An Improved Non-monotonic Transition System for Dependency Parsing

Matthew Honnibal
spaCy.io
Berlin, Germany
matt@spacy.io

Mark Johnson
Department of Computing
Macquarie University
Sydney, Australia
mark.johnson@mq.edu.edu.au

Abstract

Transition-based dependency parsers usually use transition systems that monotonically extend partial parse states until they identify a complete parse tree. Honnibal et al. (2013) showed that greedy one-best parsing accuracy can be improved by adding additional non-monotonic transitions that permit the parser to "repair" earlier parsing mistakes by "over-writing" earlier parsing decisions. This increases the size of the set of complete parse trees that each partial parse state can derive, enabling such a parser to escape the "garden paths" that can trap monotonic greedy transition-based dependency parsers.

We describe a new set of non-monotonic transitions that permits a partial parse state to derive a larger set of completed parse trees than previous work, which allows our parser to escape from a larger set of garden paths. A parser with our new non-monotonic transition system has 91.85% directed attachment accuracy, an improvement of 0.6% over a comparable parser using the standard monotonic arc-eager transitions.

1 Introduction

Recent work from Dyer et al. (2015) and Weiss et al. (2015) show that neural network models can improve greedy transition-based parsers dramatically, even beyond the 20% error reduction reported by Chen and Manning (2014). Improvements on beam-search parsing are much more limited, due to the difficulty of applying neural networks to structured prediction.

We suggest that the lack of a ready search solution may present the next barrier to further improvements in accuracy. Some degree of search flexibility seems inherently necessary, no matter how powerful the local model becomes, as even the human sentence processor can be 'garden pathed' by local structural ambiguities.

We take inspiration from Frazier and Rayner (1982) and other psycholinguists and propose repair actions as a light-weight alternative to beam-search. In a transition-based dependency parser, transitions map parse states to parse states, ultimately producing completed parse trees. This process is non-deterministic, since usually more than one transition can apply to a parse state. This means that each partial parse state can be associated with a set of complete parse trees (i.e., the complete parses that can be produced by applying sequences of transitions to the partial parse state). In general adding additional transitions (monotonic or non-monotonic) increases the number of complete parse trees that any given partial parse state can derive.

We explore adding non-monotonic parsing transitions to a greedy arc-eager dependency parser in this paper, in order to permit the parser to recover from attachment errors made early in the parsing process. These additional non-monotonic transitions permit the parser to modify what would have been irrevocable parsing decisions in the monotonic arc-eager system when later information justifies this action. Thus one effect of adding the non-monotonic parsing transitions is to effectively delay the location in the input where the parser must ultimately commit to a particular attachment.

Our transition-system builds on the work of Honnibal et al. (2013) and Nivre and Fernandez-Gonzalez (2014), who each present modifications

Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 1385–1390,
Lisbon, Portugal, 17-21 September 2015.

to the arc-eager transition system that introduce some non-monotonic behaviour, resulting in small improvements in accuracy. However, these systems only apply non-monotonic transitions to a relatively small number of configurations, so they can only have a small impact on parse accuracy.

We introduce a non-monotonic transition system that combines ideas from these two approaches, and allows substantially more repair capability (and hence search flexibility). We observe a 0.6% improvement in accuracy on the OntoNotes corpus, which is an error reduction of 6.25% over a competitive baseline. A parser using our transition system is guaranteed to run in linear time, and the modifications to the algorithm have no negative impact on run-time in our implementation.

Very recently there has been considerable success in applying neural network models to predict which transition to apply in greedy one-best transition-based parsing. In their preprints, both Dyer et al. (2015) and Weiss et al. (2015) report error reductions of around 20-30% for greedy one-best parsing, and much more modest improvements for transition-based parsers with beam search. Because the neural network approaches improve the local model that predicts which transition to apply next, while this paper suggests changes to the transition system itself, it is reasonable to expect that the improvements reported here are largely orthogonal to those obtained using the neural network techniques. In future work we would like to explore integrating such neural network models of transition prediction with the extended transition system proposed here.

2 Improved non-monotonic transition system

Our transition-system is based on the tree-constrained arc-eager system of Nivre and Fernandez-Gonzalez (2014), which extends the classic arc-eager system (Nivre, 2003) with a new non-monotonic operation that they call "Unshift". They introduce the Unshift action to repair configurations where the buffer is exhausted and the stack contains multiple words that are without incoming arcs (i.e. without governors). The original arc-eager configuration outputs partial parses in this situation.

Nivre and Fernandez-Gonzalez restrict their Unshift action, such that it can only be applied when the buffer is exhausted and the word on top of the stack has no incoming arc. In this configuration, the Unshift action is the only action that can be applied. The use of the new action is therefore entirely deterministic, and they do not need to produce example configurations for the Unshift action during training. They train their model with what Goldberg and Nivre (2012) term a 'static oracle', which can only label configurations that are consistent with the gold-standard parse.

We take the Nivre and Fernandez-Gonzalez (2014) Unshift operation, and import it into the non-monotonic parsing model of Honnibal et al. (2013), which uses a dynamic oracle to determine the gold-standard actions for configurations produced by the parser. This training strategy is critical to the success of a non-monotonic transition system. The model cannot learn to recover from previous errors if the training data cannot contain configurations that result from incorrect actions.

Honnibal et al. (2013) allow the parser to correct prior misclassifications between the Shift and Right-Arc actions. Both of these actions push the first word of the buffer onto the stack, but the Right-Arc action also adds an arc. After the Right-Arc is applied, the top two words of the stack are connected.

In the original arc-eager system, the presence or absence of this arc determines which of the two pop moves, Reduce or Left-Arc, is valid. If the arc is present, then Left-Arc is excluded; if it is absent, the Reduce action is excluded. Honnibal et al. (2013) argue that these deterministic constraints are unmotivated when the parser is trained using a dynamic, instead of static, oracle. Instead of a constraint, they suggest that consistency be achieved by refining the logic of the actions, so that they have a broader applicability. Instead of preventing the Left-Arc from applying when the word on top of the stack has an incoming arc, they update the definition of the Left-Arc so that it first deletes the existing arc if necessary. A corresponding change is made to the Reduce action: if the model predicts Reduce when the word on top of the stack has no incoming arc, the 'missing' arc is inserted. The arc is labelled by noting the best-scoring Right-Arc label on each Shift action, so that the label can be assigned during non-monotonic Reduce.

We show that the Nivre and Fernandez-Gonzalez Unshift operation serves as a far superior non-monotonic Reduce action than the one

Notation			
$(\sigma, \beta, \mathbf{A}, \mathbf{S})$ is a configuration, where $\sigma \vert s$ is a stack of word indices with topmost element s $b \vert \beta$ is a buffer of word indices with first element b $\mathbf{A}$ is a vector of head indices $\mathbf{A}(i) = j$ denotes an arc $w_j \rightarrow w_i$ $\mathbf{S}$ is a bit-vector used to prevent Shift/Unshift cycles			
Initial	$([\,], [1...n], \mathbf{A}(1) = 1)$		
Terminal	$([i], [\,], \mathbf{A})$		
Shift	$(\sigma, b \vert \beta, \mathbf{A}, \mathbf{S}(b) = 0)$	$\Rightarrow$	$(\sigma \vert b, \beta, \mathbf{A}, \mathbf{S}(b) = 1)$
Right-Arc	$(\sigma \vert s, b \vert \beta, \mathbf{A}, \mathbf{S})$	$\Rightarrow$	$(\sigma \vert s \vert b, \beta, \mathbf{A}(b) = s, \mathbf{S})$
Reduce	$(\sigma \vert s, \beta, \mathbf{A}(s) \neq 0, \mathbf{S})$	$\Rightarrow$	$(\sigma, \beta, \mathbf{A}, \mathbf{S})$
Unshift	$(\sigma \vert s, \beta, \mathbf{A}(s) = 0, \mathbf{S})$	$\Rightarrow$	$(\sigma, s \vert \beta, \mathbf{A}, \mathbf{S})$
Left-Arc	$(\sigma \vert s, b \vert \beta, \mathbf{A}, \mathbf{S})$	$\Rightarrow$	$(\sigma, s \vert \beta, \mathbf{A}(s) = b, \mathbf{S}$

Table 1: Our non-monotonic transition system, which integrates the Unshift action of Nivre and Fernandez-Gonzalez (2014) into the model of Honnibal et al. (2013).

Honnibal et al. use in their system, and that the resulting transition system improves parse accuracy by considerably more than either the Honnibal et al or Nivre et al systems do.

2.1 Definition of Transition System

The hybrid transition system is defined in Table 1. Arcs are stored in a vector, $\mathbf{A}$, where the entry $\mathbf{A}(i) = j$ denotes an arc $w_j \rightarrow w_i$. Words are pushed from the buffer β onto the stack σ, using either the Shift or the Right-Arc actions.

If a word was pushed with the Shift action, it will not have an incoming arc. The new Unshift action will then be valid, at any point at which the word is on top of the stack — even after many actions have been performed.

The Unshift action pops the top word of the stack, s, and places it at the start of the buffer. Parsing then proceeds as normal. To prevent cycles, the Shift action checks and sets a bit in the new boolean vector $\mathbf{S}$. The Shift action is invalid if $\mathbf{S}(b) = 1$, for a word b at the front of the buffer. This bit will be set if the word was previously Shifted, and then Unshifted.

At worst, each word can be pushed and popped from the stack twice, so parsing is guaranteed to terminate after a maximum of $4n$ transitions for a sentence of length n.

The terminal condition is reached when the buffer is exhausted and exactly one word remains on the stack. This word will be deemed the root of the sentence. No 'dummy' root token is necessary, removing the need to choose whether the token is placed at the beginning or end of the buffer (Ballesteros and Nivre, 2013).

Note that if the two words each seem like the governor of the sentence, such that the parser deems all incoming arcs to these words unlikely, the transition system is guaranteed to arrive at a configuration where these two words are adjacent to each other. The model can then predict an arc between them, initiated by either word.

2.2 Dynamic Training Oracle

Goldberg and Nivre (2013) describe three questions that need to be answered in order to implement their training algorithm.

Exploration Policy: *When do we follow an incorrect transition, and which one do we follow?*
We always follow the predicted transition, i.e. their two hyper-parameters are set $k = 1$ and $p = 1.0$.

Optimality: *What constitutes an optimal transition in configurations from which the gold tree is not reachable?*
We follow Honnibal et al. (2013) in defining a transition as optimal if it:

1. Renders no additional arcs unreachable using the *monotonic* arc-eager transitions; and
2. Renders no additional arcs unreachable using the *non-monotonic* transitions.

Said another way, we mark a transition as optimal if it leads to an analysis with as few errors as possible, and in cases of ties, uses as few non-monotonic transitions as possible.

For example, given the input string *I saw Jack*, consider a configuration where *saw* is on the stack, *Jack* is at the front of the buffer, and *I* is attached to *saw*. The gold arcs are *saw* → *I* and *saw* → *Jack*. In the monotonic system, the Shift action would make the gold arc *saw* → *Jack* newly unreachable. In our system, this arc is still reachable after Shift, via the Unshift action, but we consider the Shift move non-optimal, so that the non-monotonic actions are reserved as "repair" operations.

Oracle: *Given a definition of optimality, how do we calculate the set of optimal transitions in a given configuration?*

Goldberg and Nivre (2013) show that with the monotonic arc-eager actions, the following arcs are reachable from an arbitrary configuration:

1. Arcs $\{w_i \rightarrow w_j : i \in \sigma,\ j \in \beta\}$ — i.e. all arcs from stack words to buffer words;
2. Arcs $\{w_i \rightarrow w_j : i \in \beta,\ j \in \sigma,\ \mathbf{A}(j) = 0\}$ — i.e. all arcs from buffer words to headless stack words;
3. Arcs $\{w_i \rightarrow w_j : i \in \beta,\ j \in \beta\}$ — i.e. all arcs between words in buffer.

Our non-monotonic actions additionally allow the following arcs to be reached:

4. Arcs $\{w_i \rightarrow w_j : i \in \beta,\ j \in \sigma, \mathbf{A}(j) \neq 0\}$ (LeftArc can now "clobber" existing heads)
5. Arcs $\{w_i \rightarrow w_j$ or $w_j \rightarrow w_i : i, j \in \sigma, i < j, \mathbf{A}(j) = 0\}$ — i.e. if a word i is on the stack, it can reach an arc to or from a word j ahead of it on the stack if that word does not have a head set.

In practice, we therefore only need to add two rules to determine the set of optimal transitions:

1. If σ_0 has a head, and its true head is in the buffer, the Reduce action is now non-optimal.
2. If σ_0 does not have a head, and its true head is in the stack, the LeftArc action is now non-optimal.

The oracle calculation is simple because the system preserves the *arc decomposition* property that Goldberg and Nivre (2013) prove for the arc eager system: if two arcs of a projective tree are individually reachable from a configuration, a projective tree that includes both arcs is also reachable. To see that this property is preserved in our system, consider that an arc $h \rightarrow d$ between two stack words is only unreachable if $h < d$ and $A(d) \neq 0$. But a projective tree with arc $h \rightarrow d$ cannot also have an arc $x \rightarrow y$ such that $h < x < d < y$. So there can be no other arc part of the same projective tree as $h \rightarrow d$ that would require d to be assigned to some other head.

3 Training Procedure

We follow Honnibal et al. (2013) in using the dynamic oracle-based search-and-learn training strategy introduced by Goldberg and Nivre (2012). A dynamic oracle is a function that labels configurations with gold-standard actions. Importantly, a dynamic oracle can label arbitrary configurations, while a so-called 'static' oracle can only assign labels to configurations that are part of gold-standard derivations.

We employ the dynamic oracle in an online learning strategy, similar to imitation-based learning, where the examples are configurations produced by following the current model's predictions. The configurations are labelled by the dynamic oracle, which determines which of the available actions excludes the fewest gold-standard arcs.

Often, multiple actions will be labelled as gold-standard for a given configuration. This implies either spurious ambiguity (the same analysis reachable via different derivations) or previous errors, such that the best parse reachable by different actions are equally bad. When this occurs, we base the perceptron update on the highest-scoring gold-standard label.

3.1 Single class for Unshift/Reduce

The Unshift and Reduce actions are applicable to a disjoint set of configurations. If the word on top of the stack already has an incoming arc, the Reduce move is valid; otherwise, the Unshift move is valid. For the purpose of training and prediction, we therefore model these actions as a single class, which we interpret based on the configuration. This allows us to learn the Unshift action more effectively, as it is allowed to share a representation with the Reduce move. In preliminary development, we found that assigning a distinct class to the Unshift action was not effective. We plan to evaluate this option more rigorously in future work.

4 Experiments

We implemented a greedy transition-based parser, and used rich contextual features following Zhang and Nivre (2011). We extended the feature set to include Brown cluster features, using the cluster prefix trick described by Koo and Collins (2010). Brown clusters are a standard way to improve the cross-domain performance of supervised linear models. The use of Brown cluster features accounts for the 0.7% improvement in accuracy observed between our baseline parser and the Goldberg and Nivre (2012) result shown in Table 2. The two models are otherwise the same.

Part-of-speech tags were predicted using a greedy averaged perceptron model that achieved 97.2% accuracy on the evaluation data. Most previous work uses a n-way jack-knifing to train the stacked tagger/parser model. For convenience, we instead train the tagger at the same time as the parser, as both allow online learning. We find this makes no difference to accuracy.

Our parsers are trained and evaluated on the same data used by Tetreault et al. (2015) in their recent 'bake-off' of leading dependency parsing models. Specifically, we use the OntoNotes corpus converted into dependencies using the ClearNLP 3.1 converter, with the train / dev / test split of the CoNLL 2012 shared task.

5 Results

We implemented three previous versions of the arc-eager transition system, in order to evaluate the effect of our proposed transition-system on parser accuracy. The four systems differ only in their transition system — they are otherwise identical. All use identical features, and all are trained with the dynamic oracle.

Orig. Arc Eager (Nivre, 2003): the original arc-eager system, which constrains the Reduce and Left-Arc actions to ensure monotonicity; **Prev. Non-Monotic** (Honnibal et al., 2013): relaxes the monotonicity constraints, allowing Left-Arc to "clobber" existing arcs, and inserting missing arcs on Reduce with a simple heuristic; **Tree Constrained** (Nivre and Fernandez-Gonzalez, 2014): adds an Unshift action to the arc-eager system, that is only employed when the buffer is exhausted; **This work**: merges the Unshift action into our previous non-monotonic transition system.

Transition System	Search	UAS	LAS
Orig. Arc Eager	Greedy	91.25	89.40
Tree Constrained	Greedy	91.40	89.50
Prev. Non-Monotonic	Greedy	91.36	89.52
This work	Greedy	91.85	89.91
Chen and Manning (2014)	Greedy	89.59	87.63
Goldberg and Nivre (2012)	Greedy	90.54	88.75
Choi and Mccallum (2013)	Branch	92.26	90.84
Zhang and Nivre (2011)	Beam_{32}	92.24	90.50
Bohnet (2010)	Graph	92.50	90.70

Table 2: Our non-monotonic transition system improves accuracy by 0.6% unlabelled attachment score, for a final score of 91.85 on the OntoNotes corpus.

Table 2 shows the unlabelled and labelled attachment scores of the parsers on the evaluation data. The two previous non-monotonic systems, Prev. Non-monotonic and Tree Constrained, were slightly more accurate than the Orig. Arc Eager system. Our new transition-system had a much bigger impact, improving UAS by 0.6% and LAS by 0.51%. To put the scores in context, we have also included figures reported in a recent survey of the current state-of-the-art (Tetreault et al., 2015). Our parser out-performs existing greedy parsers, and is much more efficient than non-greedy parsers.

6 Conclusions and Future Work

This paper integrates innovations from Honnibal et al. (2013) and Nivre and Fernandez-Gonzalez (2014) to produce a novel non-monotonic set of transitions for transition-based dependency parsing. Doing this required us to use the dynamic oracle of Goldberg and Nivre (2012) during training in order to produce configurations that exercise the non-monotonic transitions. We show that this combination of innovations results in a parser with 91.85% directed accuracy, which is an improvement of 0.6% directed accuracy over an equivalent arc-standard parser. Interestingly, the Honnibal et al and Nivre et al innovations applied on their own only produce improvements of 0.11% and 0.15% respectively, so it seems that these improvements taken together do interact synergistically.

Because our innovation largely affects the search space of a greedy one-best parser, it is likely to be independent of the recent improvements in parsing accuracy that come from using neural networks to predict the best next parsing transition. In future work we plan to combine such neural network models with a version of our parser that incorporates a much larger set of non-monotonic parsing transitions.

References

Miguel Ballesteros and Joakim Nivre. 2013. Going to the roots of dependency parsing. *Computational Linguistics. 39:1.*

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750. Association for Computational Linguistics, Doha, Qatar.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13–24. Association for Computational Linguistics, Beijing, China.

Lyn Frazier and Keith Rayner. 1982. Making and correcting errors during sentence comprehension: Eye movements in the analysis of structurally ambiguous sentences. *Cognitive Psychology*, 14(2):178–210.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of the 24th International Conference on Computational Linguistics (Coling 2012)*. Association for Computational Linguistics, Mumbai, India.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *TACL*, 1:403–414.

Matthew Honnibal, Yoav Goldberg, and Mark Johnson. 2013. A non-monotonic arc-eager transition system for dependency parsing. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 163–172. Association for Computational Linguistics, Sofia, Bulgaria.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1–11.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.

Joakim Nivre and Daniel Fernandez-Gonzalez. 2014. Arc-eager parsing with the tree constraint. *Computational Linguistics*, 40(2):259–267.

Joel Tetreault, Jin ho Choi, and Amanda Stent. 2015. It depends: Dependency parser comparison using a web-based evaluation tool. In *Proceedings of the 53nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13–24. Association for Computational Linguistics, Beijing, China.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proceedings of the 53nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13–24. Association for Computational Linguistics, Beijing, China.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 188–193. Association for Computational Linguistics, Portland, Oregon, USA.