The Best of Both Worlds – A Graph-based Completion Model for Transition-based Parsers

Bernd Bohnet and **Jonas Kuhn**
University of Stuttgart
Institute for Natural Language Processing
{bohnet,jonas}@ims.uni-stuttgart.de

Abstract

Transition-based dependency parsers are often forced to make attachment decisions at a point when only partial information about the relevant graph configuration is available. In this paper, we describe a model that takes into account complete structures as they become available to rescore the elements of a beam, combining the advantages of transition-based and graph-based approaches. We also propose an efficient implementation that allows for the use of sophisticated features and show that the completion model leads to a substantial increase in accuracy. We apply the new transition-based parser on typologically different languages such as English, Chinese, Czech, and German and report competitive labeled and unlabeled attachment scores.

1 Introduction

Background. A considerable amount of recent research has gone into data-driven dependency parsing, and interestingly throughout the continuous process of improvements, two classes of parsing algorithms have stayed at the centre of attention, the transition-based (Nivre, 2003) vs. the graph-based approach (Eisner, 1996; McDonald et al., 2005).[1] The two approaches apply fundamentally different strategies to solve the task of finding the optimal labeled dependency tree over the words of an input sentence (where supervised machine learning is used to estimate the scoring parameters on a treebank).

The transition-based approach is based on the conceptually (and cognitively) compelling idea that machine learning, i.e., a model of linguistic experience, is used in exactly those situations when there is an attachment choice in an otherwise deterministic incremental left-to-right parsing process. As a new word is processed, the parser has to decide on one out of a small number of possible transitions (adding a dependency arc pointing to the left or right and/or pushing or popping a word on/from a stack representation). Obviously, the learning can be based on the feature information available at a particular snapshot in incremental processing, i.e., only surface information for the unparsed material to the right, but full structural information for the parts of the string already processed. For the completely processed parts, there are no principled limitations as regards the types of structural configurations that can be checked in feature functions.

The graph-based approach in contrast emphasizes the objective of exhaustive search over all possible trees spanning the input words. Commonly, dynamic programming techniques are used to decide on the optimal tree for each particular word span, considering all candidate splits into subspans, successively building longer spans in a bottom-up fashion (similar to chart-based constituent parsing). Machine learning drives the process of deciding among alternative candidate splits, i.e., feature information can draw on full structural information for the entire material in the span under consideration. However, due to the dynamic programming approach, the features cannot use arbitrarily complex structural configurations: otherwise the dynamic programming chart would have to be split into exponentially many special states. The typical feature models are based on combinations of edges (so-

[1] More references will be provided in sec. 2.

Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics, pages 77–87,
Avignon, France, April 23 - 27 2012. ©2012 Association for Computational Linguistics

called second-order factors) that closely follow the bottom-up combination of subspans in the parsing algorithm, i.e., the feature functions depend on the presence of two specific dependency edges. Configurations not directly supported by the bottom-up building of larger spans are more cumbersome to integrate into the model (since the combination algorithm has to be adjusted), in particular for third-order factors or higher.

Empirically, i.e., when applied in supervised machine learning experiments based on existing treebanks for various languages, both strategies (and further refinements of them not mentioned here) turn out roughly equal in their capability of picking up most of the relevant patterns well; some subtle strengths and weaknesses are complementary, such that *stacking* of two parsers representing both strategies yields the best results (Nivre and McDonald, 2008): in training and application, one of the parsers is run on each sentence prior to the other, providing additional feature information for the other parser. Another successful technique to combine parsers is voting as carried out by Sagae and Lavie (2006).

The present paper addresses the question if and how a more integrated combination of the strengths of the two strategies can be achieved and implemented efficiently to warrant competitive results.

The main issue and solution strategy. In order to preserve the conceptual (and complexity) advantages of the transition-based strategy, the integrated algorithm we are looking for has to be transition-based at the top level. The advantages of the graph-based approach – a more globally informed basis for the decision among different attachment options – have to be included as part of the scoring procedure. As a prerequisite, our algorithm will require a memory for storing alternative analyses among which to choose. This has been previously introduced in transition-based approaches in the form of a beam (Johansson and Nugues, 2006): rather than representing only the best-scoring history of transitions, the k best-scoring alternative histories are kept around.

As we will indicate in the following, the mere addition of beam search does not help overcome a representational key issue of transition-based parsing: in many situations, a transition-based parser is forced to make an attachment decision for a given input word at a point where no or only partial information about the word's own dependents (and further decendents) is available. Figure 1 illustrates such a case.

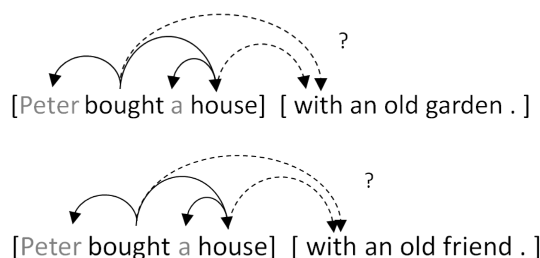

Figure 1: The left set of brackets indicates material that has been processed or is under consideration; on the right is the input, still to be processed. Access to information that is yet unavailable would help the parser to decide on the correct transition.

Here, the parser has to decide whether to create an edge between *house* and *with* or between *bought* and *with* (which is technically achieved by first popping *house* from the stack and then adding the edge). At this time, no information about the object of *with* is available; *with* fails to provide what we call a *complete factor* for the calculation of the scores of the alternative transitions under consideration. In other words, the model cannot make use of any evidence to distinguish between the two examples in Figure 1, and it is bound to get one of the two cases wrong.

Figure 2 illustrates the same case from the perspective of a graph-based parser.

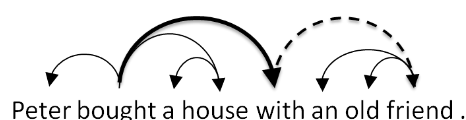

Figure 2: A second order model as used in graph-based parsers has access to the crucial information to build the correct tree. In this case, the parser condsiders the word *friend* (as opposed to *garden*, for instance) as it introduces the bold-face edge.

Here, the combination of subspans is performed at a point when their internal structure has been finalized, i.e., the attachment of *with* (to *bought* or *house*) is not decided until it is clear that *friend* is the object of *with*; hence, the semantically important lexicalization of *with*'s object informs the higher-level attachment decision through a so-called second order factor in the feature model.

Given a suitable amount of training data, the model can thus learn to make the correct decision. The dynamic-programming based graph-based parser is designed in such a way that any score calculation is based on complete factors for the subspans that are combined at this point.

Note that the problem for the transition-based parser cannot be remedied by beam search alone. If we were to keep the two options for attaching *with* around in a beam (say, with a slightly higher score for attachment to *house*, but with *bought* following narrowly behind), there would be no point in the further processing of the sentence at which the choice could be corrected: the transition-based parser still needs to make the decision that *friend* is attached to *with*, but this will not lead the parser to reconsider the decision made earlier on.

The strategy we describe in this paper applies in this very type of situation: whenever information is added in the transition-based parsing process, the scores of all the histories stored in the beam are *recalculated* based on a scoring model inspired by the graph-based parsing approach, i.e., taking complete factors into account as they become incrementally available. As a consequence the beam is reordered, and hence, the incorrect preference of an attachment of *with* to *house* (based on incomplete factors) can later be corrected as *friend* is processed and the complete second-order factor becomes available.[2]

The integrated transition-based parsing strategy has a number of advantages:
(1) We can integrate and investigate a number of third order factors, without the need to implement a more complex parsing model each time anew to explore the properties of such distinct model.
(2) The parser with completion model maintains the favorable complexity of transition-based parsers.
(3) The completion model compensates for the lower accuracy of cases when only incomplete information is available.
(4) The parser combines the two leading parsing paradigms in a single efficient parser **without** stacking the two approaches. Therefore the parser requires only one training phase (without jackknifing) and it uses only a single transition-based decoder.

The structure of this paper is as follows. In Section 2, we discuss related work. In Section 3, we introduce our transition-based parser and in Section 4 the completion model as well as the implementation of third order models. In Section 5, we describe experiments and provide evaluation results on selected data sets.

2 Related Work

Kudo and Matsumoto (2002) and Yamada and Matsumoto (2003) carried over the idea for deterministic parsing by chunks from Abney (1991) to dependency parsing. Nivre (2003) describes in a more strict sense the first incremental parser that tries to find the most appropriate dependency tree by a sequence of local transitions. In order to optimize the results towards a more globally optimal solution, Johansson and Nugues (2006) first applied beam search, which leads to a substantial improvment of the results (cf. also (Titov and Henderson, 2007)). Zhang and Clark (2008) augment the beam-search algorithm, adapting the early update strategy of Collins and Roark (2004) to dependency parsing. In this approach, the parser stops and updates the model when the oracle transition sequence drops out of the beam. In contrast to most other approaches, the training procedure of Zhang and Clark (2008) takes the complete transition sequence into account as it is calculating the update. Zhang and Clark compare aspects of transition-based and graph-based parsing, and end up using a transition-based parser with a combined transition-based/second-order graph-based scoring model (Zhang and Clark, 2008, 567), which is similar to the approach we describe in this paper. However, their approach does not involve beam rescoring as the partial structures built by the transition-based parser are subsequently augmented; hence, there are cases in which our approach is able to differentiate based on higher-order factors that go unnoticed by the combined model of (Zhang and Clark, 2008, 567).

One step beyond the use of a beam is a dynamic programming approach to carry out a full search in the state space, cf. (Huang and Sagae, 2010; Kuhlmann et al., 2011). However, in this case one has to restrict the employed features to a set which fits to the elements composed by the dy-

[2] Since search is not exhaustive, there is of course a slight danger that the correct history drops out of the beam before complete information becomes available. But as our experiments show, this does not seem to be a serious issue empirically.

namic programming approach. This is a trade-off between an exhaustive search and a unrestricted (rich) feature set and the question which provides a higher accuracy is still an open research question, cf. (Kuhlmann et al., 2011).

Parsing of non-projective dependency trees is an important feature for many languages. At first most algorithms were restricted to projective dependency trees and used pseudo-projective parsing (Kahane et al., 1998; Nivre and Nilsson, 2005). Later, additional transitions were introduced to handle non-projectivity (Attardi, 2006; Nivre, 2009). The most common strategy uses the *swap* transition (Nivre, 2009; Nivre et al., 2009), an alternative solution uses two planes and a *switch* transition to switch between the two planes (Gómez-Rodríguez and Nivre, 2010).

Since we use the scoring model of a graph-based parser, we briefly review releated work on graph-based parsing. The most well known graph-based parser is the MST (maximum spanning tree) parser, cf. (McDonald et al., 2005; McDonald and Pereira, 2006). The idea of the MST parser is to find the highest scoring tree in a graph that contains all possible edges. Eisner (1996) introduced a dynamic programming algorithm to solve this problem efficiently. Carreras (2007) introduced the left-most and right-most grandchild as factors. We use the factor model of Carreras (2007) as starting point for our experiments, cf. Section 4. We extend Carreras (2007) graph-based model with factors involving three edges similar to that of Koo and Collins (2010).

3 Transition-based Parser with a Beam

This section specifies the transition-based beam-search parser underlying the combined approach more formally. Sec. 4 will discuss the graph-based scoring model that we are adding.

The input to the parser is a word string x, the goal is to find the optimal set y of labeled edges $x_i \rightarrow_l x_j$ forming a dependency tree over x $\cup\{root\}$. We characterize the state of a transition-based parser as π_i=$\langle\sigma_i, \beta_i, y_i, h_i\rangle$, $\pi_i \in \Pi$, the set of possible states. σ_i is a stack of words from x that are still under consideration; β_i is the input buffer, the suffix of x yet to be processed; y_i the set of labeled edges already assigned (a partial labeled dependency tree); h_i is a sequence recording the history of transitions (from the set of operations Ω = {shift, left-arc$_l$, right-arc$_l$, reduce, swap}) taken up to this point.

(1) The initial state π_0 has an empty stack, the input buffer is the full input string x, and the edge set is empty. (2) The (partial) transition function $\tau(\pi_i, t) : \Pi \text{ x } \Omega \rightarrow \Pi$ maps a state and an operation t to a new state π_{i+1}. (3) Final states π_f are characterized by an empty input buffer and stack; no further transitions can be taken.

The transition function is informally defined as follows: The **shift** transition removes the first element of the input buffer and pushes it to the stack.
The **left-arc**$_l$ transition adds an edge with label l from the first word in the buffer to the word on top of the stack, removes the top element from the stack and pushes the first element of the input buffer to the stack.
The **right-arc**$_l$ transition adds an edge from word on top of the stack to the first word in the input buffer and removes the top element of the input buffer and pushes that element onto the stack.
The **reduce** transition pops the top word from the stack.
The **swap** changes the order of the two top elements on the stack (possibly generating non-projkective trees).

When more than one operation is applicable, a scoring function assigns a numerical value (based on a feature vector and a weight vector trained by supervised machine learning) to each possible continuation. When using a beam search approach with beam size k, the highest-scoring k alternative states with the same length n of transition history h are kept in a set "beam$_n$".

In the beam-based parsing algorithm (cf. the pseudo code in Algorithm 1), all candidate states for the next set "beam$_{n+1}$" are determined using the transition function τ, but based on the scoring function, only the best k are preserved. (Final) states to which no more transitions apply are copied to the next state set. This means that once all transition paths have reached a final state, the overall best-scoring states can be read off the final "beam$_n$". The y of the top-scoring state is the predicted parse.

Under the plain transition-based scoring regime score$_T$, the score for a state π is the sum of the "local" scores for the transitions t_i in the state's history sequence:

$$\text{score}_T(\pi) = \sum_{i=0}^{|h|} w \cdot f(\pi_i, t_i)$$

Algorithm 1: Transition-based parser

```
// x is the input sentence, k is the beam size
σ0 = ∅, β0 = x, y0 = ∅, h = ∅
π0 ← ⟨σ0, β0, y0, h0⟩ // initial parts of a state
beam0 ← {π0} // create initial state
n ← 0 // iteration
repeat
  n ← n + 1
  for all πj ∈ beam_{n−1} do
    transitions ← possible-applicable-transition (πj)
    // if no transition is applicable keep state πj:
    if transitions = ∅ then beam_n ← beam_n ∪ {πj}
    else for all ti ∈ transitions do
        // apply the transition i to state j
        π ← τ(πj, ti)
        beam_n ← beam_n ∪ {π}
    // end for
  // end for
  sort beam_n due to the score(πj)
  beam_n ← sublist (beam_n, 0, k)
until beam_{n−1} = beam_n // beam changed?
```

w is the weight vector. Note that the features $f(\pi_i, t_i)$ can take into account all structural and labeling information available prior to taking transition t_i, i.e., the graph built so far, the words (and their part of speech etc.) on the stack and in the input buffer, etc. But if a larger graph configuration involving the next word evolves only later, as in Figure 1, this information is not taken into account in scoring. For instance, if the feature extraction uses the subcategorization frame of a word under consideration to compute a score, it is quite possible that some dependents are still missing and will only be attached in a future transition.

4 Completion Model

We define an augmented scoring function which can be used in the same beam-search algorithm in order to ensure that in the scoring of alternative transition paths, larger configurations can be exploited as they are completed in the incremental process. The feature configurations can be largely taken from graph-based approaches. Here, spans from the string are assembled in a bottom-up fashion, and the scoring for an edge can be based on structurally completed subspans ("factors").

Our completion model for scoring a state π_n incorporates factors for all configurations (matching the extraction scheme that is applied) that are present in the partial dependency graph y_n built up to this point, which is continuously augmented. This means if at a given point n in the transition path, complete information for a particular configuration (e.g., a third-order factor involving a head, its dependent and its grand-child dependent) is unavailable, scoring will ignore this factor at time n, but the configuration will inform the scoring later on, maybe at point $n + 4$, when the complete information for this factor has entered the partial graph y_{n+4}.

We present results for a number of different second-order and third-order feature models.

Second Order Factors. We start with the model introduced by Carreras (2007). Figure 3 illustrates the factors used.

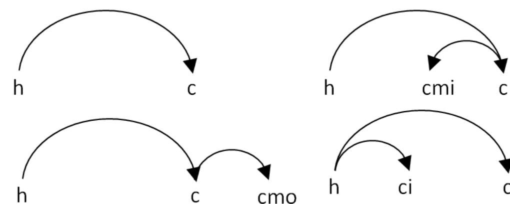

Figure 3: Model 2a. Second order factors of Carreras (2007). We omit the right-headed cases, which are mirror images. The model comprises a factoring into one first order part and three second order factors (2-4): 1) The head (h) and the dependent (c); 2) the head, the dependent and the left-most (or right-most) grandchild in between (cmi); 3) the head, the dependent and the right-most (or left-most) grandchild away from the head (cmo). 4) the head, the dependent and between those words the right-most (or left-most) sibling (ci).

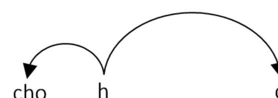

Figure 4: 2b. The left-most dependent of the head or the right-most dependent in the right-headed case.

Figure 4 illustrates a new type of factor we use, which includes the left-most dependent in the left-headed case and symmetricaly the right-most sibling in the right-head case.

Third Order Factors. In addition to the second order factors, we investigate combinations of third order factors. Figure 5 and 6 illustrate the third order factors, which are similar to the factors of Koo and Collins (2010). They restrict the factor to the innermost sibling pair for the *tri-siblings*

and the outermost pair for the *grand-siblings*. We use the first two siblings of the dependent from the left side of the head for the tri-siblings and the first two dependents of the child for the grand-siblings. With these factors, we aim to capture non-projective edges and subcategorization information. Figure 7 illustrates a factor of a sequence of four nodes. All the right headed variants are symmetrically and left out for brevity.

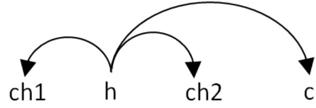

Figure 5: 3a. The first two children of the head, which do not include the edge between the head and the dependent.

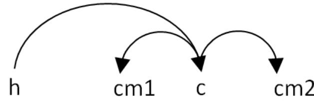

Figure 6: 3b. The first two children of the dependent.

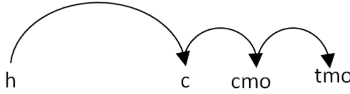

Figure 7: 3c. The right-most dependent of the right-most dependent.

Integrated approach. To obtain an integrated system for the various feature models, the scoring function of the transition-based parser from Section 3 is augmented by a family of scoring functions score_{G_m} for the completion model, where m is from 2a, 2b, 3a etc., x is the input string, and y is the (partial) dependency tree built so far:

$$\text{score}_{T_m}(\pi) = \text{score}_T(\pi) + \text{score}_{G_m}(x, y)$$

The scoring function of the completion model depends on the selected factor model G_m. The model G_{2a} comprises the edge factoring of Figure 3. With this model, we obtain the following scoring function.

$$\text{score}_{G_{2a}}(x, y) = \sum_{(h,c) \in y} w \cdot f_{first}(x,h,c)$$
$$+ \sum_{(h,c,ci) \in y} w \cdot f_{sib}(x,h,c,ci)$$
$$+ \sum_{(h,c,cmo) \in y} w \cdot f_{gra}(x,h,c,cmo)$$
$$+ \sum_{(h,c,cmi) \in y} w \cdot f_{gra}(x,h,c,cmi)$$

The function f maps the input sentence x, and a subtree y defined by the indexes to a feature-vector. Again, w is the corresponding weight vector. In order to add the factor of Figure 4 to our model, we have to add the scoring function (2a) the sum:

(2b) $\text{score}_{G_{2b}}(x, y) = \text{score}_{G_{2a}}(x, y) + \sum_{(h,c,cmi) \in y} w \cdot f_{gra}(x,h,c,cmi)$

In order to build a scoring function for combination of the factors shown in Figure 5 to 7, we have to add to the equation 2b one or more of the following sums:

(3a) $\sum_{(h,c,ch1,ch2) \in y} w \cdot f_{gra}(x,h,c,ch1,ch2)$

(3b) $\sum_{(h,c,cm1,cm2) \in y} w \cdot f_{gra}(x,h,c,cm1,cm2)$

(3c) $\sum_{(h,c,cmo,tmo) \in y} w \cdot f_{gra}(x,h,c,cmo,tmo)$

Feature Set. The feature set of the transition model is similar to that of Zhang and Nivre (2011). In addition, we use the cross product of morphologic features between the head and the dependent since we apply also the parser on morphologic rich languages.

The feature sets of the completion model described above are mostly based on previous work (McDonald et al., 2005; McDonald and Pereira, 2006; Carreras, 2007; Koo and Collins, 2010). The models denoted with + use all combinations of words before and after the head, dependent, sibling, grandchilrden, etc. These are respectively three-, and four-grams for the first order and second order. The algorithm includes these features only the words left and right do not overlap with the factor (e.g. the head, dependent, etc.). We use feature extraction procedure for second order, and third order factors. Each feature extracted in this procedure includes information about the position of the nodes relative to the other nodes of the part and a factor identifier.

Training. For the training of our parser, we use a variant of the perceptron algorithm that uses the Passive-Aggressive update function, cf. (Freund and Schapire, 1998; Collins, 2002; Crammer et al., 2006). The Passive-Aggressive perceptron uses an aggressive update strategy by modifying the weight vector by as much as needed to classify correctly the current example, cf. (Crammer et al., 2006). We apply a random function (hash function) to retrieve the weights from the weight vector instead of a table. Bohnet (2010) showed that the Hash Kernel improves parsing speed and accuracy since the parser uses additionaly negative features. Ganchev and Dredze (2008) used

this technique for structured prediction in NLP to reduce the needed space, cf. (Shi et al., 2009). We use as weight vector size 800 million. After the training, we counted 65 millions non zero weights for English (penn2malt), 83 for Czech and 87 millions for German. The feature vectors are the union of features originating from the transition sequence of a sentence and the features of the factors over all edges of a dependency tree (e.g. G_{2a}, etc.). To prevent over-fitting, we use averaging to cope with this problem, cf. (Freund and Schapire, 1998; Collins, 2002). We calculate the error e as the sum of all attachment errors and label errors both weighted by 0.5. We use the following equations to compute the update.

loss: $l_t = e$-(score$_T(x_t^g, y_t^g)$-score$_T(x_t, y_t)$)

PA-update: $\tau_t = \frac{l_t}{||f_g - f_p||^2}$

We train the model to select the transitions and the completion model together and therefore, we use one parameter space. In order to compute the weight vector, we employ standard online learning with 25 training iterations, and carry out early updates, cf. Collins and Roark (2004; Zhang and Clark (2008).

Efficient Implementation. Keeping the scoring with the completion model tractable with millions of feature weights and for second- and third-order factors requires careful bookkeeping and a number of specialized techniques from recent work on dependency parsing.

We use two variables to store the scores (a) for complete factors and (b) for incomplete factors. The complete factors (first-order factors and higher-order factors for which further augmentation is structurally excluded) need to be calculated only once and can then be stored with the tree factors. The incomplete factors (higher-order factors whose node elements may still receive additional descendants) need to be dynamically recomputed while the tree is built.

The parsing algorithm only has to compute the scores of the factored model when the transition-based parser selects a left-arc or right-arc transition and the beam has to be sorted. The parser sorts the beam when it exceeds the maximal beam size, in order to discard superfluous parses or when the parsing algorithm terminates in order to select the best parse tree. The complexity of the transition-based parser is quadratic due to swap operation in the worse case, which is rare, and $O(n)$ in the best case, cf. (Nivre, 2009). The beam size B is constant. Hence, the complexity is in the worst case $O(n^2)$.

The parsing time is to a large degree determined by the feature extraction, the score calculation and the implementation, cf. also (Goldberg and Elhadad, 2010). The transition-based parser is able to parse 30 sentences per second. The parser with completion model processes about 5 sentences per second with a beam size of 80. Note, we use a rich feature set, a completion model with third order factors, negative features, and a large beam. [3]

We implemented the following optimizations:
(1) We use a parallel feature extraction for the beam elements. Each process extracts the features, scores the possible transitions and computes the score of the completion model. After the extension step, the beam is sorted and the best elements are selected according to the beam size.
(2) The calculation of each score is optimized (beyond the distinction of a static and a dynamic component): We calculate for each location determined by the last element $s_l \in \sigma_i$ and the first element of $b_0 \in \beta_i$ a numeric feature representation. This is kept fix and we add only the numeric value for each of the edge labels plus a value for the transition left-arc or right-arc. In this way, we create the features incrementally. This has some similarity to Goldberg and Elhadad (2010).
(3) We apply edge filtering as it is used in graph-based dependency parsing, cf. (Johansson and Nugues, 2008), i.e., we calculate the edge weights only for the labels that were found for the part-of-speech combination of the head and dependent in the training data.

5 Parsing Experiments and Discussion

The results of different parsing systems are often hard to compare due to differences in phrase structure to dependency conversions, corpus version, and experimental settings. For better comparison, we provide results on English for two commonly used data sets, based on two different conversions of the Penn Treebank. The first uses the Penn2Malt conversion based on the head-

[3] 6 core, 3.33 Ghz Intel Nehalem

	Section	Sentences	PoS Acc.
Training	2-21	39.832	97.08
Dev	24	1.394	97.18
Test	23	2.416	97.30

Table 1: Overview of the training, development and test data split converted to dependency graphs with head-finding rules of (Yamada and Matsumoto, 2003). The last column shows the accuracy of Part-of-Speech tags.

finding rules of Yamada and Matsumoto (2003). Table 1 gives an overview of the properties of the corpus. The annotation of the corpus does not contain non-projective links. The training data was 10-fold jackknifed with our own tagger.[4]. Table 1 shows the tagging accuracy.

Table 2 lists the accuracy of our transition-based parser with completion model together with results from related work. All results use predicted PoS tags. As a baseline, we present in addition results without the completion model and a graph-based parser with second order features (G_{2a}). For the Graph-based parser, we used 10 training iterations. The following rows denoted with T_a, T_{2a}, T_{2ab}, T_{2ab3a}, T_{2ab3b}, T_{2ab3bc}, and T_{2a3abc} present the result for the parser with completion model. The subscript letters denote the used factors of the completion model as shown in Figure 3 to 7. The parsers with subscribed plus (e.g. G_{2a+}) in addition use feature templates that contain one word left or right of the head, dependent, siblings, and grandchildren. We left those feature in our previous models out as they may interfere with the second and third order factors. As in previous work, we exclude punctuation marks for the English data converted with Penn2Malt in the evaluation, cf. (McDonald et al., 2005; Koo and Collins, 2010; Zhang and Nivre, 2011).[5] We optimized the feature model of our parser on section 24 and used section 23 for evaluation. We use a beam size of 80 for our transition-based parser and 25 training iterations.

The second English data set was obtained by using the *LTH* conversion schema as used in the CoNLL Shared Task 2009, cf. (Hajič et al., 2009). This corpus preserves the non-projectivity of the phrase structure annotation, it has a rich edge label set, and provides automatic assigned PoS tags. From the same data set, we selected the corpora for Czech and German. In all cases, we used the provided training, development, and test data split, cf. (Hajič et al., 2009). In contrast to the evaluation of the Penn2Malt conversion, we include punctuation marks for these corpora and follow in that the evaluation schema of the CoNLL Shared Task 2009. Table 3 presents the results as obtained for these data set.

Parser	UAS	LAS
(McDonald et al., 2005)	90.9	
(McDonald and Pereira, 2006)	91.5	
(Huang and Sagae, 2010)	92.1	
(Zhang and Nivre, 2011)	92.9	
(Koo and Collins, 2010)	93.04	
(Martins et al., 2010)	93.26	
T (baseline)	92.7	
G_{2a} (baseline)	92.89	
T_{2a}	92.94	91.87
T_{2ab}	93.16	92.08
T_{2ab3a}	93.20	92.10
T_{2ab3b}	93.23	92.15
T_{2ab3c}	93.17	92.10
$T_{2ab3abc+}$	93.39	92.38
G_{2a+}	93.1	
(Koo et al., 2008) †	93.16	
(Carreras et al., 2008) †	93.5	
(Suzuki et al., 2009) †	93.79	

Table 2: English **Attachment Scores** for the Penn2Malt conversion of the Penn Treebank for the test set. Punctuation is excluded from the evaluation. The results marked with † are not directly comparable to our work as they depend on additional sources of information (Brown Clusters).

The transition-based parser obtains higher accuracy scores for Czech but still lower scores for English and German. For Czech, the result of T is 1.59 percentage points higher than the top labeled score in the CoNLL shared task 2009. The reason is that T includes already third order features that are needed to determine some edge labels. The transition-based parser with completion model T_{2a} has even 2.62 percentage points higher accuracy and it could improve the results of the parser T by additional 1.03 percentage points. The results of the parser T are lower for English and German compared to the results of the graph-based parser G_{2a}. The completion model T_{2a} can reach a similar accuracy level for these two languages. The third order features let the transition-based parser reach higher scores than the graph-based parser. The third order features contribute for each language a relatively small improvement

[4] http://code.google.com/p/mate-tools/

[5] We follow Koo and Collins (2010) and ignore any token whose POS tag is one of the following tokens `` '' : , .

Parser	Eng.	Czech	German
(Gesmundo et al., 2009)†	88.79/-	80.38	87.29
(Bohnet, 2009)	89.88/-	80.11	87.48
T (Baseline)	89.52/92.10	81.97/87.26	87.53/89.86
G_{2a} (Baseline)	90.14/92.36	81.13/87.65	87.79/90.12
T_{2a}	90.20/92.55	83.01/88.12	88.22/90.36
T_{2ab}	90.26/92.56	83.22/88.34	88.31/90.24
T_{2ab3a}	90.20/90.51	83.21.88.30	88.14/90.23
T_{2ab3b}	90.26/92.57	83.22/88.35	88.50/90.59
$T_{2ab3abc}$	90.31/92.58	83.31/88.30	88.33/90.45
G_{2a+}	90.39/92.8	81.43/88.0	88.26/90.50
$T_{2ab3ab+}$	90.36/92.66	83.48/88.47	88.51/90.62

Table 3: **Labeled Attachment Scores** of parsers that use the data sets of the CoNLL shared task 2009. In line with previous work, punctuation is included. The parsers marked with † used a joint model for syntactic parsing and semantic role labelling. We provide more parsing results for the languages of CoNLL-X Shared Task at http://code.google.com/p/mate-tools/.

Parser	UAS	LAS
(Zhang and Clark, 2008)	84.3	
(Huang and Sagae, 2010)	85.2	
(Zhang and Nivre, 2011)	86.0	84.4
$T_{2ab3abc+}$	87.5	85.9

Table 4: Chinese **Attachment Scores** for the conversion of CTB 5 with head rules of Zhang and Clark (2008). We take the standard split of CTB 5 and use in line with previous work gold segmentation, POS-tags and exclude punctuation marks for the evaluation.

of the score. Small and statistically significant improvements provides the additional second order factor (2b).[6] We tried to determine the best third order factors or set of factors but we cannot denote such a factor which is the best for all languages. For German, we obtained a significant improvement with the factor (3b). We believe that this is due to the flat annotation of PPs in the German corpus. If we combine all third order factors we obtain for the Penn2Malt conversion a small improvement of 0.2 percentage points over the results of (2ab). We think that a more deep feature selection for third order factors may help to improve the actuary further.

In Table 4, we present results on the Chinese Treebank. To our knowledge, we obtain the best published results so far.

[6]The results of the baseline T compared to $T_{2ab3abc}$ are statistically significant (p $<$ 0.01).

6 Conclusion and Future Work

The parser introduced in this paper combines advantageous properties from the two major paradigms in data-driven dependency parsing, in particular worst case quadratic complexity of transition-based parsing with a swap operation and the consideration of *complete* second and third order factors in the scoring of alternatives. While previous work using third order factors, cf. Koo and Collins (2010), was restricted to unlabeled and projective trees, our parser can produce labeled and non-projective dependency trees.

In contrast to parser stacking, which involves running two parsers in training and application, we use only the feature model of a graph-based parser but not the graph-based parsing algorithm. This is not only conceptually superior, but makes training much simpler, since no jackknifing has to be carried out. Zhang and Clark (2008) proposed a similar combination, without the rescoring procedure. Our implementation allows for the use of rich feature sets in the combined scoring functions, and our experimental results show that the "graph-based" completion model leads to an increase of between 0.4 (for English) and about 1 percentage points (for Czech). The scores go beyond the current state of the art results for typologically different languages such as Chinese, Czech, English, and German. For Czech, English (Penn2Malt) and German, these are to our knowlege the highest reported scores of a dependency parser that does not use additional sources of information (such as extra unlabeled training data for clustering). Note that the efficient techniques and implementation such as the Hash Kernel, the incremental calculation of the scores of the completion model, and the parallel feature extraction as well as the parallelized transition-based parsing strategy play an important role in carrying out this idea in practice.

References

S. Abney. 1991. Parsing by chunks. In *Principle-Based Parsing*, pages 257–278. Kluwer Academic Publishers.

G. Attardi. 2006. Experiments with a Multilanguage Non-Projective Dependency Parser. In *Tenth Conference on Computational Natural Language Learning (CoNLL-X)*.

B. Bohnet. 2009. Efficient Parsing of Syntactic and

Semantic Dependency Structures. In *Proceedings of the 13th Conference on Computational Natural Language Learning (CoNLL-2009)*.

B. Bohnet. 2010. Top accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 89–97, Beijing, China, August. Coling 2010 Organizing Committee.

X. Carreras, M. Collins, and T. Koo. 2008. Tag, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, CoNLL '08, pages 9–16, Stroudsburg, PA, USA. Association for Computational Linguistics.

X. Carreras. 2007. Experiments with a Higher-order Projective Dependency Parser. In *EMNLP/CoNLL*.

M. Collins and B. Roark. 2004. Incremental parsing with the perceptron algorithm. In *ACL*, pages 111–118.

M. Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *EMNLP*.

K. Crammer, O. Dekel, S. Shalev-Shwartz, and Y. Singer. 2006. Online Passive-Aggressive Algorithms. *Journal of Machine Learning Research*, 7:551–585.

J. Eisner. 1996. Three New Probabilistic Models for Dependency Parsing: An Exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 340–345, Copenhaen.

Y. Freund and R. E. Schapire. 1998. Large margin classification using the perceptron algorithm. In *11th Annual Conference on Computational Learning Theory*, pages 209–217, New York, NY. ACM Press.

K. Ganchev and M. Dredze. 2008. Small statistical models by random feature mixing. In *Proceedings of the ACL-2008 Workshop on Mobile Language Processing*. Association for Computational Linguistics.

A. Gesmundo, J. Henderson, P. Merlo, and I. Titov. 2009. A Latent Variable Model of Synchronous Syntactic-Semantic Parsing for Multiple Languages. In *Proceedings of the 13th Conference on Computational Natural Language Learning (CoNLL-2009)*, Boulder, Colorado, USA., June 4-5.

Y. Goldberg and M. Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *HLT-NAACL*, pages 742–750.

C. Gómez-Rodríguez and J. Nivre. 2010. A Transition-Based Parser for 2-Planar Dependency Structures. In *ACL*, pages 1492–1501.

J. Hajič, M. Ciaramita, R. Johansson, D. Kawahara, M. Antònia Martí, L. Màrquez, A. Meyers, J. Nivre, S. Padó, J. Štěpánek, P. Straňák, M. Surdeanu, N. Xue, and Y. Zhang. 2009. The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*, pages 1–18, Boulder, United States, June.

L. Huang and K. Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086, Uppsala, Sweden, July. Association for Computational Linguistics.

R. Johansson and P. Nugues. 2006. Investigating multilingual dependency parsing. In *Proceedings of the Shared Task Session of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 206–210, New York City, United States, June 8-9.

R. Johansson and P. Nugues. 2008. Dependency-based Syntactic–Semantic Analysis with PropBank and NomBank. In *Proceedings of the Shared Task Session of CoNLL-2008*, Manchester, UK.

S. Kahane, A. Nasr, and O. Rambow. 1998. Pseudo-projectivity: A polynomially parsable non-projective dependency grammar. In *COLING-ACL*, pages 646–652.

T. Koo and M. Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11, Uppsala, Sweden, July. Association for Computational Linguistics.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. pages 595–603.

T. Kudo and Y. Matsumoto. 2002. Japanese dependency analysis using cascaded chunking. In *proceedings of the 6th conference on Natural language learning - Volume 20*, COLING-02, pages 1–7, Stroudsburg, PA, USA. Association for Computational Linguistics.

M. Kuhlmann, C. Gómez-Rodríguez, and G. Satta. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *ACL*, pages 673–682.

Andre Martins, Noah Smith, Eric Xing, Pedro Aguiar, and Mario Figueiredo. 2010. Turbo parsers: Dependency parsing by approximate variational inference. pages 34–44.

R. McDonald and F. Pereira. 2006. Online Learning of Approximate Dependency Parsing Algorithms. In *In Proc. of EACL*, pages 81–88.

R. McDonald, K. Crammer, and F. Pereira. 2005. Online Large-margin Training of Dependency Parsers. In *Proc. ACL*, pages 91–98.

J. Nivre and R. McDonald. 2008. Integrating Graph-Based and Transition-Based Dependency Parsers. In *ACL-08*, pages 950–958, Columbus, Ohio.

J. Nivre and J. Nilsson. 2005. Pseudo-projective dependency parsing. In *ACL*.

J. Nivre, M. Kuhlmann, and J. Hall. 2009. An improved oracle for dependency parsing with online reordering. In *Proceedings of the 11th International Conference on Parsing Technologies*, IWPT '09, pages 73–76, Stroudsburg, PA, USA. Association for Computational Linguistics.

J. Nivre. 2003. An Efficient Algorithm for Projective Dependency Parsing. In *8th International Workshop on Parsing Technologies*, pages 149–160, Nancy, France.

J. Nivre. 2009. Non-Projective Dependency Parsing in Expected Linear Time. In *Proceedings of the 47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP*, pages 351–359, Suntec, Singapore.

K. Sagae and A. Lavie. 2006. Parser combination by reparsing. In *NAACL '06: Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers on XX*, pages 129–132, Morristown, NJ, USA. Association for Computational Linguistics.

Q. Shi, J. Petterson, G. Dror, J. Langford, A. Smola, and S.V.N. Vishwanathan. 2009. Hash Kernels for Structured Data. In *Journal of Machine Learning*.

J. Suzuki, H. Isozaki, X. Carreras, and M Collins. 2009. An empirical study of semi-supervised structured conditional models for dependency parsing. In *EMNLP*, pages 551–560.

I. Titov and J. Henderson. 2007. A Latent Variable Model for Generative Dependency Parsing. In *Proceedings of IWPT*, pages 144–155.

H. Yamada and Y. Matsumoto. 2003. Statistical Dependency Analysis with Support Vector Machines. In *Proceedings of IWPT*, pages 195–206.

Y. Zhang and S. Clark. 2008. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of EMNLP*, Hawaii, USA.

Y. Zhang and J. 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, Portland, Oregon, USA, June. Association for Computational Linguistics.