Arc-Eager Parsing with the Tree Constraint

Joakim Nivre[*]
Uppsala University

Daniel Fernández-González[**]
University of Vigo

The arc-eager system for transition-based dependency parsing is widely used in natural language processing despite the fact that it does not guarantee that the output is a well-formed dependency tree. We propose a simple modification to the original system that enforces the tree constraint without requiring any modification to the parser training procedure. Experiments on multiple languages show that the method on average achieves 72% of the error reduction possible and consistently outperforms the standard heuristic in current use.

1. Introduction

One of the most widely used transition systems for dependency parsing is the arc-eager system first described in Nivre (2003), which has been used as the backbone for greedy deterministic dependency parsers (Nivre, Hall, and Nilsson 2004; Goldberg and Nivre 2012), beam search parsers with structured prediction (Zhang and Clark 2008; Zhang and Nivre 2011), neural network parsers with latent variables (Titov and Henderson 2007), and delexicalized transfer parsers (McDonald, Petrov, and Hall 2011). However, in contrast to most similar transition systems, the arc-eager system does not guarantee that the output is a well-formed dependency tree, which sometimes leads to fragmented parses and lower parsing accuracy. Although various heuristics have been proposed to deal with this problem, there has so far been no clean theoretical solution that also gives good parsing accuracy. In this article, we present a modified version of the original arc-eager system, which is provably correct for the class of projective dependency trees, which maintains the linear time complexity of greedy (or beam search) parsers, and which does not require any modifications to the parser training procedure. Experimental evaluation on the CoNLL-X data sets show that the new system consistently outperforms the standard heuristic in current use, on average achieving 72% of the error reduction possible (compared with 41% for the old heuristic).

* Uppsala University, Department of Linguistics and Philology, Box 635, SE-75126, Uppsala, Sweden. E-mail: joakim.nivre@lingfil.uu.se.
** Universidad de Vigo, Departamento de Informática, Campus As Lagoas, 32004, Ourense, Spain. E-mail: danifg@uvigo.es.

Submission received: 25 June 2013; accepted for publication: 4 November 2013.

doi:10.1162/COLI_a_00185

2. The Problem

The dependency parsing problem is usually defined as the task of mapping a sentence $x = w_1, \ldots, w_n$ to a dependency tree T, which is a directed tree with one node for each input token w_i, plus optionally an artificial root node corresponding to a dummy word w_0, and with arcs representing dependency relations, optionally labeled with dependency types (Kübler, McDonald, and Nivre 2009). In this article, we will furthermore restrict our attention to dependency trees that are projective, meaning that every subtree has a contiguous yield. Figure 1 shows a labeled projective dependency tree.

Transition-based dependency parsing views parsing as heuristic search through a non-deterministic transition system for deriving dependency trees, guided by a statistical model for scoring transitions from one configuration to the next. Figure 2 shows the arc-eager transition system for dependency parsing (Nivre 2003, 2008). A parser configuration consists of a stack σ, a buffer β, and a set of arcs A. The initial configuration for parsing a sentence $x = w_1, \ldots, w_n$ has an empty stack, a buffer containing the words $w_1, \ldots, w_n$, and an empty arc set. A terminal configuration is any configuration with an empty buffer. Whatever arcs have then been accumulated in the arc set A defines the output dependency tree. There are four possible transitions from a configuration

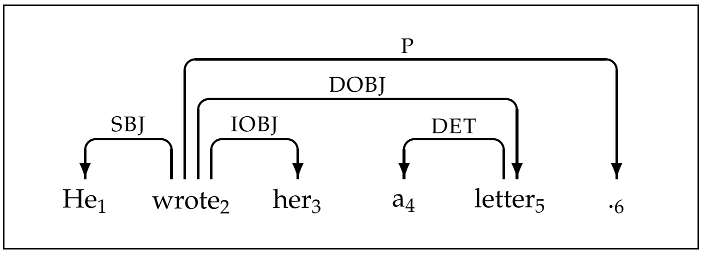

Figure 1
Projective labeled dependency tree for an English sentence.

Initial:	$([\,], [w_1, \ldots, w_n], \{\,\})$			
Terminal:	$(\sigma, [\,], A)$			
Shift:	$(\sigma, w_i \vert \beta, A)$	$\Rightarrow$	$(\sigma \vert w_i, \beta, A)$	
Reduce:	$(\sigma \vert w_i, \beta, A)$	$\Rightarrow$	(σ, β, A)	$\text{HEAD}(w_i)$
Right-Arc:	$(\sigma \vert w_i, w_j \vert \beta, A)$	$\Rightarrow$	$(\sigma \vert w_i \vert w_j, \beta, A \cup \{w_i \rightarrow w_j\})$	
Left-Arc:	$(\sigma \vert w_i, w_j \vert \beta, A)$	$\Rightarrow$	$(\sigma, w_j \vert \beta, A \cup \{w_i \leftarrow w_j\})$	$\neg\text{HEAD}(w_i)$

Figure 2
Arc-eager transition system for dependency parsing. We use | as list constructor, meaning that $\sigma|w_i$ is a stack with top w_i and remainder σ and $w_j|\beta$ is a buffer with head w_j and tail β. The condition $\text{HEAD}(w_i)$ is true in a configuration (σ, β, A) if A contains an arc $w_k \rightarrow w_i$ (for some k).

where *top* is the word on top of the stack (if any) and *next* is the first word of the buffer:[1]

1. **Shift** moves *next* to the stack.
2. **Reduce** pops the stack; allowed only if *top* has a head.
3. **Right-Arc** adds a dependency arc from *top* to *next* and moves *next* to the stack.
4. **Left-Arc** adds a dependency arc from *next* to *top* and pops the stack; allowed only if *top* has no head.

The arc-eager system defines an incremental left-to-right parsing order, where left dependents are added bottom–up and right dependents top–down, which is advantageous for postponing certain attachment decisions. However, a fundamental problem with this system is that it does not guarantee that the output parse is a projective dependency tree, only a projective dependency forest, that is, a sequence of adjacent, non-overlapping projective trees (Nivre 2008). This is different from the closely related arc-standard system (Nivre 2004), which constructs all dependencies bottom–up and can easily be constrained to only output trees. The failure to implement the tree constraint may lead to fragmented parses and lower parsing accuracy, especially with respect to the global structure of the sentence. Moreover, even if the loss in accuracy is not substantial, this may be problematic when using the parser in applications where downstream components may not function correctly if the parser output is not a well-formed tree.

The standard solution to this problem in practical implementations, such as MaltParser (Nivre, Hall, and Nilsson 2006), is to use an artificial root node and to attach all remaining words on the stack to the root node at the end of parsing. This fixes the formal problem, but normally does not improve accuracy because it is usually unlikely that more than one word should attach to the artificial root node. Thus, in the error analysis presented by McDonald and Nivre (2007), MaltParser tends to have very low precision on attachments to the root node. Other heuristic solutions have been tried, usually by post-processing the nodes remaining on the stack in some way, but these techniques often require modifications to the training procedure and/or undermine the linear time complexity of the parsing system. In any case, a clean theoretical solution to this problem has so far been lacking.

3. The Solution

We propose a modified version of the arc-eager system, which guarantees that the arc set A in a terminal configuration forms a projective dependency tree. The new system, shown in Figure 3, differs in four ways from the old system:

1. Configurations are extended with a boolean variable e, keeping track of whether we have seen the end of the input, that is, whether we have passed through a configuration with an empty buffer.

1 For simplicity, we only consider unlabeled parsing here. In labeled parsing, which is used in all experiments, **Right-Arc** and **Left-Arc** also have to select a label for the new arc.

Initial:	$([\,], [w_1, \ldots, w_n], \{\,\}, \mathbf{false})$			
Terminal:	$([w_i], [\,], A, \mathbf{true})$			
Shift:	$(\sigma, w_i\vert\beta, A, \mathbf{false})$	$\Rightarrow$	$(\sigma\vert w_i, \beta, A, [\![\beta = [\,]]\!])$	
Unshift:	$(\sigma\vert w_i, [\,], A, \mathbf{true})$	$\Rightarrow$	$(\sigma, [w_i], A, \mathbf{true})$	$\neg\textsc{head}(w_i)$
Reduce:	$(\sigma\vert w_i, \beta, A, e)$	$\Rightarrow$	(σ, β, A, e)	$\textsc{head}(w_i)$
Right-Arc:	$(\sigma\vert w_i, w_j\vert\beta, A, e)$	$\Rightarrow$		
	$(\sigma\vert w_i\vert w_j, \beta, A \cup \{w_i \rightarrow w_j\}, [\![e \vee \beta = [\,]]\!])$			
Left-Arc:	$(\sigma\vert w_i, w_j\vert\beta, A, e)$	$\Rightarrow$	$(\sigma, w_j\vert\beta, A \cup \{w_i \leftarrow w_j\}, e)$	$\neg\textsc{head}(w_i)$

Figure 3
Arc-eager transition system enforcing the tree constraint. The expression $[\![\phi]\!]$ evaluates to **true** if ϕ is true and **false** otherwise.

2. Terminal configurations have the form $([w_i], [\,], A, \mathbf{true})$, that is, they have an empty buffer, exactly one word on the stack, and $e = \mathbf{true}$.
3. The **Shift** transition is allowed only if $e = \mathbf{false}$.
4. There is a new transition **Unshift**, which moves *top* back to the buffer and which is allowed only if *top* has no head and the buffer is empty.

The new system behaves exactly like the old system until we reach a configuration with an empty buffer, after which there are two alternatives. If the stack contains exactly one word, we terminate and output a tree, which was true also in the old system. However, if the stack contains more than one word, we now go on parsing but are forbidden to make any **Shift** transitions. After this point, there are two cases. If the buffer is empty, we make a deterministic choice between **Reduce** and **Unshift** depending on whether *top* has a head or not. If the buffer is not empty, we non-deterministically choose between **Right-Arc** and either **Left-Arc** or **Reduce** (the latter again depending on whether *top* has a head). Because the new **Unshift** transition is only used in completely deterministic cases, we can use the same statistical model to score transitions both before and after we have reached the end of the input, as long as we make sure to block any **Shift** transition favored by the model.

We first show that the new system is still guaranteed to terminate and that the maximum number of transitions is $O(n)$, where n is the length of the input sentence, which guarantees linear parsing complexity for greedy (and beam search) parsers with constant-time model predictions and transitions. From previous results, we know that the system is guaranteed to reach a configuration of the form $(\sigma, [\,], A)$ in $2n - k$ transitions, where $k = |\sigma|$ (Nivre 2008).[2] In any non-terminal configuration arising from this point on, we can always perform **Reduce** or **Unshift** (in case the buffer is empty) or **Right-Arc** (otherwise), which means that termination is guaranteed if we can show that the number of additional transitions is bounded.

2 This holds because we must move n words from the buffer to the stack (in either a **Shift** or a **Right-Arc** transition) and pop $n - k$ words from the stack (in either a **Reduce** or **Left-Arc** transition).

Note first that we can perform at most $k-1$ **Unshift** transitions moving a word back to the buffer (because a word can only be moved back to the buffer if it has no head, which can only happen once since **Shift** is now forbidden).[3] Therefore, we can perform at most $k-1$ **Right-Arc** transitions, moving a word back to the stack and attaching it to its head. Finally, we can perform at most $k-1$ **Reduce** and **Left-Arc** transitions, removing a word from the stack (regardless of whether it has first been moved back to the buffer). In total, we can thus perform at most $2n-k+3(k-1) < 4n$ transitions, which means that the number of transitions is $O(n)$.

Having shown that the new system terminates after a linear number of transitions, we now show that it also guarantees that the output is a well-formed dependency tree. In order to reach a terminal configuration, we must pop $n-1$ words from the stack, each of which has exactly one incoming arc and is therefore connected to at least one other node in the graph. Because the word remaining in the stack has no incoming arc but must be connected to (at least) the last word that was popped, it follows that the resulting graph is connected with exactly $n-1$ arcs, which entails that it is a tree.

It is worth noting that, although the new system can always construct a tree over the unattached words left on the stack in the first configuration of the form $(\sigma, [\,], A)$, it may not be able to construct every possible tree over these nodes. More precisely, a sequence of words $w_j, \ldots, w_k$ can only attach to a word on the left in the form of a chain (not as siblings) and can only attach to a word on the right as siblings (not as a chain). Nevertheless, the new system is both sound and complete for the class of projective dependency trees, because every terminating transition sequence derives a projective tree (soundness) and every projective tree is derived by some transition sequence (completeness). By contrast, the original arc-eager system is complete but not sound for the class of projective trees.

4. Experiments

In our empirical evaluation we make use of the open-source system MaltParser (Nivre, Hall, and Nilsson 2006), which is a data-driven parser-generator for transition-based dependency parsing supporting the use of different transition systems. Besides the original arc-eager system, which is already implemented in MaltParser, we have added an implementation of the new modified system. The training procedure used in MaltParser derives an oracle transition sequence for each sentence and gold tree in the training corpus and uses every configuration–transition pair in these sequences as a training instance for a multi-class classifier. Because the oracle sequences in the arc-eager system always produce a well-formed tree, there will be no training instances corresponding to the extended transition sequences in the new system (i.e., sequences containing one or more non-terminal configurations of the form $(\sigma, [\,], A)$). However, because the **Unshift** transition is only used in completely deterministic cases, where the classifier is not called upon to rank alternative transitions, we can make use of exactly the same classifier for both the old and the new system.[4]

We compare the original and modified arc-eager systems on all 13 data sets from the CoNLL-X shared task on multilingual dependency parsing (Buchholz and Marsi 2006),

3 The number is $k-1$, rather than k, because **Unshift** requires an empty buffer, which together with only one word on the stack would imply a terminal configuration.

4 Although this greatly simplifies the integration of the new system into existing parsing frameworks, it is conceivable that accuracy could be improved further through specialized training methods, for example, using a dynamic oracle along the lines of Goldberg and Nivre (2012). We leave this for future research.

which all assume the existence of a dummy root word prefixed to the sentence. We tune the feature representations separately for each language and projectivize the training data for languages with non-projective dependencies but otherwise use default settings in MaltParser (including the standard heuristic of attaching any unattached tokens to the artificial root node at the end of parsing for the original system). Because we want to perform a detailed error analysis for fragmented parses, we initially avoid using the dedicated test set for each language and instead report results on a development set created by splitting off 10% of the training data.

Table 1 (columns 2–3) shows the unlabeled attachment score (including punctuation) achieved with the two systems. We see that the new system improves over the old one by 0.19 percentage points on average, with individual improvements ranging from 0.00 (Japanese) to 0.50 (Slovene). These differences may seem quantitatively small, but it must be remembered that the unattached tokens left on the stack in fragmented parses constitute a very small fraction of the total number of tokens on which these scores are calculated. In order to get a more fine-grained picture of the behavior of the two systems, we therefore zoom in specifically on these tokens in the rest of Table 1.

Column 4 shows the number of unattached tokens left on the stack when reaching the end of the input (excluding the artificial root node). Column 5 shows for how many of these tokens the correct head is also on the stack (including the artificial root node). Both statistics are summed over all sentences in the development set. We see from these figures that the amount of fragmentation varies greatly between languages, from only four unattached tokens for Japanese to 230 tokens for Slovene. These tendencies seem to reflect properties of the data sets, with Japanese having the lowest average sentence length of all languages and Slovene having a high percentage of non-projective dependencies and a very small training set. They also partly explain why these languages show the smallest and largest improvement, respectively, in overall attachment score.

Table 1
Experimental results for the old and new arc-eager transition systems (development sets). UAS = unlabeled attachment score; Stack-Token = number of unattached tokens left in the stack when reaching the end of the input (excluding the artificial root node); Stack-Head = number of unattached tokens for which the head is also left in the stack (including the artificial root node); Correct = number of tokens left in the stack that are correctly attached in the final parser output; Recall = Correct/Stack-Head (%).

	UAS		Stack		Correct		Recall	
Language	**Old**	**New**	**Token**	**Head**	**Old**	**New**	**Old**	**New**
Arabic	77.38	77.74	38	24	3	22	12.50	91.67
Bulgarian	90.32	90.42	20	15	7	13	46.67	86.67
Chinese	89.28	89.48	33	31	8	18	25.81	58.06
Czech	83.26	83.46	41	36	8	21	22.22	58.33
Danish	88.10	88.17	29	23	18	22	78.26	95.65
Dutch	86.23	86.49	63	57	13	28	22.80	49.12
German	87.44	87.75	66	39	6	27	15.38	69.23
Japanese	93.53	93.53	4	4	4	4	100.00	100.00
Portuguese	87.68	87.84	44	36	15	26	41.67	72.22
Slovene	76.50	77.00	230	173	50	97	28.90	56.07
Spanish	81.43	81.59	57	42	13	22	30.95	52.38
Swedish	88.18	88.33	43	28	13	24	46.43	85.71
Turkish	81.44	81.45	24	16	9	10	56.25	62.50
Average	85.44	85.63	53.23	40.31	12.85	25.69	40.60	72.12

Table 2
Results on final test sets. LAS = labeled attachment score. UAS = unlabeled attachment score.

		Ara	Bul	Cze	Chi	Dan	Dut	Ger	Jap	Por	Slo	Spa	Swe	Tur	Ave
LAS	Old	65.90	87.39	86.11	77.82	84.11	77.28	85.42	90.88	83.52	70.30	79.72	83.19	73.92	80.43
	New	66.13	87.45	86.27	77.93	84.16	77.37	85.51	90.84	83.76	70.86	79.73	83.19	73.98	80.55
UAS	Old	76.33	90.51	89.79	83.09	88.74	79.95	87.92	92.44	86.28	77.09	82.47	88.70	81.09	84.95
	New	76.49	90.58	89.94	83.09	88.82	80.18	88.00	92.40	86.33	77.34	82.49	88.76	81.20	85.05

Columns 6 and 7 show, for the old and the new system, how many of the unattached tokens on the stack are attached to their correct head in the final parser output, as a result of heuristic root attachment for the old system and extended transition sequences for the new system. Columns 8 and 9 show the same results expressed in terms of recall or error reduction (dividing column 6/7 by column 5). These results clearly demonstrate the superiority of the new system over the old system with heuristic root attachment. Whereas the old system correctly attaches 40.60% of the tokens for which a head can be found on the stack, the new system finds correct attachments in 72.12% of the cases. For some languages, the effect is dramatic, with Arabic improving from just above 10% to over 90% and German from about 15% to almost 70%, but all languages clearly benefit from the new technique for enforcing the tree constraint.

Variation across languages can to a large extent be explained by the proportion of unattached tokens that should be attached to the artificial root node. Because the old root attachment heuristic attaches all tokens to the root, it will have 100% recall on tokens for which this is the correct attachment and 0% recall on all other tokens. This explains why the old system gets 100% recall on Japanese, where all four tokens left on the stack should indeed be attached to the root. It also means that, on average, root attachment is only correct for about 40% of the cases (which is the overall recall achieved by this method). By contrast, the new system only achieves a recall of 82.81% on root attachments, but this is easily compensated by a recall of 63.50% on non-root attachments.

For completeness, we report also the labeled and unlabeled attachment scores (including punctuation) on the dedicated test sets from the CoNLL-X shared task, shown in Table 2. The results are perfectly consistent with those analyzed in depth for the development sets. The average improvement is 0.12 for LAS and 0.10 for UAS. The largest improvement is again found for Slovene (0.58 LAS, 0.25 UAS) and the smallest for Japanese, where there is in fact a marginal drop in accuracy (0.04 LAS/UAS).[5] For all other languages, however, the new system is at least as good as the old system and in addition guarantees a well-formed output without heuristic post-processing. Moreover, although the overall improvement is small, there is a statistically significant improvement in either LAS or UAS for all languages except Bulgarian, Czech, Japanese, Spanish, and Swedish, and in both LAS and UAS on average over all languages according to a randomized permutation test ($\alpha = .05$) (Yeh 2000). Finally, it is worth noting that there is no significant difference in running time between the old and the new system.

5 As we saw in the previous analysis, fragmentation happens very rarely for Japanese and all unattached tokens should normally be attached to the root node, which gives 100% recall for the baseline parser.

5. Conclusion

In conclusion, we have presented a modified version of the arc-eager transition system for dependency parsing, which, unlike the old system, guarantees that the output is a well-formed dependency tree. The system is provably sound and complete for the class of projective dependency trees, and the number of transitions is still linear in the length of the sentence, which is important for efficient parsing. The system can be used without modifying the standard training procedure for greedy transition-based parsers, because the statistical model used to score transitions is the same as for the old system. An empirical evaluation on all 13 languages from the CoNLL-X shared task shows that the new system consistently outperforms the old system with the standard heuristic of attaching all unattached tokens to the artificial root node. Whereas the old method only recovers about 41% of the attachments that are still feasible, the new system achieves an average recall of 72%. Although this gives only a marginal effect on overall attachment score (at most 0.5%), being able to guarantee that output parses are always well formed may be critical for downstream modules that take these as input. Moreover, the proposed method achieves this guarantee as a theoretical property of the transition system without having to rely on ad hoc post-processing and works equally well regardless of whether a dummy root word is used or not.

Acknowledgments
This research has been partially funded by the Spanish Ministry of Economy and Competitiveness and FEDER (project TIN2010-18552-C03-01), Ministry of Education (FPU Grant Program), and Xunta de Galicia (projects CN 2012/319 and CN 2012/317).

References

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)*, pages 149–164, New York, NY.

Goldberg, Yoav and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of the 24th International Conference on Computational Linguistics (COLING)*, pages 959–976, Jeju Island.

Kübler, Sandra, Ryan McDonald, and Joakim Nivre. 2009. *Dependency Parsing*. Morgan and Claypool.

McDonald, Ryan and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 122–131, Prague.

McDonald, Ryan, Slav Petrov, and Keith Hall. 2011. Multi-source transfer of delexicalized dependency parsers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 62–72, Edinburgh.

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

Nivre, Joakim. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together (ACL)*, pages 50–57, Stroudsburg, PA.

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

Nivre, Joakim, Johan Hall, and Jens Nilsson. 2004. Memory-based dependency parsing. In *Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL)*, pages 49–56, Boston, MA.

Nivre, Joakim, Johan Hall, and Jens Nilsson. 2006. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*, pages 2,216–2,219, Genoa.

Titov, Ivan and James Henderson. 2007. A latent variable model for generative dependency parsing. In *Proceedings of the 10th International Conference on Parsing Technologies (IWPT)*, pages 144–155, Prague.

Yeh, Alexander. 2000. More accurate tests for the statistical significance of result differences. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING)*, pages 947–953, Saarbrüken.

Zhang, Yue and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 562–571, Honolulu, HI.

Zhang, Yue and Joakim Nivre. 2011. Transition-based parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 188–193, Portland, OR.