (predicting tree UAS) instead.

As for the actual learning algorithm, we suggest using e.g. the Margin Infused Relaxed Algorithm (MIRA) [Crammer and Singer, 2003], which has been successfully employed in NLP several times, including the MST parser [McDonald et al., 2005].

5. Feature set

A natural starting point for designing the feature set is taking the edge-local features from standard parsers, such as MST parser [McDonald et al., 2005], looking at edge direction and length, word forms, lemmas and part-of-speech tags of the child and parent node, as well as neighbouring words. While the MST parser looks at sentence-wise neighbouring words, our parser may look at tree-wise neighbouring words as well (or instead). It is easy to integrate any higher order features, both ones already used in parsers [Carreras, 2007], as well as ones previously impossible to employ – sibling features, grandparent/grandchild features, cousin features, uncle features, etc. Of course, care has to be taken not to run into serious data sparseness issues, especially with lexicalized features.[8]

A further place to look for features are rerankers [Charniak and Johnson, 2005; Collins and Koo, 2006] and unsupervised parsers [Mareček and Straka, 2013; Spitkovsky et al., 2013]. These already include some features that try to capture the "niceness" of a tree, but we believe that more can be used, both local and global – e.g. node fertility (i.e. number of children), node depth, subtree depth, edge non-projectivity… Many of these could also be factored according to lemma or part-of-speech – e.g. the number of noun children of a given verb could be a useful feature for modelling verb valency.

For many features, it is also possible to design cumulative features computed on the whole tree, such as total number of non-projective edges, left/right edge ratio, average node fertility, maybe even node fertility variance, etc.

As usual, we propose to use bucketing to turn numerical features into categorial, binarization to turn categorial features into binary, and feature conjunctions to capture interesting phenomena.

We would like to note here our belief that in a harmonized setting, i.e. one where treebanks for multiple languages follow the same annotation style, as in Zeman et al. [2012], many of the

[8]An explicit way of modelling out-of-vocabulary items may be necessary, e.g. by replacing infrequent word forms and lemmas by their suffix only (so e.g. "platypus" might become "-us").

features (e.g. node fertilities) are likely to be largely language-independent; we plan to further investigate this in future.

6. Conclusion and future work

We suggest a new algorithm for syntactic dependency parsing. The algorithm differs from existing parsing algorithms by allowing any features, including global features. This leads to an increased time complexity; however, we discuss several possible ways of treating this issue, and we believe that eventually, inference might be fast enough for the parser to be practical.

The main focus of this paper is the novel inference algorithm, including its possible variations that we suggest be tried out. We also present suggestions towards the training procedure and the feature set.

This work is only theoretical so far. A vital next step is to evaluate the usefulness of our ideas in practice by implementing the parser and evaluating its performance.

If our approach proves to be useful, we further plan to extend it to perform labelled parsing (joint parsing and labelling) in a similar way – i.e. starting with a baseline labelling (all nouns are objects, all adjectives are attributes…) and iteratively relabelling the nodes to increase the overall score of the labelled tree.

We also intend to follow the aforementioned possible research path of using our approach for treebank harmonization. For this task, some changes to our method will be necessary. Most importantly, we will need to account for a cross-lingual setting, where the parser (now in the role of a "transformer") will have to be trained on one or multiple already harmonized treebanks, and then applied to a treebank of a different language; we believe that a delexicalized approach and/or employment of word-based machine translation techniques may be used to achieve this goal. Also, semi-lexicalized features may be useful, such as word length (probably normalized across languages), word affix identity (for a pair of nodes), word/lemma frequency (in the treebank or another corpus), etc.

Acknowledgments. This research was supported by the grants GAUK 1572314 and SVV 260 104.

References

Carreras, X., Experiments with a higher-order projective dependency parser, in *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL*, 2007.

Charniak, E. and Johnson, M., Coarse-to-fine n-best parsing and maxent discriminative reranking, in *Proceedings of ACL*, 2005.

Collins, M. and Koo, T., Discriminative reranking for natural language parsing, *Comp. Ling.*, 2006.

Crammer, K. and Singer, Y., Ultraconservative online algorithms for multiclass problems, *The Journal of Machine Learning Research*, 2003.

Mareček, D. and Straka, M., Stop-probability estimates computed on a large corpus improve unsupervised dependency parsing, in *Proceedings of ACL*, 2013.

McDonald, R., Crammer, K., and Pereira, F., Online large-margin training of dependency parsers, in *Proceedings of ACL*, 2005.

Nivre, J., Hall, J., and Nilsson, J., Maltparser: A data-driven parser-generator for dependency parsing, in *Proceedings of LREC*, 2006.

Spitkovsky, V. I., Alshawi, H., and Jurafsky, D., Breaking out of local optima with count transforms and model recombination: A study in grammar induction, in *Proceedings of EMNLP*, 2013.

Zeman, D., Mareček, D., Popel, M., Ramasamy, L., Štěpánek, J., Žabokrtský, Z., and Hajič, J., HamleDT: To parse or not to parse?, in *Proceedings of LREC*, 2012.