Abstract

Parser combination is an effective technique for increasing the accuracy of dependency parsers. The typical approach is to use independent parsers, often based on a few variants of the same algorithm. In this paper we introduce both a new parsing method and a new parser combination algorithm. The new parsing method uses the output of a previous parse as hint while parsing the same text in the reverse direction. The new parser combination algorithm exploits the fact that its inputs are trees in order to avoid the quadratic cost of algorithms for computing the maximum spanning tree of a graph. We present the method and discuss the results of experiments performed on corpora from the CoNLL X and 2007 shared tasks. An analysis of results shows that the accuracy improvements are often due to more correct predictions of long span dependencies. Both the proposed techniques maintain a linear or quasi-linear computational cost, hence the resulting parser achieves both high accuracy and performance.

1 Introduction

The comparative error analysis performed by Nivre and McDonald (2007) revealed as a weakness of deterministic transition-based Shift/Reduce dependency parsers a low accuracy in the analysis of long span dependencies.

Experiments on the CoNLL 2007 corpora (Dell’Orletta, 2008) have shown that for English this occurs especially for distances in the range from 6 to 25.

The parser by Titov and Henderson (2007) addresses this accuracy drop by using a beam search instead of a greedy algorithm for predicting the next parser transition. However, employing a beam search method increases the otherwise linear computational cost of a deterministic Shift/Reduce parsing algorithm.

We propose a parsing method that allows reducing several of the errors made by a Shift/Reduce parser when dependency links span a long distance on the input sequence, albeit without any noticeable increase in computational costs.

The method consists in two steps: first the sentence is parsed by a deterministic Shift/Reduce parser, then a second deterministic Shift/Reduce parser analyses the sentence in reverse using additional features extracted from the parse trees by the first parser. As both steps are linear, so is the overall two-step process. The approach was used in the CoNLL 2008 Shared Task by Ciaramita et al. (2007) and is similar to the stacking approach independently developed by Nivre and McDonald (2008).

The use of a Right-to-Left parser was proposed by Sagae and Lavie (2006), as part of an ensemble-based parser. The parse trees produced by each parser in the ensemble are turned into a graph weighted according to the accuracy of each parser and the combined parse tree is computed as the maximum spanning tree of this graph.

Calculating the maximum spanning tree of a graph is an expensive process. The Chu-Liu/Edmonds algorithm (Chu & Liu, 1965; Edmonds, 1967), which is typically used both in MST parsing (McDonald et al., 2005) and ensemble-based parsers (Hall et al., 2007; Sagae & Tsujii, 2007), has quadratic time complexity, leading to an $O(n^2k^2)$ complexity when combin-
We introduce an alternative combination method that has quasi-linear complexity, i.e. linear up to a logarithmic factor. The algorithms exploits the fact that the graphs to be combined are trees. In fact we apply it to the well-formed dependency trees produced by three parsers, a Left-to-Right, a Right-to-Left and a stacked Right-to-Left parser.

In practice the combination method minimally affects the overall parser performance, which remains linear.

2 Related Work

Nivre and McDonald (2008) exploit the complementary properties of graph-based and transition-based dependency parsing models and propose a simple way of integrating these models.

In graph-based parsing, parsing is performed by searching for the highest-scoring among all possible dependency graphs for a given sentence, see for example (Eisner, 1967) and (McDonald et al., 2005).

In transition-based parsing, parsing is done greedily by performing at each parser state the highest-scoring transition until a complete dependency graph is produced. Transitions are for example Shift/Reduce actions which are learnt from corpora, based on the current parser state ((Yamada & Matsumoto, 2003), (Nivre & Scholz, 2004), (Attardi, 2006)) and possibly on history as well (Titov & Henderson, 2007).

The method by Nivre and McDonald (2008) combines the two models by allowing the output of one model to define features for the other. They evaluate the method on data sets from the CoNLL-X shared task showing that parsing accuracy is consistently improved beyond what is possible by either model in isolation.

Our method also uses input from a previous parser but only uses parsers of a single model type, deterministic transition-based Shift/Reduce, maintaining an overall linear complexity, rather than the $O(n^3)$ or $O(n^2)$ complexity of first and second order non-projective MST algorithm (McDonald & Pereira, 2006).

The use of a single type of parser has also benefits from a software engineering perspective, requiring a single code base.

The technique of parser combination was proposed by Sagae and Lavie (2007) and also used by Hall et al. (2007) with six transition-based parsers in the best performing system in the CoNLL 2007 shared task.

Integrating parsing models by using features derived from the output of a different model was explored by McDonald (2006) and Charniak (2000). Feature-based integration has been used in other linguistic tasks by Taskar et al. (2005) and Florian et al. (2004).

3 Reverse Revision Parser

Attardi and Ciaramita (2007) introduced a method for revising parse trees produced by a dependency parser. The approach is based on training a classifier to learn which revision rule to apply to incorrect parse trees in order to replace incorrect dependencies with correct ones. The approach assumed that only local revisions to the parse tree would be needed, since the dependency parser mostly gets individual phrases correctly. Hence the classifier can learn how to associate a corrective rule to a portion of a parse tree from a set of examples obtained from parsing the training corpus. Experiments showed that indeed few rules where needed, for example moving a dependency link up to the parent node, or moving it to the token on the left, or moving it to the token on the right, etc. However, since these rules were applied independently, there was no guarantee that the final tree would be a well-formed dependency tree.

Our approach refines this idea, but instead of using an independent classifier for the revision, we use the parser itself. The second parser is trained on the original corpus extended with dependency information predicted by a lower accuracy parser.

Intuitively the reverse parser can be trained to recognize where the first parser makes mistakes and learns how to avoid them.

The experiments in (Attardi & Ciaramita, 2007) showed that improvements were larger when the revision classifier was trained on the output of a low accuracy parser. This was explained by the fact that a low accuracy parser produces a larger number of errors that are helpful in training the revision classifier.

This effect was confirmed in our settings as well. The best results were obtained when the reverse parser is trained on the training set extended with dependency information predicted by a low accuracy parser.
4 Shift/Reduce Parsing

Recent progress in statistical dependency parsing (see for instance the CoNLL 2007 Shared Task on multilingual dependency parsing (Nivre at al., 2007) shows that dependency parsing can be made robust and capable of achieving high accuracy in the analysis of multiple languages.

A dependency parser takes as input a sentence $s$ and returns a dependency graph $G$. Let $D = \{d_1, d_2, \ldots, d_m\}$ be the set of permissible dependency types. A dependency graph for a sentence $s = (s_1, s_2, \ldots, s_n)$ is a labeled directed graph $G = (s, A)$, such that:

- $s$ is the set of nodes, corresponding to the tokens in the input string;
- $A$ is a set of labeled arcs $(w_i, d, w_j)$, $w_{i,j} \in s$, $d \in D$; $w_j$ is called the head, $w_i$ the dependent and $d$ the dependency label;
- $\forall w_i \in s$ there is at most one arc $a \in A$, such that $a = (w_i, d, w_j)$;
- there are no cycles.

Yamada and Matsumoto (2003) proposed a transition-based dependency parser that uses a Shift/Reduce deterministic algorithm. The parser analyzes sentences bottom-up performing Shift/Reduce transitions predicted by a classifier in order to build the parse tree. A statistical classifier is used which is trained on an annotated corpus.

Nivre and Scholz (2004) proposed a variant of the approach reducing the complexity from the worst case quadratic to linear.

Attardi (2006) proposed additional rules that allow deterministic single-pass parsing as well as handling non-projective relations.

In the reported experiments we used DeSR (Attardi at al., 2007), a freely available implementation of the latter variant. The parser processes input tokens advancing on the input with Shift actions and accumulates processed tokens on a stack with Reduce actions. The parsing algorithm is fully deterministic and has linear complexity.

The state of the parser is represented by a quadruple $(S, I, T, A)$ where $S$ is the stack, $I$ is the list of input tokens that remain to be processed, $T$ is a stack of temporary tokens and $A$ is the arc relation for the dependency graph.

Given an input sentence $W$, the parser is initialized to $(\emptyset, W, \emptyset, \emptyset)$, and applies its parsing rules transforming its state until it reaches the configuration $(S, (,), (,))$. An example of a parsing rule is the Left$_d$ reduction, which creates a dependency link with label $d$ from the next input token $n$ to the token $s$ on top of the stack and moves $s$ back to the input queue:

$$\text{Left}_d \colon (s|n|I,T,A) \rightarrow (S|s|I,T,\cup\{n,d,s\})$$

For the full list of parsing rules we refer to (Attardi, 2006).

5 Experiments

An experimental evaluation of the parsing methods was done using data from the CoNLL-X and CoNLL 2007 shared tasks (Buchholz & Marsi, 2006; Nivre at al., 2007).

For the Left-to-Right parser and the Reverse Revision parser we employed an SVM classifier while a Maximum Entropy classifier, with lower accuracy, was used to create the training set for the Reverse Revision parser.

The Reverse Revision parser exploits the same basic set of features as in the Left-to-Right parser, which were chosen with a series of experiments through cross validation on the training set.

Table 1 lists the additional features used in the Reverse Revision parser, where: PHLEMMA is the lemma of the predicted head, PHPOS is the Part of Speech of the predicted head, PDEP is the predicted dependency label of a token to its predicted head, PHDIST indicates whether a token is located before or after its predicted head, PHLemma is the lemma of the predicted grandparent and PHDEP is the predicted dependency label of the predicted head of a token to the predicted grandparent. Features were extracted for the tokens mentioned in the second column, where positive numbers refer to tokens on the input queue while negative numbers refer to tokens on the stack.

These features and tokens were chosen through a series of tests on the English corpus among a set of manually prepared features models. They might not be the best choice for other languages.

The feature model resulting from the union of the basic and additional features was used in training the Reverse Revision parser for each language.

We present experiments and comparative error analysis on three languages from the CoNLL 2007
shared task: Italian, Czech and English. We also report an evaluation on all thirteen languages of the CoNLL X shared task, for comparison with the results by Nivre and McDonald (2008).

The training sets from the CoNLL 2007 shared task (Nivre at al., 2007) consist of 71,000 tokens for the the Italian language, 432,000 tokens for Czech and 447,000 tokens for English. The corpora from the CoNLL 2006 shared task (Nivre at al., 2007) range from 29,000 tokens for Slovene to 1,249,000 tokens for Czech. The test sets are about 5,000 tokens for all languages.

The three languages from the CoNLL 2007 shared task were selected in order to provide a coverage of languages with different characteristics.

English is a language with minimal inflection and presents a fixed word order. These two linguistic characteristics, together with the large size of its training corpus make English one of the languages for which dependency parsers achieve the best accuracy.

Czech is a highly inflected language and presents a relatively free word order. These linguistic characteristics make Czech one of the languages that are most difficult to analyse, despite the availability of a large training set.

Italian is an inflected language and presents a relatively free word order. Differently from Czech, Italian does not have cases (with the only marginal exception of personal pronouns) and allows less word order freedom (Dell’Orletta et al., 2006). Moreover the CoNLL training set is quite smaller.

Table 2 presents the results, scored in terms of Labeled Accuracy Score (LAS), for the Left-to-right parser (LR), Right-to-Left (RL), Reverse Revision parser (Rev2) and parser combination (Comb).

The Left-to-Right parser and the Reverse Revision parser accuracies will be compared relatively to the sentence length and in terms of Precision, Recall and F-Measure, relatively to dependency distance. Where the "sentence length" is the number of tokens in the sentence and the ”dependency distance” from a token $w_i$ to its head $w_j$ is equal to $|i - j|$.

**Italian**

In the case of Italian the Right-to-Left parser achieves higher accuracy than the Left-to-Right parser and the Reverse Revision parser using output from the Left-to-Right parser even higher.

Figure 1 compares the Left-to-Right and the Reverse Revision parsers with respect to dependency length, in terms of Precision, Recall and F-Measure.

The graph confirms the intuition that the Reverse Revision parser is capable of correcting errors on long dependencies by the other parser. In fact both parsers present a quite similar behaviour for dependencies up to length 3, while for greater lengths, the Reverse Revision parser, which exploits the output of the Left-to-Right parser, has noticeably higher accuracy. Such an improvement influences the whole sentence accuracy, especially for very long sentences, since these contain a higher number of long dependencies. Figure 2 shows the considerable positive effect of the Reverse Revision parser for sentences longer than 10 tokens, while below this limit both parsers behave similarly.

**Czech**

For Czech the Left-to-Left parser returns better results than the Right-to-Right parser while the Reverse Revision parser obtains the best score.

The graphs in figure 3 and 4 show that the Reverse Revision parser improves over the Left-to-Right parser everywhere except the Recall for dependencies of length between 10 and 14. Such an improvement has positive impact on the analysis of sentences longer than 10 tokens, like for Italian.

**English**

Results for English are similar to Italian.
Figure 1: Italian. Precision, Recall and F-Measure relative to dependency length.

Figure 5 and 6 present the accuracies of the Left-to-Right and Reverse Revision parsers relative to the dependency length and the length of sentences, respectively.

Differently from Italian and Czech, the accuracy improvement for English is noticeable also for sentences shorter than 10 tokens.

5.1 Conll X Results

For direct comparison with the approach by Nivre and McDonald (2008), we present the results on the Conll X corpora: MST and MST\textsubscript{Malt} are the results achieved by the MST parser and the MST parser using hints from Maltparser.

5.2 Remarks

The experiments show that the Reverse Revision parser, using dependency information predicted by a traditional Left-to-Right Shift/Reduce parser, achieves significantly improved accuracies without significant computational burden.

The graphs confirm the correctness of our intuition: exploiting the dependency information produced by the Left-to-Right parser and analysing the sentence in reverse, the Reverse Revision parser realizes that some predictions made by the first parser are not correct and is able to correct them. The graphs show in particular that the Reverse Revision parser is able to reduce errors made on long dependency links.

This kind of errors may be regarded as the weak point of the Shift/Reduce parsers compared with other parsing algorithms that are more complex and that present an higher computational cost.

In addition to accuracy gain and minimal computational cost, the method proposed has the advantage of simplicity. The base Left-to-Right parser itself is used as Reverse Revision parser, just by inverting the direction of reading and letting the latter to exploit the features predicted by the first parser.

The accuracy of the Reverse Revision parser
might further improved by more sophisticated feature selection, choosing features that better represent hints to the second parsing stage.

6 Quasi-Linear Combination

Our final improvements arise by combining the outputs of the three parser models discussed in section 5: the Left-to-Right parser, the Right-to-Left parser and the Reverse Revision parser.

Instead of using a general algorithm for calculating the maximum spanning tree of a graph, we exploit the fact that we are combining trees and hence we developed an algorithm that has $O(n \log(m))$ complexity, where $n$ is the number of nodes in a tree and $m$ is the number of arcs labeled differently in the trees being combined, which is bound to $kn$, where $k$ is the number of trees.

To each arc $a$ in the parse trees from a given parser a weight $w(a)$ is assigned corresponding to the LAS of the parser with respect to the coarse-grained part-of-speech of the dependent node in $a$. These scores were computed by performing a separate training and evaluation run using holdout validation on the training corpus.

The algorithm builds the combined tree $T$ incrementally, starting from the empty tree. We will argue that an invariant of the algorithm is that the partial result $T$ is always a tree.

The algorithm exploits the notion of fringe $F$, that is the set of arcs whose head is in $T$ and can be added to $T$ without affecting the invariant. Initially $F$ consists of the roots of all trees to be combined. The weight of each arc in the fringe is the sum of the weights of arcs in all trees having the same head, label and dependent.

At each step select from $F$ the arc $a = (d, r, h)$ where $h \in T$ and $a$ has maximum weight, then:

1. add $a$ to $T$

2. remove from $F$ all arcs whose dependent is $d$

3. add to $F$ all arcs $(d', r', h')$ in the original trees where $h' \in T$ and $d' \notin T$.

Step 3 guarantees that no cycles are present in $T$. The final $T$ is connected because each node added is connected to a previous node in $T$. $T$ is maximal because if there were another tree with higher score including arc $(n, r, h)$, either it is present in $T$ or its weight is smaller than the weight for node $(n, r', h')$ in $T$, as chosen by the algorithm.

The algorithm has $O(n \log(n))$ complexity. A sketch of the proof can be given as follows.

Step 3 guarantees that the algorithm is iterated $n$ times, where $n$ is the number of nodes in a component tree.

Using for instance self-balancing binary trees to represent the fringe $F$, insert or delete operations take $O(\log(|F|))$ time, where $|F|$ is the number of elements in $F$. $|F| < m < kn$, where $m$ is the number of different arcs in all trees, $k$ is the number of trees. At each iteration of the algorithm the maximum number of removals from $F$ (step 2) is constant and it is equal to $k$, hence the overall cost is $O(n \log(n))$. However in practice the number of differences in the parse trees is fairly small, so the practical cost is low.

Table 2 shows the results for the three languages from CoNLL 2007. Comparing the results in Table 2 with those achieved at the CoNLL 2007 Shared Task one notices that:

<table>
<thead>
<tr>
<th>Language</th>
<th>LR</th>
<th>RL</th>
<th>Rev2</th>
<th>Comb</th>
<th>Conll-X Best</th>
<th>MST</th>
<th>MST_{Malt}</th>
</tr>
</thead>
<tbody>
<tr>
<td>arabic</td>
<td>67.27</td>
<td>66.05</td>
<td>67.54</td>
<td>68.36</td>
<td>66.91</td>
<td>66.91</td>
<td>68.64</td>
</tr>
<tr>
<td>bulgarian</td>
<td>86.83</td>
<td>87.13</td>
<td>87.41</td>
<td>87.87</td>
<td>87.57</td>
<td>87.57</td>
<td>89.05</td>
</tr>
<tr>
<td>chinese</td>
<td>87.44</td>
<td>85.77</td>
<td>87.51</td>
<td>87.77</td>
<td>89.96</td>
<td>85.90</td>
<td>88.43</td>
</tr>
<tr>
<td>czech</td>
<td>79.84</td>
<td>79.46</td>
<td>81.78</td>
<td>81.76</td>
<td>80.18</td>
<td>80.18</td>
<td>82.26</td>
</tr>
<tr>
<td>danish</td>
<td>83.89</td>
<td>83.63</td>
<td>84.85</td>
<td>84.71</td>
<td>84.79</td>
<td>84.79</td>
<td>86.67</td>
</tr>
<tr>
<td>dutch</td>
<td>75.71</td>
<td>77.27</td>
<td>78.77</td>
<td>79.59</td>
<td>79.19</td>
<td>79.19</td>
<td>81.63</td>
</tr>
<tr>
<td>german</td>
<td>85.34</td>
<td>85.20</td>
<td>86.50</td>
<td>85.32</td>
<td>87.34</td>
<td>87.34</td>
<td>88.46</td>
</tr>
<tr>
<td>japanese</td>
<td>90.03</td>
<td>90.63</td>
<td>90.87</td>
<td>91.53</td>
<td>91.65</td>
<td>90.71</td>
<td>91.43</td>
</tr>
<tr>
<td>portuguese</td>
<td>86.84</td>
<td>87.00</td>
<td>87.86</td>
<td>88.14</td>
<td>87.60</td>
<td>86.82</td>
<td>87.50</td>
</tr>
<tr>
<td>slovene</td>
<td>73.64</td>
<td>74.40</td>
<td>75.32</td>
<td>75.54</td>
<td>73.44</td>
<td>73.44</td>
<td>75.94</td>
</tr>
<tr>
<td>spanish</td>
<td>81.63</td>
<td>81.61</td>
<td>81.85</td>
<td>82.33</td>
<td>82.25</td>
<td>82.25</td>
<td>83.99</td>
</tr>
<tr>
<td>swedish</td>
<td>82.95</td>
<td>81.62</td>
<td>82.91</td>
<td>83.17</td>
<td>84.58</td>
<td>82.55</td>
<td>84.66</td>
</tr>
<tr>
<td>turkish</td>
<td>64.91</td>
<td>61.92</td>
<td>63.33</td>
<td>65.25</td>
<td>65.68</td>
<td>63.19</td>
<td>64.29</td>
</tr>
<tr>
<td>Average</td>
<td>80.49</td>
<td>80.13</td>
<td>81.27</td>
<td>81.62</td>
<td>81.63</td>
<td>80.83</td>
<td>82.53</td>
</tr>
</tbody>
</table>

Table 3: Labeled attachment scores for Conll X corpora.
• for Czech we achieve the best LAS;
• for Italian we achieve the second best LAS;
• for English we achieve the second best LAS.

The results for the CoNLL 2006 languages show also significant improvements: the Reverse Revision parser is more accurate than MST, except for Bulgarian, Dutch, German, and Spanish, where the difference is within 1%.

7 Conclusions

We presented a method for improving the accuracy of a dependency parser by using a parser that analyses a sentence in reverse while exploiting the information from the trees produced by a forward parser.

We also introduced a new quasi-linear algorithm to perform parser combination.

Experiments on the corpora of languages from the CoNLL X and the CoNLL 2007 share tasks showed that reverse revision parsing alone improves the accuracy of a single parser while more significant improvements are obtained by using our parser combination algorithm on the outputs of three parsers: a Left-to-Right parser, a Right-to-Left parser and a Reverse-Revision parser.

The combination parser achieves accuracies that are best or second best with respect to the results of the CoNLL 2007 shared task and maintains an overall quasi-linear computational time. On the languages from the CoNLL X shared task the combination parser achieves often the best accuracy in seven out of 13 languages but falls short of the accuracy achieved by integrating a graph-based with a transition based parser.

We expect that further tuning of the algorithm might help reduce the differences.
Figure 5: English. Precision, Recall and F-Measure relative to dependency length.

Acknowledgments

References


Figure 6: English. Accuracy relative to sentence length.


shop on Frontiers in Linguistically Annotated Corpora, ACL 2006.


