Tuning DeSR for Dependency Parsing of Italian

Giuseppe Attardi, Maria Simi, and Andrea Zanelli

Università di Pisa, Dipartimento di Informatica, Largo B. Pontecorvo 3,
I-56127 Pisa, Italy
{attardi,simi,andreaz}@di.unipi.it

Abstract. DeSR is a statistical transition-based dependency parser that learns from a training corpus suitable actions to take in order to build a parse tree while scanning a sentence. DeSR can be configured to use different feature models and classifier types. We tuned the parser for the Evalita 2011 corpora by performing several experiments of feature selection and also by adding some new features. The submitted run used DeSR with two additional techniques: (1) *reverse revision parsing*, which addresses the problem of long distance dependencies, by extracting hints from the output of a first parser as input to a second parser running in the opposite direction; (2) parser combination, which consists in combining the outputs of different configurations of the parser. The submission achieved best accuracy among pure statistical parsers. An analysis of the errors shows that the accuracy is quite high on half of the test set and lower on the second half, which belongs to a different domain. We propose a variant of the parsing algorithm to address these shortcomings.

Keywords: Dependency parser, shift-reduce parser, stacked parser, parser combination, Evalita.

1 Description of the System

DeSR (Dependency Shift Reduce) is a transition-based statistical parser [10] [13] which builds dependency trees while scanning a sentence and applying at each step a proper parsing action selected through a classifier based on a set of representative features of the current parse state [2]. Parsing is performed bottom-up in a *Shift/Reduce* style [1], except that the parsing rules are special and allow parsing to be performed deterministically in a single pass [2]. Beam search is used when possible in order to carry on several, among the most likely, alternative parsing actions and hence to reduce the effect of error propagation due to an early incorrect choice.

The state of the parser is represented by a triple $\langle S, I, A\rangle$, where I is a sequence of tokens still remaining in the input. Initially I contains the sequence of tokens $\langle t_1 \ldots t_n\rangle$ for the sentence being parsed: each token t_i contains the word w_i as well as a set of word features p_i, consisting typically of the POS tag, the word lemma and possibly morphological features. S is the stack containing already analyzed tokens; A is a set of labeled dependencies constructed so far. At each step, the parser selects a parsing rule to apply and modifies its state accordingly. The parsing rule is chosen by means of a classifier trained on an annotated training corpus. The input to the classifier is a

B. Magnini et al. (Eds.): EVALITA 2012, LNCS 7689, pp. 37–45, 2013.

context representing the current parser state expressed as a set of features extracted from such state.

The parser allows specifying, through a configuration file, the set of features to use (e.g. POS tag, word lemma, morphological features) and the classification algorithm (e.g. Multi-Layer Perceptron, Support Vector Machine, Maximum Entropy). The parser can use beam search in conjunction with those classifiers that provide a probability distribution for the predictions, i.e. Maximum Entropy and Perceptron classifiers. Moreover the parser can be configured to run either in left-to-right or right-to-left word order.

A quite effective use of DeSR is the *Reverse Revision* parser [3], a *stacked parser* [3] which first runs in one direction, and then extracts hints from its output to feed another parser running in the opposite direction. A Reverse Revision parser was used successfully in several parser competitions, including Evalita 2009 [5] and Icon 2010 [6].

These options allow creating a number of different parser variants, all based on the same basic algorithm. This allows selecting the most effective variants and then a further improvement can be achieved by the technique of parser combination [3]. For parser combination we use a greedy algorithm, which hence preserves the linear complexity of the individual parsers and often outperforms other more complex algorithms [11]. In the Evalita 2009 experiments, the algorithm was able to reduce the error rate up to 8% in the pilot task on dependency parsing and achieved the tied first best score of 88.67% LAS in the task [5].

For the Evalita 2011 competition, we started from a configuration similar to that for Evalita 2009 and performed a number of experiments of feature selection to improve the model.

2 Experiments

The dependency parsing task at Evalita 2011 [2] provided six corpora annotated according to the TUT guidelines:

- NEWS and VEDCH, from newspapers (700 + 400 sentences, 18,044 tokens)
- CODCIV, from the Italian Civil Law Code (1,100 sentences, 28,048 tokens)
- EUDIR, from the JRC-Acquis Corpus[5] (201 sentences, 7,455 tokens)
- Wikipedia, from Wikipedia (459 sentences, 14,746 tokens)
- COSTITA, the full text of the Costituzione Italiana (682 sentences, 13,178 tokens)

It must be noted that the TUT guidelines do not allow for non-projective trees, hence some aspects of the syntax are represented by special annotations that are partly lost when converted to the CoNLL format adopted in the task. This solution also may penalize a parser like DeSR [2] which is capable of handling non-projectivity directly.

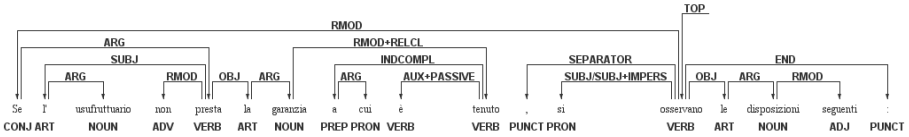

Fig. 1. An example of a TUT dependency parse tree

Figure 1 shows an example of a sentence annotated according to the TUT guidelines. One can note a tendency to favor a linear structure with articles as the heads of noun phrases ("osservano le disposizioni vigenti") and the use of a special tag (RMOD+RECL) to denote the referent of a pronoun ("cui" refers to "garanzia"), which is a typical solution to let the pronoun depend in a normal way on the verb, while this depends on the noun governing the clause.

The set of dependency tags is uncommonly large (92), as they are often made as combinations of elementary aspects. This in principle could be a problem for a parser based on classifier, since it requires an ability to discriminate among subtle categories. However the results show that the parser is capable to cope quite well with this situation.

We merged the six corpora into a single *initial corpus*. We converted this *initial corpus* by rewriting the morphological information present in the FEATS field in a more convenient format and by transferring other information from this field to the fine-grained PoS column and to two new columns: one column EXTRA that contains additional morphological information and one column SEM with semantic information. An appropriate corpus format file was created to inform DeSR of the presence of these two extra columns so that it could exploit this additional information. We will call *base corpus* the corpus with the two new columns.

During development the *base corpus* was divided randomly into: a training set (93% of sentences) for model training and a development set (7%) for model testing.

Starting from the configurations that gave the best results in the Evalita 2009 Dependency Parsing main task, we performed a feature selection process by adding and deleting individual features, and verifying the improvements brought by each of them. For each set of features we tried as classifiers both Multi-Layer Perceptron (MLP) and Support Vector Machine (SVM). Moreover, for each set of features, both the normal parser and the stacked parser were run, in both directions.

The feature selection process produced about 170 different models and as many parse results. This was possible since the parser is fast enough that training in a typical configuration takes about 3 min. on a Linux server with an Intel® Xeon® 2.53GHz CPU. The 25 best configurations were tested in combinations of 3 or 4, using the method described in [3]. The configurations of the four best parsers were chosen for the final run by training four parsers on the whole *base corpus* and combining their outputs using the algorithm from [3].

The four parser configurations selected for the final run share the set of common features reported in Table 1.

The features listed in the first 5 rows are features extracted from token attributes, as provided in the CoNLL-X format. The tokens from which such features are extracted are listed in the corresponding column in the Table: positive numbers refer to tokens from the input queue *I* (0 is the next input token, etc.), tokens from the stack *S* are numbered negatively (-1 represents the top of the stack). Operators are provided to navigate from these tokens to other tokens: *leftChild* leads to the leftmost child of a token, *rightChild* to the rightmost, *prev* to the immediately preceding, *next* to the immediately following. For example, *leftChild*(-1) in the line of the table corresponding to feature LEMMA, means to include as feature the lemma of the

closest left child of the token on top of the stack (token number -1), while *prev*(0) means to use the lemma of the token preceding the next one on the input queue (token number 0).

The last 4 lines in Table 1 describe other global features in the configuration: `LexChildNonWord` extracts a feature when the top token has children including punctuation characters, `StackSize` adds a feature corresponding to the size of the stack, `VerbCount` takes note of the number of verbs encountered so far and `PastAction` includes the value of previous past actions as a feature, in this case just of one previous action.

Notice that all configurations are partly unlexicalized, since only lemmas are used as features, discarding the form of tokens.

Table 1. Common features of all parsers

Feature	Tokens or Value
LEMMA	-2 -1 0 1 2 3 *prev*(0) *leftChild*(-1) *leftChild*(0) *rightChild*(-1) *rightChild*(0)
POSTAG	-2 -1 0 1 2 3 *next*(-1) *leftChild*(-1) *leftChild*(0) *rightChild*(-1) *rightChild*(0)
CPOSTAG	-1 0 1
FEATS	-1 0 1
DEPREL	*leftChild*(-1) *leftChild*(0) *rightChild*(-1)
LexChildNonWord	true
StackSize	true
VerbCount	true
PastActions	1

All four parsers are stacked parsers, which use an additional set of common features for the second stage, as reported in Table 2. PHLEMMA refers to the lemma of the predicted head, PDEP is the predicted dependency tag and PHPOS is the POS of the predicted head, PLOC expresses whether the predicted head occurs before or after the token.

Table 2. Common features for reverse revision parsers

Feature	Tokens
PHLEMMA	-1 0 1
PDEP	-1 0 1
PHPOS	-1 0 1

The specific features used by the four parsers selected for the final run are listed in Table 3. Most differences lie in the use of fields EXTRA and SEM. This is due to the fact that indeed the best configurations found through feature selection differed mainly in such fields: more significantly, variations in other features (with respect to the common configuration reported above) led to decay in performance.

A further experiment was performed to assess the usefulness of features expressing morphological agreement between words, either in gender and number.

The best configurations were tested by adding the feature `MorphoAgreement`. We considered two different ways to represent morphological agreement:

- Adding features `=N` or `=G` to express the cases when the top and next token agree in number or gender respectively (neutral agrees with any other value)
- Adding features `!=N` or `!=G` to express the cases when the top and next token do not agree

The second alternative avoids that a missing feature would be considered as a disagreement. For example in "potuto essere vista", "potuto" and "vista" are indirectly connected even though there is no gender agreement. In both cases also `=NG!1` and `=NG!2` is added if the top token disagrees either in gender or number with the second or third token on the input respectively.

While the addition of `MorphoAgreement` features corrects some errors due to wrong agreement, it introduces errors in other cases. Overall the accuracy improves slightly in about half of the runs (with an average variation of 0.3%) but is slightly worse in the others, and hence the effectiveness of the feature remains questionable.

Table 3. Comparison between the four best parsers

Parser	Type	Classifier	Features	Stacked Parser Features
1	Forward Revision	MLP	EXTRA -1 0 SEM -1 0 1 LexCutoff 0	EXTRA -1 0 SEM -1 0 1 PLOC -1 0 1 LexCutoff 0
2	Reverse Revision	MLP	EXTRA -1 0 SEM -2 -1 0 1 2 3 LexCutoff 2	EXTRA -1 0 SEM -1 0 1 PLOC -1 0 1 LexCutoff 0
3	Reverse Revision	SVM	EXTRA -1 0 SEM -2 -1 0 1 2 3 LexCutoff 2	EXTRA -1 0 SEM -1 0 1 PLOC -1 0 1 LexCutoff 0
4	Forward Revision	SVM	EXTRA -1 0 1 SEM -2 -1 0 1 2 3 LexCutoff 2	EXTRA -1 0 1 SEM -2 -1 0 1 2 PLOC 0 1 PHDEP -1 0 1 PHHLEMMA 0 1 LexCutoff 2

The best score on the development set (LAS 87.78%) was achieved by the combination among parsers without `MorphoAgreement`, with a small margin with respect to the combination of parsers with the feature (LAS 87.70%).

The option `LexCutoff` determines that words or lemmas that occur less than the specified number of times in the training set are collapsed to a single `<unknown>` word.

3 Results

Table 4 reports the values of Labeled Attachment Score (LAS) and Unlabeled Attachment Score (UAS) achieved by the four individual parsers and by their combination on the development set.

Table 4. Results of the four parser and their combination on the development set

Parser	LAS on Dev Set	UAS on Dev Set
1	85.34%	89.49%
2	86.67%	90.55%
3	85.90%	89.89%
4	85.05%	88.92%
Combination	87.78%	91.40%

We used this parser combination for our official submission which achieved the official scores reported in Table 5, compared with the scores of the best submission for this task.

Table 5. Final result on the Evalita 2011 test set

Run	LAS	UAS
EVALITA_11_PAR_DEP_UNIPI	89.88	93.73
Best submission to Evalita 2011	91.23	96.16

The results show a significant improvement over our previous result at Evalita 2009, from 88.67% to 89.88%.

With respect to the best submission, it must be noted that the latter uses a constraint-based parser, which enforces constraints obtained from grammatical rules, and exploits a "wide coverage lexicon …, which includes subcategorization for nouns, adjectives, verbs and adverbs" [8]. This would classify this submission as an open submission using additional resources, while our submission is a closed one, only uses data provided in the task.

After the submission we explored adding composite features, consisting of combinations of features, in the style proposed by [14]. We tried with features consisting of combinations of pairs of POS tags from top tokens on the stack and on the input, as well as their children, but there was no improvement. A possible explanation is that such combinations are useful when using linear classifier like the linear regression used in Zhang's parser but they are less effective when using a non linear classifier like MLP or SVM which we used with DeSR.

4 Discussion and Error Analysis

The official test set consists of 150 sentences from the Civil Law domain and 150 sentences from other domains. The parser achieves excellent accuracy on the first portion of the test set (92.85%), indeed the best of all submissions, while the score drops significantly on the rest of the test set (86.61%), as shown in the following table.

Table 6. Breakdown of accuracy on the test set

	Unipi		Best Submission	
Test Set	**LAS**	**UAS**	**LAS**	**UAS**
tut_test	89.88	93.73	**91.23**	**96.16**
tut_test_law	**92.85**	96.18	92.21	**97.01**
tut_test_rest	86.61	91.04	**90.16**	**95.24**

The results on the portion of the test set deriving from the Civil Law corpus is the best overall and improves also on the score of 91.75% LAS that we achieved in Evalita 2009 on the portion of the test set of the same genre.

A detailed analysis shows the following distribution of head errors according to the CPOS tag of the token, in the two subparts of the test set (Civil Law and Rest).

Table 7. Breakdown of errors according to head CPOS

CPOS	Civil Law		Rest	
	Head Errors	**%**	**Head Errors**	**%**
NOUN	5	1	20	2
PREP	48	8	67	12
ART	8	1	19	4
VERB	17	2	45	10
PUNCT	31	10	109	27
ADJ	6	3	8	3
PRON	3	1	5	3
ADV	4	3	15	11
CONJ	26	13	134	19

There is a considerable increase in errors for punctuation and conjunctions. Simply discarding the punctuation errors would increase the accuracy to 91.6%. An analysis of the errors on punctuations led to grouping them in the following categories:

Top error	errors due to incorrect identification of parse tree root
Parenthetical	error in commas surrounding a parenthetical phrase
Apposition	error in commas separating an apposition
Coordination	errors in coordinate attachment
Balance	errors in balancing punctuations, quotes or parentheses.

Indeed the parser has often difficulty in deciding where to attach a comma, since when the comma is reached it has constructed the trees for phrases preceding the comma, but it only can see individual tokens after the comma.

For example, in the sentence "… draft, cioè una bozza …", the parser would have to figure out that "bozza" relates to "draft". However "bozza" is a child of "una", which is a child of "cioè", which is a child of comma, which is a child of "draft". Hence, in order to figure out that the comma is a way to relate "draft" with "bozza" it would have to look ahead 4 tokens and be sure that the intervening tokens do not relate to something else. In order to handle this problem we have experimented with a variant of the parsing algorithm that delays *Left* reductions until the phrases on the right have been parsed. This requires also introducing an *UnShift* operation, in order to resume the *Left* reduction at the proper time. Exploring the effectviness of this approach is an argument for future research.

5 Conclusions

Dependency parsing technology has achieved satisfactory levels of accuracy and deterministic parsers like DeSR also provide excellent performance. For example parsing the Evalita 2011 test set takes 31 sec on a PC with an Intel® Xeon® 2.53GHz CPU, i.e. about 250 tokens/sec. This speed is to be compared with approximately 5 minutes required by the constraint-based parser [8] that achieved best score at Evalita 2011.

Extensive experiments of feature selection allowed us to tune the parser achieving values of LAS in the range between 86% and 88%. A final step of parser combination produced a further 2% improvement.

Our error analysis indicates that most errors are due to either semantic aspects that are outside the reach of a syntactic parser or due to the limited extent of the context examined by a transition based parser in order to select its actions. Graph-based parsers are allowed more freedom in the pairs of nodes to consider as candidates for a link, but this at the expense of typically an order of magnitude loss in running time.

An alternative approach, suggested in [7], is to perform reductions on pairs of neighbor tokens whose likelihood is highest, and leave harder decision to later, exploiting features that can be extracted from structures already built on the sides of the attachment points. This leads though to an increase in complexity by a factor of *log n*. We tested the approach using the implementation provided by the authors[1], which only annotates unlabeled trees, and it achieved an Unlabeled Accuracy Score (UAS) of 89.01% on the Evalita 2011 test set, which is well below the UAS of our submission 93.73%.

We suggested a possible alternative, which would retain the linear complexity of a transition-based parser, consisting in delaying reduce decision until later portions of the parse tree have been built.

[1] `http://www.cs.bgu.ac.il/~yoavg/software/easyfirst/`

Acknowledgments. Partial support for this work has been provided by the PARLI Project (Portale per l'Accesso alle Risorse Linguistiche per l'Italiano – MIUR – PRIN 2008).

References

1. Aho, V., Ullman, J. D.: The Theory of Parsing, Translation and Compiling. Prentice-Hall Inc., Upper Saddle River (1972)
2. Attardi, G.: Experiments with a Multilanguage non-projective dependency parser. In: Proc. of the Tenth Conference on Computational Natural Language Learning (CoNLL-X 2006), pp. 166–170. ACL, Stroudsburg (2006)
3. Bosco, C., Mazzei, A.: The EVALITA Dependency Parsing Task: From 2007 to 2011. In: Magnini, B., Cutugno, F., Falcone, M., Pianta, E. (eds.) EVALITA 2012. LNCS, vol. 7689, pp. 1–12. Springer, Berlin (2011)
4. Attardi, G., Dell'Orletta, F.: Reverse Revision and Linear Tree Combination for Dependency Parsing. In: Proc. of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the ACL, Companion Volume: Short Papers (NAACL-Short 2009), pp. 261–264. ACL, Stroudsburg (2009)
5. Attardi, G., Dell'Orletta, F., Simi, M., Turian, J.: Accurate Dependency Parsing with a Stacked Multilayer Perceptron. In: Proc. of Workshop Evalita 2009 (2009) ISBN 978-88-903581-1-1
6. Attardi, G., Dei Rossi, S., Simi, M.: Dependency Parsing of Indian Languages with DeSR. In: Proc. of ICON 2010 NLP Tools Contest: Indian Language Dependency Parsing, Kharagpur, India, pp. 15–19 (2010)
7. Goldberg, Y., Elhadad, M.: An Efficient Algorithm for Easy-First Non-Directional Dependency Parsing. In: Proc. of the North American Chapter of the Association for Computational Linguistics Conference, NAACL 2010 (2010)
8. Grella, M., Nicola, M., Christen, D.: Experiments with a Constraint-based Dependency Parser. In: Proc. of EVALITA 2011. LNCS. Springer, Berlin (2012)
9. Grella, M.: Personal Communication (2011)
10. Nivre, J., Scholz, M.: Deterministic Dependency Parsing of English Text. In: Proc. of COLING 2004, Geneva, Switzerland, pp. 64–70 (2004)
11. Nivre, J., Hall, J., Kübler, S., McDonald, R., Nilsson, J., Riedel, S., Yuret, D.: The CoNLL 2007 shared task on dependency parsing. In: Proc. of the CoNLL 2007 Shared Task. Joint Conf. on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, EMNLP-CoNLL (2007)
12. Surdeanu, M., Manning, C. D.: Ensemble Models for Dependency Parsing: Cheap and Good? In: Proc. of the North American Chapter of the Association for Computational Linguistics Conference, NAACL 2010 (2010)
13. Yamada, H., Matsumoto, Y.: Statistical Dependency Analysis with Support Vector Machines. In: Proc. of the 8th International Workshop on Parsing Technologies (IWPT), pp. 195–206 (2003)
14. Zhang, Y., Nivre, J.: Transition-based dependency parsing with rich non-local features. In: Proc. of the 49th Annual Meeting of the ACL: Human Language Technology, Portland, Oregon, USA, pp. 188–193. ACL, Stroudsburg (2011)