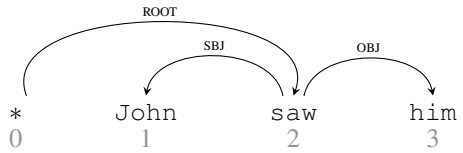
Report on ZPar Experiments

1 Overview

ZPar is an implementation of a statistical natural language parser by Zhang and Nivre (2011). In particular, it is a dependency parser trained with the perceptron algorithm; given a sentence $x = x_1 \dots x_n$, it finds a dependency tree y (a directed tree over x rooted at a special node $*$) that explains the sentence's syntactic structure. A dependency tree is characterized as a set of arcs (i, j, l) where the pair (i, j) forms a directed edge and l is the label of the edge. For example, for the sentence $x =$ `John saw him`, the corresponding dependency tree y may be the following:

$$x = \texttt{John} \quad \texttt{saw} \quad \texttt{him}$$
$$y = \{(0, 2, \text{ROOT}), (2, 1, \text{SBJ}), (2, 3, \text{OBJ})\}$$



We want to verify whether ZPar can be used to parse multiple languages at high speed and accruacy. In this report, we describe various experiments on speed and accruacy that we performed for this purpose. In section 2, we describe the English and German datasets we used. In section 3, we describe the parsing experiments with ZPar.

2 Datasets

We focused on two languages: English and German. For English, we used the Penn Treebank which consists of news articles from the Wall Street Journal. We trained on Sections 2-21 and tested on Section 23. For German, we used the TIGER corpus, which also a newswire dataset. We reserved around 80% of the data for training, and split the remaining data evenly for development and testing. The summary statistics of these datasets are shown in table 1.

There is one important detail about the data that ZPar expects: all dependency trees must be **projective**. That is, the parser does not allow arcs that cross one another. We have no problem parsing with the English dataset since all its trees are projective. However, around a third of the trees in the German training data are non-projective. Thus we first transformed the non-projective trees to projective ones using MaltParser (`http://www.maltparser.org`), and then trained on these projective trees. At test time, given a German sentence, we output a projective tree using the trained model, and then transformed it back to a (possibly) non-projective tree again using MaltParser.

3 Parsing Experiments

Firstly, there are two versions of ZPar in terms of its parsing capability:

Data	TrainSents	Train Words	Test Sents	Test Words	Dev Sents	Dev Words
English (Penn Treebank)	39832	950028	2416	56684	-	-
German (TIGER)	40000	712052	5235	88085	5237	88100

Table 1: Sumarry statistics of the datasets.

- **Language-specific version.** For certain languages, we have versions of ZPar specifically tailored to them. For instance, we have an English version of ZPar in which the labels are hard-coded. These language-specific parsers cannot be used for languages other than the ones they are designed for.
- **Generic version.** We also have a version of ZPar that can handle any (projective) languages.

Language-specific parsers are significantly (around four times) faster than the generic parser. However, we only have one for English, so we have to resort to the generic version to parse German. Secondly, we have multiple implementations of ZPar, each with some flaw as follows:

- **ZPar0.5** (released): Memory leak when number of iteration increases.
- **ZPar_acl11** (non-released model of Zhang and Nivre (2011)): No memory leak, but does not support generic parsing (i.e., it can only parse English).
- **ZPar0.6 (released)**: No memory leak. But would crash when parsing.

In practice, we can use ZPar0.5 by training for a *single* iteration multiple times to bypass the memory problem. Or we can use ZPar0.6 for training and use ZPar0.5 for testing.

3.1 English-Specific Parser

Our first goal is to replicate the default stored model, which is an English-specific parser trained by Zhang and Nivre (2011) that obtains 92.85% accuracy on the test data. To this goal, we trained an English-specific parser with different numbers of beams ($1, 2, 4, 8, 16, 32, 64, 128$) and different implementations (ZPar0.5, ZPar_acl, ZPar0.6). Then at test time, we used ZPar0.5 with 64 beams.

The result is shown in table 2. In this table, "Tagger" specifies whether we use the stored tagger provided in the ZPar package or our own tagger to tag given sentences. Interestingly, when we use our own tagger, the preformance decreases a little. As shown here, there is little difference in accuracy (and in speed) between these implementations (ZPar0.5, ZPar_acl, ZPar0.6). Hence, we will use ZPar0.5 for all of the remaining results in this report.

Version	Model	Tagger	Beam (train)	Iteration	Best Accuracy
Zpar0.5	trained model	stored tagger	64	34	92.55
Zpar_acl			64	34	92.55
Zpar0.6			64	29	92.57
	stored model	stored tagger	-	-	92.85
Zpar0.5	trained model	our tagger	64	41	92.01
Zpar_acl			64	41	92.01
Zpar0.6			128	28	91.98
	stored model	our tagger	-	-	92.43

Table 2: Replicating the "stored model". We try different Zpar implementations versions (ZPar0.5, ZPar_acl, ZPar0.6) but find that there is little difference in accuracy and speed. Here, we only report the performance of the best model out of many trained with different beam sizes ($1, 2, 4, 8, 16, 32, 64, 128$) and training iteration numbers.

Observe that ZPar_acl and ZPar0.5 are identical in performance. Although the performance of our models is slightly below that of the stored model, we can say we have essentially replicated the general performance of the given model of Zhang and Nivre (2011).

A note on the tradeoff between speed and performance. We investigate how much we can speed up the parser without sacrificing too much accuracy by parsing with different beam sizes. For this experiment, we use the stored English-specific parser of Zhang and Nivre (2011). Table 3 shows the speed and accuracy results of this model with different numbers of beams at test time. We can say that beam size of 8 might be a good compromise between speed (over 5000 words per second) and accuracy (above 92%).

Tagger	Beam (test)	Speed (words/sec)	Accuracy
stored tagger	1	14423	85.64
	2	11760	89.80
	4	8654	91.60
	8	5445	92.16
	16	2997	92.61
	32	1564	92.80
	64	799	92.85
our tagger	1	35427	85.06
	2	20389	89.46
	4	12458	91.01
	8	6708	91.58
	16	3360	91.95
	32	1608	92.22
	64	813	92.43

Table 3: Parsing with the stored English-specific parser of Zhang and Nivre (2011) with different beam sizes.

3.2 Generic Parser

As explained, the English-specific parser of the previous section cannot handle languages other than (Penn Treebank style) English. In order to be able to parse different languages, we must use the generic parser. First, we use it on the English dataset to show that it yields the same level of accuracy as the English-specific parser. We then use it on the German dataset and report performance.

3.2.1 Generic Parser on the English Dataset

We trained generic parsers with beam sizes $1, 2, 4, 8, 16, 32, 64$ on the English dataset, and tested with beam size 64. ZPar does not support a tagger for the generic parser, so we use our own tagger for testing. The best model's performance is reported in table 4; for comparison, we also include the best result we get with an English-specific parser. From this, we can conclude that the accuracy of a generic parser on the English dataset is comparable to the accuracy of an English-specific parser. However, the generic parser is much slower in speed, around four times slower when parsing with 64 beams.

Depparser	Beam (train)	Beam (test)	Iteration	Speed (words/sec)	Accuracy
English	64	64	41	754	92.01
Generic	64	64	42	160	91.86

Table 4: Comparison between Generic and English parser on English data

We also give additional speed experiments with the generic parser on English parsing in table 5. Comparing with table 3, we can see that the speed gap between the generic English parser and the English-specific parser grows as the beam size increases.

Beam (test)	Iteration	Speed (words/sec)	Accuracy
1	5	17881	85.32
2	6	9432	89.31
4	11	4317	90.65
8	16	1802	91.27
16	13	707	91.59
32	38	322	91.73
64	42	160	91.86

Table 5: Parsing with the generic parser on the English dataset with different beam sizes—compare with table 3.

3.2.2 Generic Parser on the German Dataset

We trained a generic parser with beam size 64 on the training portion of the German TIGER corpus. Then we parsed with beam sizes $1, 2, 4, 8, 16, 32, 64$ on the development data and used the best performing beam size to parse the test set. The best performing model was trained with 30 training iterations and used 64 beams for testing. Its results are shown in table 6. The level of accuracy can be viewed as respectable.

Data	Beam (test)	Iteration	Accuracy
Development	64	30	90.94
Test	64	30	90.62

Table 6: Generic parser trained on the German TIGER corpus with beam size 64.

Table 7 shows the results of the generic German parser with different beam sizes. The total time includes the deprojectivization time. It is slower than the generic English parser since there are more arc labels for German. In the English dataset, we have 12 arc labels; in the German dataset, we have 42 arc labels.

Beam (test)	Iteration	Speed (words/sec)	Accuracy
1	9	17443	84.83
2	15	5086	88.28
4	14	2561	89.81
8	15	1181	90.41
16	30	475	90.69
32	33	181	90.83
64	30	60	90.94

Table 7: Parsing with the generic parser on the German dataset with different beam sizes.

4 Conclusion

In summary, ZPar is an efficient and effective parser that can be used to support multiple languages. In this report, we have shown that we can basically replicate the performance of the model trained by Zhang and Nivre (2011) on the Penn Treebank English dataset. We have also shown that we can use the generic version of ZPar for parsing arbitrary languages (may involve some preprocessing steps such as projectivization of the trees), although the speed is slower than the language-specific version.

References

Zhang, Y. and Nivre, J. (2011). Transition-based dependency parsing with rich non-local features. In *ACL (Short Papers)*, pages 188–193.