MICA: A Probabilistic Dependency Parser Based on Tree Insertion Grammars

Application Note

Srinivas Bangalore
AT&T Labs – Research
Florham Park, NJ, USA
srini@research.att.com

Pierre Boulllier
INRIA
Rocquencourt, France
Pierre.Boullier@inria.fr

Alexis Nasr
LIF – CNRS – Aix Marseille Université
Marseille, France
alexis.nasr@lif.univ-mrs.fr

Owen Rambow
CCLS, Columbia Univserity
New York, NY, USA
rambow@ccls.columbia.edu

Benoît Sagot
INRIA
Rocquencourt, France
benoit.sagot@inria.fr

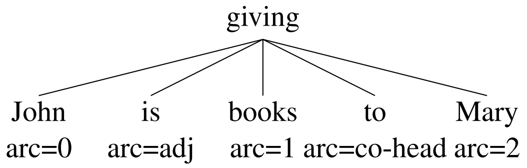
Abstract

MICA is a dependency parser which returns deep dependency representations, is fast, has state-of-the-art performance, and is freely available.

1 Overview

This application note presents a freely available parser, MICA (Marseille-INRIA-Columbia-AT&T).[1] MICA has several key characteristics that make it appealing to researchers in NLP who need an off-the-shelf parser.

• MICA returns a deep dependency parse, in which dependency is defined in terms of lexical predicate-argument structure, not in terms of surface-syntactic features such as subject-verb agreement. Function words such as auxiliaries and determiners depend on their lexical head, and strongly governed prepositions (such as *to* for *give*) are treated as co-heads rather than as syntactic heads in their own right. For example, *John is giving books to Mary* gets the following analysis (the arc label is on the terminal).



The arc labels for the three arguments *John*, *books*, and *Mary* do not change when the sentence is passivized or *Mary* undergoes dative shift.

• MICA is based on an explicit phrase-structure tree grammar extracted from the Penn Treebank. Therefore, MICA can associate dependency parses with rich linguistic information such as voice, the presence of empty subjects (PRO), *wh*-movement, and whether a verb heads a relative clause.

• MICA is fast (450 words per second plus 6 seconds initialization on a standard high-end machine on sentences with fewer than 200 words) and has state-of-the-art performance (87.6% unlabeled dependency accuracy, see Section 5).

• MICA consists of two processes: the supertagger, which associates tags representing rich syntactic information with the input word sequence, and the actual parser, which derives the syntactic structure from the n-best chosen supertags. Only the supertagger uses lexical information, the parser only sees the supertag hypotheses.

• MICA returns n-best parses for arbitrary n; parse trees are associated with probabilities. A packed forest can also be returned.

• MICA is freely available[2], easy to install under Linux, and easy to use. (Input is one sentence per line with no special tokenization required.)

There is an enormous amount of related work, and we can mention only the most salient, given space constraints. Our parser is very similar to the work of (Shen and Joshi, 2005). They do not employ a supertagging step, and we do not restrict our trees to spinal projections. Other parsers using supertagging include the LDA of Bangalore and Joshi (1999), the CCG-based parser of Clark and Curran (2004), and the constraint-based approach of Wang

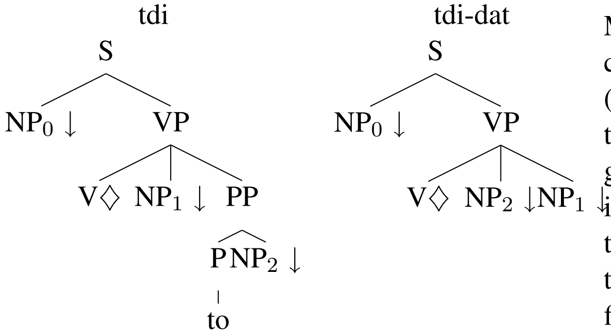
[1] We would like to thank Ryan Roth for contributing the MALT data.

[2] http://www1.ccls.columbia.edu/~rambow/mica.html

and Harper (2004). Widely used dependency parsers which generate deep dependency representations include Minipar (Lin, 1994), which uses a declarative grammar, and the Stanford parser (Levy and Manning, 2004), which performs a conversion from a standard phrase-structure parse. All of these systems generate dependency structures which are slightly different from MICA's, so that direct comparison is difficult. For comparison purposes, we therefore use the MALT parser generator (Nivre et al., 2004), which allows us to train a dependency parser on our own dependency structures. MALT has been among the top performers in the CoNLL dependency parsing competitions.

2 Supertags and Supertagging

Supertags are elementary trees of a lexicalized tree grammar such as a Tree-Adjoining Grammar (TAG) (Joshi, 1987). Unlike context-free grammar rules which are single level trees, supertags are multi-level trees which encapsulate both predicate-argument structure of the anchor lexeme (by including nodes at which its arguments must substitute) and morpho-syntactic constraints such as subject-verb agreement within the supertag associated with the anchor. There are a number of supertags for each lexeme to account for the different syntactic transformations (relative clause, *wh*-question, passivization etc.). For example, the verb *give* will be associated with at least these two trees, which we will call tdi and tdi-dat. (There are also many other trees.)



Supertagging is the task of disambiguating among the set of supertags associated with each word in a sentence, given the context of the sentence. In order to arrive at a complete parse, the only step remaining after supertagging is establishing the attachments among the supertags. Hence the result of supertagging is termed as an "almost parse" (Bangalore and Joshi, 1999).

The set of supertags is derived from the Penn Treebank using the approach of Chen (2001). This extraction procedure results in a supertag set of 4,727 supertags and about one million words of supertag annotated corpus. We use 950,028 annotated words for training (Sections 02-21) and 46,451 (Section 00) annotated words for testing in our experiments. We estimate the probability of a tag sequence directly as in discriminative classification approaches. In such approaches, the context of the word being supertagged is encoded as features for the classifier. Given the large scale multiclass labeling nature of the supertagging task, we train supertagging models as one-vs-rest binary classification problems. Detailed supertagging experiment results are reported in (Bangalore et al., 2005) which we summarize here. We use the lexical, part-of-speech attributes from the left and right context in a 6-word window and the lexical, orthographic (e.g. capitalization, prefix, suffix, digit) and part-of-speech attributes of the word being supertagged. Crucially, this set does not use the supertags for the words in the history. Thus during decoding the supertag assignment is done locally and does not need a dynamic programming search. We trained a Maxent model with such features using the labeled data set mentioned above and achieve an error rate of 11.48% on the test set.

3 Grammars and Models

MICA grammars are extracted in a three steps process. In a first step, a Tree Insertion Grammar (TIG) (Schabes and Waters, 1995) is extracted from the treebank, along with a table of counts. This is the grammar that is used for supertagging, as described in Section 2. In a second step, the TIG and the count table are used to build a PCFG. During the last step, the PCFG is "specialized" in order to model more finely some lexico-syntactic phenomena. The second and third steps are discussed in this section.

The extracted TIG is transformed into a PCFG which generates strings of supertags as follows. Initial elementary trees (which are substituted) yield rules whose left hand side is the root category of the elementary tree. Left (respectively right) auxiliary trees (the trees for which the foot node is the left (resp. right) daughter of the root) give birth to rules whose left-hand side is of the form X_{l} (resp. X_{r}), where X is the root category of the elementary tree. The right hand side of each rule is built during a top down traversal of the corresponding elementary tree. For every node of the tree visited, a new symbol is added to the right hand side of rule, from left to right, as follows:

- The anchor of the elementary tree adds the supertag (i.e., the name of the tree), which is a terminal symbol, to the context-free rule.
- A substitution node in the elementary tree adds its nonterminal symbol to the context-free rule.
- A interior node in the elementary tree at which adjunction may occur adds to the context-free rule the nonterminal symbol X_{r}^{*} or X_{l}^{*}, where X is the node's nonterminal symbol, and l (resp. r) indicates whether it is a left (resp. right) adjunction. Each interior node is visited twice, the first time from the left, and then from the right. A set of non-lexicalized rules (i.e., rules that do not generate a terminal symbol) allow us to generate zero or more trees anchored by X_{l} from the symbol X_{l}^{*}. No adjunction, the first adjunction, and the second adjunction are modeled explicitly in the grammar and the associated probabilistic model, while the third and all subsequent adjunctions are modeled together.

This conversion method is basically the same as that presented in (Schabes and Waters, 1995), except that our PCFG models multiple adjunctions at the same node by positions (a concern Schabes and Waters (1995) do not share, of course). Our PCFG construction differs from that of Hwa (2001) in that she does not allow multiple adjunction at one node (Schabes and Shieber, 1994) (which we do since we are interested in the derivation structure as a representation of linguistic dependency). For more information about the positional model of adjunction and a discussion of an alternate model, the "bigram model", see (Nasr and Rambow, 2006).

Tree tdi from Section 2 gives rise to the following rule (where tdi and tCO are terminal symbols and the rest are nonterminals): $\mathrm{S} \rightarrow \mathrm{S}_{\mathrm{l}}^{*}$ NP $\mathrm{VP}_{\mathrm{l}}^{*}$ $\mathrm{V}_{\mathrm{l}}^{*}$ tdi $\mathrm{V}_{\mathrm{r}}^{*}$ NP $\mathrm{PP}_{\mathrm{l}}^{*}$ $\mathrm{P}_{\mathrm{l}}^{*}$ tCO $\mathrm{P}_{\mathrm{r}}^{*}$ NP $\mathrm{PP}_{\mathrm{r}}^{*}$ $\mathrm{VP}_{\mathrm{r}}^{*}$ $\mathrm{S}_{\mathrm{r}}^{*}$

The probabilities of the PCFG rules are estimated using maximum likelihood. The probabilistic model refers only to supertag names, not to words. In the basic model, the probability of the adjunction or substitution of an elementary tree (the daughter) in another elementary tree (the mother) only depends on the nonterminal, and does not depend on the mother nor on the node on which the attachment is performed in the mother elementary tree. It is well known that such a dependency is important for an adequate probabilistic modelling of syntax. In order to introduce such a dependency, we condition an attachment on the mother and on the node on which the attachment is performed, an operation that we call mother specialization. Mother specialization is performed by adding to all nonterminals the name of the mother and the address of a node. The specialization of a grammar increase vastly the number of symbols and rules and provoke severe data sparseness problems, this is why only a subset of the symbols are specialized.

4 Parser

SYNTAX (Boullier and Deschamp, 1988) is a system used to generate lexical and syntactic analyzers (parsers) (both deterministic and non-deterministic) for all kind of context-free grammars (CFGs) as well as some classes of contextual grammars. It has been under development at INRIA for several decades. SYNTAX handles most classes of deterministic (unambiguous) grammars (LR, LALR, RLR) as well as general context-free grammars. The non-deterministic features include, among others, an Earley-like parser generator used for natural language processing (Boullier, 2003).

Like most SYNTAX Earley-like parsers, the architecture of MICA's PCFG-based parser is the following:

- The Earley-like parser proper computes a shared parse forest that represents in a factorized (polynomial) way *all* possible parse trees according to the underlying (non-probabilistic) CFG that represents the TIG;
- Filtering and/or decoration modules are applied on the shared parse forest; in MICA's case, an n-best module is applied, followed by a dependency extractor that relies on the TIG structure of the CFG.

The Earley-like parser relies on Earley's algorithm (Earley, 1970). However, several optimizations have been applied, including guiding techniques (Boullier, 2003), extensive static (offline) computations over the grammar, and efficient data structures. Moreover, Earley's algorithm has been extended so as to handle input DAGs (and not only sequences of forms). A particular effort has been made to handle huge grammars (over 1 million symbol occurrences in the grammar), thanks to advanced dynamic lexicalization techniques (Boullier and Sagot, 2007). The resulting efficiency is satisfying: with standard ambiguous NLP grammars, huge shared parse forest (over 10^{10} trees) are often generated in a few dozens of milliseconds.

Within MICA, the first module that is applied on top of the shared parse forest is Syntax's n-best module. This module adapts and implements the algorithm of (Huang and Chiang, 2005) for efficient n-best trees extraction from a shared parse forest. In practice, and within the current version of MICA, this module is usually used with $n = 1$, which identifies the optimal tree w.r.t. the probabilistic model embedded in the original PCFG; other values can also be used. Once the n-best trees have been extracted, the dependency extractor module transforms each of these trees into a dependency tree, by exploiting the fact that the CFG used for parsing has been built from a TIG.

5 Evaluation

We compare MICA to the MALT parser. Both parsers are trained on sections 02-21 of our dependency version of the WSJ PennTreebank, and tested on Section 00, not counting true punctuation. "Predicted" refers to tags (PTB-tagset POS and supertags) predicted by our taggers; "Gold" refers to the gold POS and supertags. We tested MALT using only POS tags (MALT-POS), and POS tags as well as 1-best supertags (MALT-all). We provide unlabeled ("Un") and labeled ("Lb") dependency accuracy (%). As we can see, the predicted supertags do not help MALT. MALT is significantly slower than MICA, running at about 30 words a second (MICA: 450 words a second).

	MICA		MALT-POS		MALT-all	
	Pred	Gold	Pred	Gold	Pred	Gold
Lb	85.8	97.3	86.9	87.4	86.8	96.9
Un	87.6	97.6	88.9	89.3	88.5	97.2

References

Srinivas Bangalore and Aravind Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–266.

Srinivas Bangalore, Patrick Haffner, and Gaël Emami. 2005. Factoring global inference by enriching local representations. Technical report, AT&T Labs – Reserach.

Pierre Boullier and Philippe Deschamp. 1988. Le système Syntax$^{\text{TM}}$ – manuel d'utilisation et de mise en œuvre sous Unix$^{\text{TM}}$. http://syntax.gforge.inria.fr/syntax3.8-manual.pdf.

Pierre Boullier and Benoît Sagot. 2007. Are very large grammars computationnaly tractable? In *Proceedings of IWPT'07*, Prague, Czech Republic.

Pierre Boullier. 2003. Guided Earley parsing. In *Proceedings of the 7th International Workshop on =20 Parsing Technologies*, pages 43–54, Nancy, France.

John Chen. 2001. *Towards Efficient Statistical Parsing Using Lexicalized Grammatical Information*. Ph.D. thesis, University of Delaware.

Stephen Clark and James R. Curran. 2004. Parsing the WSJ using CCG and log-linear models. In *ACL'04*.

Jay Earley. 1970. An efficient context-free parsing algorithm. *Communication of the ACM*, 13(2):94–102.

Liang Huang and David Chiang. 2005. Better k-best parsing. In *Proceedings of IWPT'05*, Vancouver, Canada.

Rebecca Hwa. 2001. *Learning Probabilistic Lexicalized Grammars for Natural Language Processing*. Ph.D. thesis, Harvard University.

Aravind K. Joshi. 1987. An introduction to Tree Adjoining Grammars. In A. Manaster-Ramer, editor, *Mathematics of Language*. John Benjamins, Amsterdam.

Roger Levy and Christopher Manning. 2004. Deep dependencies from context-free statistical parsers: Correcting the surface dependency approximation. In *ACL'04*.

Dekang Lin. 1994. PRINCIPAR—an efficient, broad-coverage, principle-based parser. In *Coling'94*.

Alexis Nasr and Owen Rambow. 2006. Parsing with lexicalized probabilistic recursive transition networks. In *Finite-State Methods and Natural Language Processing*, Springer Verlag Lecture Notes in Commputer Science.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2004. Memory-based dependency parsing. In *CoNLL-2004*.

Yves Schabes and Stuart Shieber. 1994. An alternative conception of tree-adjoining derivation. *Computational Linguistics*, 1(20):91–124.

Yves Schabes and Richard C. Waters. 1995. Tree Insertion Grammar. *Computational Linguistics*, 21(4).

Libin Shen and Aravind Joshi. 2005. Incremental ltag parsing. In *HLT-EMNLP'05*.

Wen Wang and Mary P. Harper. 2004. A statistical constraint dependency grammar (CDG) parser. In *Proceedings of the ACL Workshop on Incremental Parsing*.